The learning of algorithms and architectures

Mikhail Khodak

CMU-CS-24-145
August 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Maria-Florina Balcan, Co-Chair
Ameet Talwalkar, Co-Chair
Tom Mitchell
Peter Bartlett (University of California-Berkeley)
Piotr Indyk (Massachusetts Institute of Technology)
Alexander Smola (Boson AI)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by a TCS Presidential Fellowship, a Facebook PhD Fellowship, a Smola Bloomberg gift, JP Morgan, Two Sigma, Meta under award number PO70000298580, the Defense Advanced Research Projects Agency under award number HR00112020003, the National Science Foundation with UCLA under award number PO0145GVA403, and the National Science Foundation under award numbers CCF-1535967, IIS-1901403, CCF-1910321, and SES-1919453.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: machine learning, meta-learning, algorithms with predictions, architecture search

семье

Abstract

How should we design the algorithms we run and the architectures we learn? Several high-impact areas of computing have begun to automate these procedures using machine learning (ML), reducing the need for human effort by using our expanding amount of data and compute. This thesis uses ideas from ML, algorithm design, and optimization to advance our understanding of these areas of data-driven computing—meta-learning, algorithms with predictions, and architecture search—and to translate the resulting methodologies into state-of-the-art implementations.

- In **meta-learning**, which uses ML itself to improve ML algorithms by learning across many learning tasks, we introduce ARUBA, a framework for designing and analyzing meta-learning methods. Our analysis yields the first guarantees for gradient-based meta-learning, showing how such methods improve performance based upon quantifiable measures of similarity between learning tasks. We use ARUBA to extend the practical impact of meta-learning to new areas of ML, including to learning with partial feedback and to federated learning; in the latter case, we introduce FedEx, a new state-of-the-art method for tuning federated optimizers, which train models on networks of distributed heterogeneous datasets such as mobile devices and hospital records.
- We build upon the success of ARUBA by taking its core approach—the optimization of surrogate loss functions approximating algorithmic objectives—and extending it beyond *learning* algorithms to show learning guarantees for **algorithms with predictions**, which are algorithms that take advantage of ML predictions about their instances; in particular, we show the first learning-theoretic guarantees for predictions that depend on the instance the algorithm is run on, a crucial property for practical applications. Our framework again serves as an algorithm design tool, which we use to build the first algorithms with predictions for mechanisms that release (differentially) private statistics about sensitive datasets and for linear system solvers; in the latter case, we design learning algorithms that, under natural structural assumptions, can learn to make *instance-optimal* predictions.
- Lastly, this thesis addresses the problem of finding neural network architectures to train on specific learning tasks, or **architecture search**, where we make progress towards understanding the optimization and generalization properties of weight-sharing, a dominant heuristic used throughout the field. We then extend weight-sharing to design new search spaces based around neural *operations* that allow for the automated discovery of truly novel architectures from data; the culmination of this effort is DASH, a method that efficiently finds architectures that outperform human expert-designed neural architectures on the majority of diverse tasks we test.

Acknowledgments

The last six years have left me with no shortage of people to be grateful to. Thank you to my wonderful advisers, to their lively labs, to my supportive committee, and to CSD's excellent administrative staff. Thank you to the kind mentors I have been lucky to know through my internships, collaborations, and travels. Thank you to the friends I made at CMU: you have forever biased my view of Pittsburgh with the rosy color of nostalgia.

This document may have taken six years to write, but the privilege of spending that time on this effort was made possible by three decades of support and kindness. Thank you to my patient mentors in the Princeton math and CS departments, at PPPL, and at LLNL. Thank you to my lifelong friends from Forbes, the OGC, and beyond. Thank you above all to Shir and to my family: my mother, father, sister, and grandparents.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 0

Introduction

Classically, algorithm design and machine learning (ML) are studied on individual, well-defined tasks, such as a problem (e.g. a linear system) to be solved or a model class (e.g. a specific neural architecture) to be learned. In the real world, computations and datasets do not exist in a vacuum, with practitioners specifying algorithms and architectures via inductive biases, prior experience, and increasingly through *learning* from related tasks. This last approach *automatically* improves the algorithms we run and the architectures we use by using experience on previous instances to obtain better performance on future instances. It encompasses many of the most important paradigms in modern data-driven computing, including the meta-learning or large-scale pretraining of gradient descent initializations on heterogeneous tasks or corpora, algorithms that take advantage of learned predictions about their instances, and the automated discovery of neural architectures to train on specific tasks. The goal of this thesis is to establish learning from *multiple* tasks, datasets, and computations on firm theoretical and practical foundations, allowing scientists and engineers to confidently use the resulting algorithms to power new innovations.

At the technical level, we focus on the learning of *two* types of objects: algorithms—both regular algorithms and learning algorithms—and neural network architectures. This is done by "meta-learning" parameters encoding algorithmic or architectural settings while minimizing appropriate cost measures across multiple learning tasks or problem instances. However, thematically the contributions in the thesis are split into the following *three* parts:

1. **Meta-learning:** In many applications, we want to learn a good learning algorithm—e.g. find a good gradient descent initialization—from a large collection of heterogeneous datasets or tasks; for example, large language models (LLMs) are pretrained on big, multi-distribution corpora before fine-tuning on target data. Understanding such settings requires going beyond the single-distribution paradigm of classical ML. We show some of the first guarantees for *gradient-based* meta-learning, a major approach in this area numerous applications. Our theory describes task similarity conditions under which learning from multiple tasks is useful and prescribes algorithms that can exploit this similarity. Since the publication of the initial results of the thesis, our framework has been directly built upon theoretically by scientists in diverse areas such as algorithmic game theory and reinforcement learning. Within the thesis itself, we use our framework to design a new state-of-the-art method for tuning hyperparameters in federated learning (FL) that has been found to consistently improve regular tuners in both our own and in recent independent evaluations.

2. **Algorithms with predictions:** Also known as *learning-augmented algorithms*, this rapidly growing subfield of theoretical CS designs algorithms whose performance can be improved by learned predictions of their outputs. It is a leading way of analyzing algorithms beyond worst-case instances and has had a significant practical impact in areas such as databases and energy systems. Our work provides the first systematic understanding of the critical *learning* aspect of learning-augmented algorithms, introducing a unified way to determine learnability and in doing so dramatically improving several existing theoretical results while proving many new ones. This thesis also expands the scope of learning-augmented algorithms beyond online and graph algorithms, including to privacy-preserving statistics and to scientific computing.
3. **Architecture search:** The third focus area of this thesis is on the automated discovery of good neural architectures, with an emphasis on neural architecture search (NAS) methods that employ the *weight-sharing* heuristic. We develop the first theoretical understanding of the optimization and generalization properties of this technique by conducting a mathematical analysis of different NAS objectives. Then we deploy weight-sharing to design novel search spaces, and associated search algorithms, for finding truly novel neural architectures that work for diverse data modalities beyond vision, text, and audio. This effort culminates in a new NAS method whose discovered architectures tend to outperform human expert-designed architectures on the latest benchmark in the field.

In summary, this thesis advances our theoretical understanding of training on multi-distribution data, which underlies everything from foundation models powering the latest breakthrough artificial intelligence (AI) systems to the go-to methods in distributed learning. It also provides new frameworks that guide the design and analysis of state-of-the-art paradigms in algorithm design (algorithms with predictions) and neural architecture search (weight-sharing). In applications, the thesis combines these insights with domain-specific knowledge to develop algorithms for distributed (federated) learning on heterogeneous data, incorporating ML into statistics and scientific computing, and automating the application of ML to understudied modalities.

In the rest of this chapter, we first detail the contributions we make towards the learning of algorithms (Section 0.1) and architectures (Section 0.2), and how the individual results are connected. We then conclude with an organizational overview of the rest of the thesis (Section 0.3).

0.1 Learning to parameterize algorithms

Learning to set algorithmic parameters is an important use-case of ML that encompasses many overlapping areas such as multi-task and meta-learning, personalized federated learning, algorithms with predictions, amortized optimization, and data-driven algorithm design. Here the data is a collection of learning tasks or computational instances, and the goal is to reduce the cost of running some parameterized algorithm on them by learning a good parameter to use. Designing and analyzing algorithms that learn to set the parameters for other algorithms, or *meta-algorithms*, is challenging because of the complicated way in which the performance being optimized—e.g. an algorithm's runtime or regret—depends on the parameter used.

A key insight that drives the first two parts of this thesis is that we often do not need to work with the exact performance metric and can instead use a good approximation to achieve mean-

ingful results. The closest analogy is that in (single-task) supervised classification we rarely optimize the (nonconvex) classification loss and instead use *surrogate loss functions*. Similarly, an algorithm's performance can also often be approximated by a simple function of (a) the parameters to be set by the meta-algorithm and (b) the dataset or instance the algorithm will be run on. For example, the popular stochastic gradient descent (SGD) algorithm provably performs well on an optimization problem if the distance from the initialization (a parameter) to the (instance-specific) optimum is small.

We now detail the consequences of this insight for two important areas at the intersection of ML and algorithms: meta-learning and algorithms with predictions. The resulting methods, which comprise the first two parts of the thesis, enjoy provable guarantees that show improved performance as a function of similarity between problem instances. At the same time, they are practical to apply in large-scale settings beyond the regimes in which we study them, such as when the tasks involve tuning diverse hyperparameters or solving linear systems.

0.1.1 Meta-learning

In the first part of this thesis we develop the idea of optimizing such simple functions—i.e. using them as *surrogate algorithmic losses*—into a framework called **ARUBA** for designing and analyzing meta-learning algorithms, i.e. meta-algorithms specifically for *learning* algorithms. Crucially, the performance of learning algorithms run using the parameters set by our meta-learners is provably better than comparable single-task learning methods if the tasks are similar in a natural, algorithm-specific way. For example, gradient descent with a meta-learned initialization performs well if the tasks' optima are close in terms of average Euclidean distance.

ARUBA is applicable in numerous settings where the goal is to alleviate the lack of data in individual learning tasks using data from many related tasks. This collection or "meta-dataset" of tasks is used by the meta-learner to set the parameters of a "within-task" or "base learner" algorithm to be run on individual tasks; for example, a dataset of mobile device data can be used to meta-learn an initialization for SGD that yields a personalized language model when fine-tuned on the data of a new user. Such approaches have found important applications in areas such as distributed (federated) learning, computer vision, reinforcement learning, and the pretraining and fine-tuning of LLMs. In this thesis we show the utility of ARUBA in the following settings:

1. **Gradient-based meta-learning:** Many modern ML algorithms, including in deep learning, are adaptations of methods for online convex optimization (OCO), where the learner is faced with sequentially choosing good parameters for a sequence of convex loss functions. As a result, a popular paradigm for multi-task training of neural network initializations known as *gradient-based meta-learning* is implicitly meta-learning variants of OCO algorithms. Thus, by using ARUBA to study the meta-learning of online mirror descent family, a large family of OCO algorithms, we show the first upper and lower bounds on the performance of gradient-based meta-learning algorithms. As before, these guarantees improve with a natural notion of task similarity between learning tasks; specifically, task-averaged performance is good if the optimal parameters of different tasks are close together.
2. **Meta-learning of online learners:** Following the study of gradient-based meta-learning, we proceed to show that ARUBA is applicable even when the assumptions of online convex

optimization are relaxed. In particular, we use it to design meta-learning algorithms and show guarantees in the non-adversarial (statistical) setting, the partial information (bandit) setting, and the (nonconvex) piecewise-Lipschitz setting. As before, these results show improved performance with setting-specific notions of task similarity, e.g. in the multi-armed bandit setting we show that the average regret across tasks will have a logarithmic dependence on the number of arms—unlike the square-root dependence that is minimax-optimal in the single-task setting—so long as a constant number of unknown arms is ever optimal on any task.

3. **Federated learning:** Lastly, we exploit the equivalence between popular methods for gradient-based meta-learning and federated learning—training models across a heterogeneous network of devices—to design **FedEx**, an algorithm for tuning the hyperparameters of a large class of federated learning methods. FedEx can tune all local hyperparameters, enjoys ARUBA-based provable guarantees for the case of learning the local step-size, and leads to significant improvement across three standard tasks in federated learning.

0.1.2 Algorithms with predictions

In the second part of the thesis, we extend the core idea of ARUBA—the optimizing of surrogate losses for algorithmic performance measures—beyond learning algorithms, demonstrating its potential use in any setting where we might hope to use learning to speed up or otherwise improve a computation. We work mainly in the paradigm of algorithms with predictions, a growing area of algorithm design where the goal is to use possibly imperfect predictors of the outcomes or optimal settings of an algorithm to reduce its cost. While standard analysis of algorithms characterizes performance in the worst or average case, in domains ranging from database systems to energy management we can realize substantial gains by augmenting methods with *learned predictions* about their instances. This has inspired a large body of theoretical work focused on quantifying improvement via prediction-dependent performance guarantees and designing methods that are *robust* to poor predictions. Such results can have a direct impact on important applications such as caching protocols, energy systems, and job scheduling.

This thesis makes two fundamental contributions to algorithms with predictions: (1) addressing the crucial question of *learning* and (2) extending the field's scope beyond its origins in online and graph problems. The first direction is important because, while the field had produced many useful algorithms with predictions, the question of where the predictions themselves come has not been systematically addressed. In practice, predictions often come from meta-algorithms trained by applying ML to algorithmic data, and so the question becomes whether and how these meta-algorithms can be efficiently learned. We observe that, just like for initialization-dependent learning algorithms in ARUBA, existing performance guarantees for learning-augmented algorithms can be *also* converted into surrogate losses. Distilling this approach into two steps—(1) proving an optimizable prediction-dependent performance bound and (2) applying online learning to minimize it across instances—yields a powerful tool for showing end-to-end guarantees for algorithms with predictions, i.e. results that address both how to use predictions and how to learn them. Because it focuses on surrogate loss functions amenable to optimization, the framework also leads to efficient and practical prediction-learning methods.

As summarized below, we use the idea of proving learnable performance bounds both to show learning-theoretic guarantees for existing algorithms with predictions and to expand the scope of learning-augmented algorithms to two new areas:

1. **Learning predictions:** We start by using ARUBA to systematically integrate learning into algorithms with predictions by taking advantage of existing bounds on the cost of parameterizable algorithms. Via a series of results on maximum-weight independent matching, online page migration, job scheduling, and ski rental, we show the first online learning results for this paradigm along with several new or improved sample complexity bounds. Importantly, our guarantees are the first learning guarantees for *instance-adaptive* parameterizations of algorithms, i.e. where instead of learning one fixed parameter to use on all instances we learn a policy mapping instances to customized parameters for them. This is a practically critical but theoretically understudied aspect of data-driven algorithms.
2. **Multi-dataset private statistics:** The algorithms with predictions paradigm can be viewed as a toolkit for deriving and analyzing data-driven methods, with our learning-theoretic framework being an important new addition to existing capabilities such as robustness-consistency analysis. We demonstrate how the utility of this view via the design and analysis of algorithms with predictions in an entirely new area: differentially private (DP) statistics. In this field, the goal is to release information about sensitive datasets while protecting the privacy of individuals appearing in it, generally by injecting noisy. Here we study learning-augmented procedures for multiple quantile release, covariance estimation, and data release, all of them endowed with both robustness guarantees and efficient learning procedures derived by optimizing well-chosen surrogate losses on external datasets. Along the way, we introduce the first algorithm for DP quantile release that does not depend on boundedness assumption and show the usefulness of both our robustness and learning analysis in several multi-dataset settings. Our results yielded substantial reductions in the error of privately released statistics, especially at high privacy levels.
3. **Learning to solve linear systems:** Lastly, we study the data-driven solving of linear systems, which has important applications in scientific computing problems such as partial differential equations (PDEs). We examine the problem from both the algorithms with predictions perspective and from that of data-driven algorithm design, designing bandit algorithms for setting good relaxation parameters and preconditioners. We also show under natural structural and smoothness assumptions that we can learn an *instance-optimal* policy for setting algorithmic parameters. When used to speed up a two-dimensional heat simulation over a fine-grained mesh our algorithms lead to significant—up to almost three-fold—wallclock improvements over strong baselines.

0.2 Discovering effective neural architectures

While large-scale neural networks have achieved incredible success in recent years, progress has been distributed very unevenly among different domains. Methodological development has focused on a set of well-studied domains—vision, text, and audio—and data and compute demands have made it difficult for academic and some industry researchers to apply state-of-the-art ML.

This has led to important research directions aimed at making such models more efficient and widely applicable, such as neural architecture search (NAS) and more broadly automated machine learning (AutoML). However, prior to the work in this thesis the design and evaluation of NAS methods has itself focused on well-studied data modalities, especially natural images, and the resulting algorithms depend on poorly understood heuristics such as weight-sharing.

This thesis introduces novel search spaces and parameterizations that (1) enable more effective gradient-based NAS algorithms, (2) expand our empirical understanding of the weight-sharing and bilevel optimization approaches to NAS, and (3) yield effective architectures on diverse tasks in the natural sciences, healthcare, and beyond. Specifically, we study the following search spaces and associated algorithms:

1. **Operation simplices:** Most differentiable NAS algorithms work by determining which of a finite set of operations—e.g. identity, convolution, or pooling—to assign to an edge in a computational graph. Usually this is done by continuously parameterizing each choice using a real number and using a softmax selection, leading to poor optimization and non-sparse discretization. We propose **GAEA**, a method that uses a simplex re-parameterization of this search space and exponentiated gradient to traverse it, yielding provable convergence guarantees, empirically faster recovery of sparse architecture parameters, and improved performance on standard NAS benchmarks.
2. **Feature map selection:** Because they are entangled with optimization of deep neural networks, aspects of NAS such as weight-sharing and bilevel optimization are poorly understood. We propose feature map selection as a simple setting for studying NAS and show that empirically it also benefits from weight-sharing. We also provide theoretical evidence that bilevel optimization helps in this setting.
3. **Expressive diagonalization:** NAS operation spaces, including the ones studied above, are generally small sets of a few named operations, preventing the discovery of truly novel architectures. We propose **XD-operations**, which dramatically expand the operation space by parameterizing the discrete Fourier transforms (DFTs) of the convolution operations diagonalization to take on continuous matrix values. The resulting search space provably contains many important operations, including all kinds of convolutions, transposed convolutions, pooling operations, Fourier neural operators (FNOs), graph convolutions, and many more. Empirically XD-operations outperform standard NAS operation spaces on permuted image classification, PDE solving, protein folding, and music modeling tasks.
4. **Efficient diagonalization:** While XD-operations are both expressive and theoretically efficient, they face significant memory and computation challenges when applied to practical tasks. We find that on tasks involving high-dimensional unstructured data such as images, it is often sufficient to take a simple convolutional neural network (CNN) such as a Wide ResNet (WRN) and search for better kernel sizes and dilation rates for its convolutional filters. The resulting method—**DASH**—outperforms expert-designed architectures on seven out of ten evaluated tasks, spanning diverse applications and dimensionalities from cosmology to genomics.

These algorithms and search spaces point the way towards automated ML methods that can truly be applied out-of-the-box on a wide array of applications, especially understudied ones beyond vision and text.

0.3 Organization and contributions

The thesis is organized into three parts: Part I introduces ARUBA and its applications in meta-learning such as personalized federated learning, Part II deals with its extension to algorithms with predictions and new directions of data-driven algorithms, and Part III covers architecture search. Each part has an introductory chapter giving an overview of the topic, the contributions to it in this thesis, and a discussion of recent related work and future directions. This is followed by a chapter on the core theoretical contributions (often including empirical demonstrations) and one or more chapters on (often theory-driven) applications. The appendix in the introductory chapter of each part provides background information for it.

The work presented in this thesis is largely contained in existing publications, cited here as well as in the first footnotes of the relevant chapters [Khodak et al., 2019b,a, Balcan et al., 2021b, Khodak et al., 2023b, 2021, 2022, 2023a, Amin et al., 2023, Khodak et al., 2024, Li et al., 2021a, Khodak et al., 2020, Roberts et al., 2021, Shen et al., 2022]. Significant content will be reused from these publications, especially in chapters devoted to the main results. This thesis does provide a unified overview of many of the theoretical results, corrects some minor errors in the original works, and situates the contributions in the context of more recent developments.

Part I

Meta-learning

Chapter 1

Overview

Meta-learning, or *learning-to-learn* (LTL) [Thrun and Pratt, 1998], has re-emerged as an important direction for developing algorithms for multi-task learning, dynamic environments, and federated settings. By using the data of numerous training tasks, meta-learning methods seek to perform well on new, potentially related test tasks without using many samples. Successful modern approaches have also focused on exploiting the capabilities of deep neural networks, whether by learning multi-task embeddings passed to simple classifiers [Snell et al., 2017] or by neural control of optimization algorithms [Ravi and Larochelle, 2017]. Because of its simplicity and flexibility, a common approach is *parameter-transfer*, where all tasks use the same class of Θ-parameterized functions $f_{\boldsymbol{\theta}} : \mathcal{X} \mapsto \mathcal{Y}$; often a shared model $\boldsymbol{\phi} \in \Theta$ is learned that is used to train within-task models. In *gradient-based meta-learning* (GBML) algorithms such as MAML [Finn et al., 2017], $\boldsymbol{\phi}$ is a meta-initialization for a gradient descent method over samples from a new task. GBML is used in a variety of LTL domains such as vision [Li et al., 2017, Nichol et al., 2018, Kim et al., 2018], federated learning [Chen et al., 2018a], and robotics [Duan et al., 2017, Al-Shedivat et al., 2018]. Its simplicity also raises many practical and theoretical questions about the task similarity it can exploit and the settings in which it can succeed.

The first part of this thesis deals with theoretically understanding algorithms used in modern meta-learning, especially GBML, and applying the resulting insights to improve these methods. In this chapter we review existing theoretical analyses of meta-learning, describe the contributions of this thesis at a high level, and discuss recent developments and future work in the field In Chapter 2 we introduce the core contribution: a theoretical framework called ARUBA that both (a) provides insight into how meta-learning allows learning algorithms to take advantage of task similarity and (b) guides the design of meta-learning algorithm via the idea of applying off-the-shelf optimization techniques to surrogate bounds on performance. After introducing this framework, we demonstrate it in a variety of learning-theoretic settings, showcasing its widespread applicability. Chapter 3 is then dedicated to FedEx, a method for federated hyperparameter tuning that can be understood in-part as an instantiation of ARUBA as well.

1.1 Literature

The statistical analysis of LTL was formalized by Baxter [2000]. Several works have built upon this theory for modern LTL, such as via a PAC-Bayesian perspective [Amit and Meir, 2018] or by learning the kernel for ridge regression [Denevi et al., 2018]. However, much effort has also been devoted to the online setting, often through the framework of lifelong learning [Pentina and Lampert, 2014, Balcan et al., 2015, Alquier et al., 2017]. Alquier et al. [2017] consider a many-task notion of regret similar to the one we study in order to learn a shared data representation, although our algorithms are much more practical. Recently, Bullins et al. [2019] developed an efficient online approach to learning a linear data embedding, but such a setting is distinct from GBML and more closely related to popular shared-representation methods such as ProtoNets [Snell et al., 2017]. Nevertheless, our approach does strongly rely on online learning through the study of data-dependent regret-upper-bounds, which has a long history of use in deriving adaptive single-task methods [McMahan and Streeter, 2010, Duchi et al., 2011]; however, in meta-learning there is typically not enough data to adapt to without considering multi-task data.

The theoretical study of GBML was initiated with an expressivity result shown for MAML by Finn and Levine [2018], proving that the meta-learner can approximate any permutation-invariant learner given enough data and a specific neural architecture. Under strong-convexity and (high-order) smoothness assumptions and using a fixed learning rate, Finn et al. [2019] show that the MAML meta-initialization is learnable. In contrast to these efforts, Denevi et al. [2019] focus on providing finite-sample meta-test-time performance guarantees in the convex setting. Our work improves upon these analyses by considering the case when the learning rate, a proxy for the task similarity, is not known beforehand as in Finn et al. [2019] and Denevi et al. [2019] but must be learned online. Furthermore, ARUBA results in guarantees that can handle more sophisticated and dynamic notions of task similarity and in certain settings can provide better statistical guarantees.

1.2 Contributions

The meta-learning portion of the thesis consists of a theoretical chapter (2) on the ARUBA framework and its applications to different settings learning-to-learn followed by an empirical chapter (3) dedicated to the FedEx method for federated hyperparameter tuning. Chapter 2 is mainly dedicated to learning-theoretic applications, with a few empirical demonstrations in simple settings in support of them. In contrast, the contribution in Chapter 3 is a practical method that is theoretically supported by an analysis enabled by the ARUBA framework. We now give some additional details on the contributions in each chapter.

1.2.1 ARUBA

Chapter 2 begins with a theoretical framework for designing and understanding practical meta-learning methods that integrates a mathematical understanding of task similarity with the extensive literature on online convex optimization and sequential prediction algorithms. We call this framework ARUBA, for Average Regret Upper Bound Analysis, and it is based around deriving

nice but *meaningful* bounds on the performance of learning algorithms that can then be optimized via off-the-shelf learning techniques; by analogy to supervised classification, these upper bounds can be viewed as surrogate loss functions for algorithms, in the sense that they are nice functions (e.g. the square or log loss) that we optimize instead of our actual objective (0-1 error).

Our first application of ARUBA is to the study of gradient-based meta-learning (GBML), where we use ARUBA to meta-learn the initialization and other parameters of online convex optimization algorithms such as online gradient descent, which form the basis of many modern deep learning optimizers. We show that modern GBML approaches can be viewed as optimizing a surrogate objective that automatically adapts to a natural notion of task similarity; specifically, we call tasks *similar* if their optimal parameters are close in Euclidean distance. Using ARUBA, we can generalize this simple setup to meta-learning other algorithms in the online mirror descent family—which includes important methods such as exponentiated weights—while adapting to algorithm-dependent notions of task similarity that generalize the Euclidean distance using Bregman divergences. Our approach also enables the task similarity to be learned adaptively, provides sharper transfer risk bounds in the setting of statistical learning-to-learn, and leads to straightforward derivations of average-case regret bounds for efficient algorithms in settings where the task environment changes dynamically or the tasks share a certain geometric structure. We also use ARUBA as a guide for algorithm design, as we demonstrate by modifying several popular meta-learning algorithms and improve their meta-test-time performance on standard problems in few-shot learning and federated learning.

However, the original setup of ARUBA does suggest some limitations, specifically that perhaps the nice but meaningful upper bounds it requires only arise in settings where (a) the loss functions themselves are reasonably nice (e.g. convex and Lipschitz) and (b) where we have full-information access to these losses. In the remainder of Chapter 2 we study two settings that show that our framework can in-principle get around these limitations. Firstly, we study the meta-learning of the initialization and step-size of learning algorithms for *piecewise-Lipschitz* functions, a nonconvex setting with applications to both machine learning and algorithms. Starting from recent regret bounds for the exponential forecaster on losses with dispersed discontinuities, we generalize them to be initialization-dependent and then use this result to propose a practical meta-learning procedure that learns both the initialization and the step-size of the algorithm from multiple online learning tasks. Asymptotically, we guarantee that the average regret across tasks scales with a natural notion of task similarity that measures the amount of overlap between near-optimal regions of different tasks. Our approach relies on a careful analysis of exponentiated weights run on an evolving discretization of the action domain. We also instantiate the method and its guarantee for several problems in multi-task data-driven algorithm design.

Lastly, we study meta-learning of online learning algorithms that use *bandit* feedback, with the goal of improving performance across multiple tasks if they are similar according to some natural similarity measure. As the first to target the adversarial online-within-online partial-information setting, we use ARUBA to design meta-algorithms that combine outer learners to simultaneously tune the initialization and other hyperparameters of an inner learner for two important cases: multi-armed bandits (MAB) and bandit linear optimization (BLO). For MAB, the meta-learners initialize and set hyperparameters of the Tsallis-entropy generalization of Exp3, with the task-averaged regret improving if the entropy of the optima-in-hindsight is small. For BLO, we learn to initialize and tune online mirror descent (OMD) with self-concordant barrier

regularizers, showing that task-averaged regret varies directly with an action space-dependent measure they induce. To apply ARUBA, we show that the regret of OMD in both settings can be bounded by affine functions of non-Lipschitz (and sometimes nonconvex) Bregman divergences, which we then show can be learned via unregularized follow-the-leader combined with two levels of low-dimensional hyperparameter tuning.

1.2.2 FedEx

Meta-learning can be viewed as hyperparameter tuning across multiple tasks, with the initializations, step-sizes, and other settings being the meta-learnable parameters. In Chapter 3 we show how in the setting of federated learning—where we use data from multiple devices to learn an model parameter (that can then be fine-tuned on data from individual clients)—this perspective allows us to design effective algorithms for hyperparameter tuning, which is a crucial but arduous part of any machine learning pipeline. Hyperparameter optimization is even more challenging in federated learning, where models are learned over a distributed network of heterogeneous devices; here, the need to keep data on device and perform local training makes it difficult to efficiently train and evaluate configurations.

In particular, we introduce FedEx, a new method for accelerating federated hyperparameter tuning, specifically in the setting where there are many *on-device* hyperparameters to tune. FedEx can be used with any federated optimization scheme involving a local fine-tuning step followed by server-side aggregation, which describes a very large number of the most popular methods including the most important, FedAvg. Our contribution makes connections to several different parts of the thesis, starting with the ARUBA framework we develop in Chapter 2, which we use to prove that a variant of FedEx correctly tunes the on-device learning rate in the setting of online convex optimization across devices. The FedEx idea of tuning hyperparameters by evaluating different optimizer settings using the same initialization is also closely connected to the technique of *weight-sharing* from neural architecture search (NAS), which we analyze more closely in Chapter 9. Notably, FedEx is the only method we are aware of that uses this technique to tune *non-architectural* hyperparameters, which is usually not possible to do outside of a multi-task setting. Empirically, we show that FedEx can outperform natural baselines for federated hyperparameter tuning by several percentage points on the Shakespeare, FEMNIST, and CIFAR-10 benchmarks—obtaining higher accuracy using the same training budget.

1.2.3 Contributions of independent interest

In addition to new results in the theory of meta-learning and its applications in federated learning, our investigation also make contributions to the field of online learning via the introduction of the **strongly-convex coupling** technique (c.f. Section 2.A.1). This is a proof approach for showing regret guarantees for **Follow-the-Leader** (FTL), a simple online learning algorithm that on each round takes the action that minimizes the sum of the losses seen so far. FTL is too unstable to obtain good (sublinear) regret for general convex losses and is mainly applied when the losses are strongly-convex [Kakade and Shalev-Shwartz, 2008]. However, in Theorem 2.A.1 we extend old stability results for online convex optimization to show that if FTL run on a sequence of loss

functions takes the same actions as FTL run on a different sequence of losses that is strongly-convex, then FTL will have sublinear (indeed, logarithmic) regret on the original sequence.

While it can be challenging to apply, the technique is powerful because it does not even require the original sequence of loss functions to be convex. Indeed, perhaps the most interesting application of strongly-convex coupling is shown in Corollary 2.A.1, where we use it to show that FTL obtains logarithmic regret when run on sequences of Bregman divergences of form $\mathcal{B}(\mathbf{x}_1||\cdot), \ldots, \mathcal{B}(\mathbf{x}_T||\cdot)$. This is notable because Bregman divergences can be nonconvex in their second argument, e.g. when its regularizer is the Tsallis entropy with $\beta < 1$, which arises in our analysis of meta-learning in multi-armed bandits (c.f. Section 2.4.2).

1.3 Discussion

1.3.1 Recent developments

Interest in meta-learning theory has grown concurrently with the work in this thesis as the desire to make use of multiple tasks in different learning settings has expanded. As part of this, our ARUBA framework has been applied many times to derive new meta-learning algorithms with provable guarantees, including for differentially private meta-learning [Li et al., 2020a], distributed multi-agent meta-learning [Lin et al., 2021], constrained multi-task reinforcement learning (RL) [Khattar et al., 2023], and meta-learning in games [Harris et al., 2023]. There has also been significant related work looking at optimization in nonconvex meta-learning [Fallah et al., 2020], learning good representation for few-shot learning [Du et al., 2021b, Tripuraneni et al., 2021], and specific aspects of meta-learning such as the train-validation split in MAML [Saunshi et al., 2021]. The theory presented in this thesis stands out among these results by studying the entire meta-learning pipeline—both optimization and learning—in sufficiently tractable settings. At the same time, recent work has demonstrated that a complete understanding of gradient-based meta-learning is unlikely without nonconvex analysis, i.e. that multi-layer representation learning is necessary to learn initializations that can take advantage of even simple types of task similarity [Saunshi et al., 2020]. Nevertheless, ARUBA has helped develop an understanding of meta-learning that accounts for task similarity, adapts classical learning via its lifting of surrogate loss functions via performance upper bounds, unifies multi-task approaches in disparate subfields, and continues to influence the design of meta-algorithms, especially in learning-theoretic settings beyond standard supervised learning.

In a similar vein, the field of federated hyperparameter tuning has also grown significantly since the release of FedEx, with the development of numerous methods such as FLoRA [Zhou et al., 2023], pFedEx [Wang et al., 2023], HPN [Cheng et al., 2023], FEATHERS [Seng et al., 2023], and FedPop [Chen et al., 2024]; many of these approaches adopt FedEx's use of weight-sharing for tuning non-architectural hyperparameters. Fedex has also been independently evaluated on the recently released FedHPOBench benchmark for federated hyperparameter optimization, where its use was found to improve the performance of standard hyperparameter tuners in eleven of twelve cases [Wang et al., 2023, Table 2]. It was also the subject of a study by Nakka et al. [2024], who observe that FedEx can sometimes perform insufficient exploration and note that the configurations it determines to be good are often suboptimal, a concern similar to obser-

vations of rank disorder in neural architecture search with weight-sharing [Yu et al., 2020a]; as we argue in Chapter 9, methods can still perform well even in the face of rank disorder. Beyond benchmarking and methods, recent work has also significantly expanded our understanding of the evaluation issues that we identify in Chapter 3, showing the surprising result that simple random search can outperform more sophisticated tuning schemes (e.g. successive halving) in the presence of the types of noise (e.g. due to differential privacy, client sampling, etc.) present in federated optimization [Kuo et al., 2023].

1.3.2 Looking forward

Approaches such as fine-tuning large-scale pretrained models [Devlin et al., 2019] and using them to in-context learn [Brown et al., 2020] have become extremely popular approaches for few-shot learning, the original motivation for GBML [Finn et al., 2017]. It is tempting to frame these approaches as themselves variants of meta-learning: after all, language large-scale models are often trained on vast, *heterogeneous* corpora to learn an initialization for gradient descent, and individual collections can be viewed as their own task. However, meta-learning tools have found more use in settings (e.g. federated learning) that are more similar to those analyzed in this thesis, where we learn across numerous small tasks (e.g. clients) that are themselves similar to those that on which we eventually fine-tune. Bridging this gap between large-scale pretraining and what we classically understand as meta-learning is an interesting challenge for both theory and practice. In particular, there is significant future work in understanding the properties (e.g. task size and data heterogeneity) of pretraining corpora that delineate when if ever we can improve our initializations using knowledge that they will be fine-tuned using SGD.

Beyond improved pretraining, the rise of large-scale pretraining and large language models (LLMs) yield many other interesting opportunities for multi-task techniques. For example, inference costs with such models is incredibly high but done repeatedly: can meta-learning across inferences be used to reduce such costs or to reduce the number of queries needed? In the other direction, can the representation power of LLMs be exploited to induce useful embeddings of tasks from their natural language descriptions? Such representations have been found to be practically useful ways of accessing task relatedness information [Achille et al., 2019], but the fact that language models can now meaningfully encode natural language information raises exciting new possibilities for generating them. Lastly, in the direction of theory and basic science, understanding the interplay between the heterogeneity in pretraining corpora and the "skills" learned by the resulting models [Arora and Goyal, 2023, Chen et al., 2023] is another valuable direction for future research.

1.A Background

1.A.1 Online learning

Aspects of online learning appear throughout this thesis so we give a quick overview in this section. There exist several excellent resources on this topic that we will draw from and that the reader may find helpful [Shalev-Shwartz, 2011, Hazan, 2015, Orabona, 2022].

In the basic setup of online learning we are faced with an adversary over T rounds $t = 1, \dots, T$, on each of which we first take an action $\mathbf{x}_t \in \mathcal{X}$ in some domain $\mathcal{X}$ and then suffer loss $\ell_t(\mathbf{x}_t)$ according to some loss function $\ell_t : \mathcal{X} \mapsto \mathbb{R}$ chosen by the adversary. The usual goal of online learning is to minimize **regret**, defined as the difference between the loss we incur over T rounds and the loss incurred by the best fixed action in hindsight:

$$\text{Regret} = \sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \min_{\mathbf{x} \in \mathcal{X}} \sum_{t=1}^{T} \ell_t(\mathbf{x}) \tag{1.1}$$

Specifically, the minimal goal of online learning is to obtain *sublinear* regret, i.e. $\text{Regret} = o(T)$, in which case the average loss (relative to the optimum) Regret/T approaches zero as $T \to \infty$. Starting from this basic setup, the field of online learning explores what happens under different conditions of interest, such as via restrictions on the losses (e.g. convex or Lipschitz), restrictions on the adversary (e.g. oblivious or stochastic), or different notions of regret (e.g. using dynamic comparators).

To start off we will consider the well-studied setting where the domain $\mathcal{X} \subset \mathbb{R}^d$ is convex and the losses are convex and Lipschitz, a field commonly known as **online convex optimization** (OCO). In this setting, sublinear regret is obtained via the (projected) **online gradient descent** (OGD) algorithm [Zinkevich, 2003], which given a step-size $\eta > 0$ plays

$$\mathbf{x}_{t+1} = \text{Proj}_{\mathcal{X}}(\mathbf{x}_t - \eta \nabla \ell_t(\mathbf{x}_t)) \tag{1.2}$$

Here the first point $\mathbf{x}_1$ can be an arbitrary point in $\mathcal{X}$ and the projection is in terms of the Euclidean norm. This algorithm has the following regret guarantee [Shalev-Shwartz, 2011, Theorem 2.11]:

Theorem 1.A.1. If $\mathcal{X} \subset \mathbb{R}^d$ is convex and the losses $\ell_t : \mathcal{X} \mapsto \mathbb{R}$ are convex and G_t-Lipschitz then OGD with step-size $\eta > 0$ and initialization $\mathbf{x}_1 \in \mathcal{X}$ has regret bounded as

$$\text{Regret} \leqslant \frac{\|\mathbf{x}^* - \mathbf{x}_1\|_2^2}{2\eta} + \eta G^2 T \tag{1.3}$$

for $G^2 = \frac{1}{T}\sum_{t=1}^{T} G_t^2$ and $\mathbf{x}^* \in \arg\min_{\mathbf{x} \in \mathcal{X}} \sum_{t=1}^{T} \ell_t(\mathbf{x})$. Assuming $\mathcal{X}$ has Euclidean radius at most D and setting $\eta = \frac{D}{G\sqrt{T}}$ yields regret $\mathcal{O}(GD\sqrt{T})$.

Online gradient descent is just one of many algorithms for online learning and online convex optimization; indeed the field has developed a deep understanding of the connections between various algorithms culminating in the **online mirror descent** (OMD) meta-algorithm, which is in some sense universal [Srebro et al., 2011]. OMD guarantees take a from similar to given in

Theorem 1.A.1 for OGD, except the squared Euclidean distance is replaced by a generalized measure called a **Bregman divergence**. In Chapter 2 will take advantage of this to induce many interest notions of task similarity that meta-learners can adapt to.

Lastly, a crucial aspect of online learning is that regret guarantees proved for sequences of adversarial losses can be converted into sample complexity or statistical risk bounds via results known as *online-to-batch* conversions [Cesa-Bianchi et al., 2004]. These conversions run the online algorithm on losses that are actually i.i.d., aggregate the resulting iterates (e.g. by averaging), and then bound the excess risk by a quantity scaling in $U(T)/T$, where $U(T) \geqslant \mathrm{Regret}$ upper-bounds the regret. Since a typical rate for $U(T)$ is $\mathcal{O}(\sqrt{T})$, it is easy to see that these conversions can attain excess risk rates that are competitive with standard uniform convergence guarantees; indeed it can require significant effort to show a learning-theoretic separation between adversarial and online analysis [Hazan and Kale, 2014]. As a result, we will be able to use online-to-batch conversion to obtain compelling statistical guarantees, both for meta-learning and later for algorithms with predictions.

1.A.2 Multi-task learning

The term *meta-learning* has been used to describe a variety of settings, including others addressed in this thesis such as hyperparameter tuning. We will mainly use it to refer to a kind of multi-task learning where the goal is to learn to parameterize a learning algorithm; this distinguishes it from hyperparameter tuning, where usually the objective is to parameterize an algorithm on a single task, although as shown in Chapter 3 even this distinction can be blurred. It also distinguishes it from what we usually think of as *multi-task learning* [Caruana, 1997], in which a shared model or representation is trained to solve multiple tasks [Evgeniou and Pontil, 2004], e.g. via different output heads of a neural network [Caruana et al., 1995, Weinberger et al., 2009]. Note that we will often consider an *online* setting of meta-learning, where the tasks are to be solved sequentially; this and similar setups are sometimes referred to as *lifelong learning* and *continual learning*, but here again terms can be defined in different ways by different authors [Alquier et al., 2017, Jerfel et al., 2019]. Our online setting for meta-learning is distinct, however, from *online multi-task learning* [Cavallanti et al., 2010], in which there is a fixed number of tasks and the examples arrive sequentially (and with an associated task index).

A core empirical motivation of this thesis is *gradient-based* meta-learning, in which the goal is to learn an initialization for methods in the gradient descent family. Prominent methods that we will refer to here are Model-Agnostic Meta-Learning (MAML) [Finn et al., 2017], in which gradient descent is used to optimize an objective averaging the loss one gradient step away from the learned initialization, and a first-order variant called Reptile [Nichol et al., 2018] that simply minimizes the average distance between the initialization and the last iterate. As we make extensive use of in both chapters, the update rule of Reptile is a generalization of the popular federated learning algorithm FedAvg [McMahan et al., 2017]: whereas the FedAvg update moves all the way to the average of last iterates of local SGD applied to a batch of tasks (client devices), Reptile moves to a convex combination of that average and the initialization used for local SGD.

Chapter 2

ARUBA: Provable guarantees for meta-learning

The first two parts of this thesis are concerned with the learning of algorithms, specifically with optimizing their parameters over a sequence or distribution of tasks in order to improve some relevant notion of cost. Target methods whose parameters we will consider learning include regular algorithms such as the linear system solvers and learning algorithms such as online gradient descent. In the former case the tasks will be individual computational instances and the cost measure will often be runtime, while in the latter case we will meta-learn using learning tasks to minimize quantities such as regret or risk.

In this chapter we introduce the main theoretical tool we develop for this purpose—Average Regret Upper Bound Analysis (ARUBA)—and discuss its advantages and disadvantages as a tool for developing new methods for meta-learning, understanding existing ones, and learning predictions. We will then demonstrate its application in a variety of learning-theoretic settings, while in the next chapter we highlight an empirical application to federated hyperparameter tuning.

2.1 Framework

We introduce our main theoretical tool in the context in which it was originally developed: meta-learning to initialize gradient descent. The empirical motivation to study this problem comes from the empirical literature on meta-learning, also known as *learning-to-learn*, in which the learner is in an environment with numerous learning tasks each having little data; the goal is to use data from previously seen *meta-training* tasks in order to learn how to do well when faced with a new *meta-test* task and given only a few examples from it. While this type of multi-task learning has been studied both empirically and theoretically for many years [Thrun and Pratt, 1998, Baxter, 2000, Maurer, 2005], its widespread integration with modern applications (e.g. both few-shot supervised learning and reinforcement learning) and modern models (deep neural networks) came through the gradient-based meta-learning (GBML) approach [Finn et al., 2017].

[0]ARUBA was first introduced in Khodak et al. [2019a], building upon ideas in an earlier work [Khodak et al., 2019b]; the original focus was to show guarantees for gradient-based meta-learning. Its subsequent applications to nonconvex and bandit meta-learning first appeared in Balcan et al. [2021b] and Khodak et al. [2023b], respectively.

There the authors introduced **Model-Agnostic Meta-Learning** (MAML), an approach in which the meta-training tasks are used to meta-learn an initialization for a deep network; this initialization is then used to initialize gradient descent on samples from a meta-test task.

MAML and its variants became popular because they can be used with any system relying on gradient-based methods for learning, covering much of modern deep learning. A particularly simple and illustrative variant is Reptile [Nichol et al., 2018], which meta-learns an initialization $\hat{\boldsymbol{\phi}}$ by running stochastic gradient descent (SGD) starting from some initialization $\boldsymbol{\phi}_t$ on each of a sequence of tasks $t = 1, \dots, T$ and setting the next initialization to

$$\boldsymbol{\phi}_{t+1} = (1-\alpha)\boldsymbol{\phi}_t + \alpha\hat{\boldsymbol{\theta}}_t \tag{2.1}$$

where $\hat{\boldsymbol{\theta}}_t$ is the last iterate of SGD on task t and $\alpha > 0$ is a meta-step-size. At meta-test time Reptile runs SGD from the meta-learned initialization $\hat{\boldsymbol{\phi}} = \boldsymbol{\phi}_{T+1}$ on samples from unseen tasks.

We cover subsequent methods, other work on theoretical guarantees for GBML, and the motivation for the online setting in subsequent chapters. For now, to introduce ARUBA, we consider a multi-task extension of the online learning framework introduced in Section 1.A.1. We consider a *meta*-learner faced with mT losses $\ell_{t,i}$ for $t = 1, \dots, T$ and $i = 1, \dots, m$; thus each t corresponds to a task with m rounds each. The learner's goal is to play actions $\mathbf{x}_{t,i} \in \mathcal{X}$ that minimize their **task-averaged regret**:

$$\overline{\mathrm{Regret}} = \frac{1}{T}\sum_{t=1}^{T}\mathrm{Regret}_t = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m}\ell_{t,i}(\mathbf{x}_{t,i}) - \min_{\mathbf{x}\in\mathcal{X}}\sum_{i=1}^{m}\ell_{t,i}(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m}\ell_{t,i}(\mathbf{x}_{t,i}) - \ell_{t,i}(\mathbf{x}_t^*) \tag{2.2}$$

Here we define Regret_t to be the realized regret on task t and $\mathbf{x}_t^*$ to be the its minimum-Euclidean-norm optimum-in-hindsight.

To minimize task-averaged regret we take the GBML approach of using the same algorithm, e.g. OGD, on each task t in the sequence. This reduces the question to setting the parameters of this algorithm—e.g. the initialization $\mathbf{x}$ and step-size η in the case of OGD—for each task. Focusing mainly on the initialization, we now introduce our key technique: analyzing the meta-learner's performance by studying the online learning of a sequence of regret-upper-bounds $U_t(\mathbf{x}_{t,1}) \geqslant \mathrm{Regret}_t$, specifically by bounding the average regret-upper-bound $\overline{U} = \frac{1}{T}\sum_{t=1}^{T}U_t(\mathbf{x}_{t,1})$. The following two observations highlight why we care about this quantity:

1. **Generality:** Many learning algorithms of interest have regret-upper-bounds $U_t(\mathbf{x})$ with nice, e.g. convex, functional forms that depend strongly on both their parameterizations $\mathbf{x} \in \mathcal{X}$ and the task data. This data-dependence lets us adaptively set $\mathbf{x}_{t,1} \in \mathcal{X}$.
2. **Consequences:** By definition of U_t we have that $\overline{U}$ bounds the task-averaged regret $\overline{\mathrm{Regret}}$. Thus if the average regret-upper-bound is small then the meta-learner will perform well on-average across tasks.

ARUBA's applicability depends only on finding a low-regret algorithm for the functions U_t; then by observation 2 we get a task-averaged regret bound where the first term vanishes as $T \to \infty$ while by observation 1 the second term is small due to the data-dependent task similarity:

$$\overline{\mathrm{Regret}} \leqslant \overline{U} \leqslant o_T(1) + \min_{\mathbf{x}\in\mathcal{X}}\frac{1}{T}\sum_{t=1}^{T}U_t(\mathbf{x}) \tag{2.3}$$

As the first simple instantiation of ARUBA, suppose the meta-learner is indeed using OGD as the within-task algorithm and the losses on all tasks are convex and G-Lipschitz. Then by Theorem 1.A.1 we have an upper bound of $U_t(\mathbf{x}) = \frac{1}{2\eta}\|\mathbf{x}_t^* - \mathbf{x}\|_2^2 + \eta G^2 m$ on the regret of OGD with initialization $\mathbf{x} \in \mathcal{X}$ and step-size $\eta > 0$. Note that U_t is convex in the initialization $\mathbf{x}$ and depends strongly on the data via the optimal action $\mathbf{x}_t^*$ in hindsight. In-particular, this means that OGD is also a low-regret algorithm on the sequence $U_1, \ldots, U_T$,[1] and thus using it to set the initializations $\mathbf{x}_{t,1}$ will result in a task-averaged regret bounded as

$$\overline{\text{Regret}} \leqslant \overline{U} \leqslant o_T(1) + \min_{\mathbf{x}\in\mathcal{X}} \frac{1}{T}\sum_{t=1}^{T} \frac{\|\mathbf{x}_t^* - \mathbf{x}\|_2^2}{2\eta} + \eta G^2 m = o_T(1) + \mathcal{O}(GV\sqrt{m}) \tag{2.4}$$

Here we have introduced a *task similarity* notion $V^2 = \min_{\mathbf{x}\in\mathcal{X}} \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{x}_t^* - \mathbf{x}\|_2^2$ measuring the empirical variance between the optima in hindsight and then set $\eta = \mathcal{O}\left(\frac{V}{G\sqrt{m}}\right)$.[2] The key result here is that in the case where tasks are similar according to this notion—i.e. when $V \ll D$ for D the Euclidean radius of $\mathcal{X}$—then as $T \to \infty$ meta-learning improves dramatically upon the $\mathcal{O}(GD\sqrt{m})$ regret obtained by simply running OGD independently on each task.

As a final observation, note that if the meta-OGD procedure has step-size $\alpha\eta > 0$ then the meta-update to set $\mathbf{x}_{t,1}$ has the form

$$\mathbf{x}_{t+1,1} = \text{Proj}_{\mathcal{X}}(\mathbf{x}_{t,1} - \alpha\eta\nabla U_t(\mathbf{x}_{t,1})) = \text{Proj}_{\mathcal{X}}(\mathbf{x}_{t,1} - \alpha(\mathbf{x}_{t,1} - \mathbf{x}_t^*)) = (1-\alpha)\mathbf{x}_{t,1} + \alpha\mathbf{x}_t^* \tag{2.5}$$

which, apart from the use of the optimum-in-hindsight $\mathbf{x}_t^*$ rather than the last-iterate is identical to the Reptile update (2.1). Since Reptile uses a very similar method to OGD (SGD) on each task, our result shows that it can be interpreted as meta-learning an initialization that performs well on subsequent tasks so long as the optimal parameters of most tasks are close to each other. This interpretation is the first formal justification for Reptile, which was originally introduced without any formal optimization objective.

2.1.1 Advantages of learning algorithmic upper bounds

Having introduced ARUBA in the context of minimizing average regret across a a sequence of online learning tasks, we now state it more generally and discuss several advantages of this approach. Broadly speaking, the idea of ARUBA is summarized in the following two-step process:

1. For a given algorithm, find or derive a convenient-to-optimize upper bound $U_t(\mathbf{x})$ on the cost of running it on task or instance t using parameterization $\mathbf{x}$.
2. Apply online learning tools to obtain both regret guarantees against adversarial sequences and sample complexity bounds for instances drawn from a distribution.

We now discuss some factors behind the success of this approach.

Existence of meaningful upper bounds. To apply ARUBA we require algorithms to have upper bounds that (a) can be optimized and (b) provide some meaningful bound on the performance.

[1] In subsequent sections we use a better algorithm that uses the strong-convexity of U_t.

[2] In subsequent sections we also learn η rather than assume knowledge of V.

For example, in the OGD example the quadratic upper bound was optimizable due to its (strong) convexity and provided a meaningful performance bound via the task similarity notion of average squared distance from the best initialization. In fact guarantees that satisfy such notions appear frequently throughout machine learning, especially online learning, in which the mirror descent family of algorithms all have regret guarantees that depend on a Bregman divergence between the optimal in hindsight and the initialization [Shalev-Shwartz, 2011]. This family includes well-known examples such as OGD and exponentiated gradient (EG). Beyond learning algorithms, computational methods also exhibit such bounds; as a basic example, the error of batch gradient descent can also be shown to depend on the initialization and step-size [Karimi et al., 2016]. Moreover, the field of algorithms with predictions [Mitzenmacher and Vassilvitskii, 2021] is effectively dedicated devising algorithms whose runtime or other cost measure depends directly on the error of a predictor or other measure of suboptimality of some tuneable parameter.

Depth of results and applications of online learning. The choice of online learning as a source of both within-task and meta-learning algorithms is important to ARUBA, as it allows us to obtain guarantees under general assumptions and draw upon a very large literature in sequential prediction. This is especially useful for showing learnability of the wide variety of upper bounds we encounter and for converting the adversarial results to statistical learning guarantees. Online learning is also notable as a major source of learning algorithms for modern neural network optimization; in-particular, the most popular adaptive methods such as Adam [Kingma and Ba, 2015] and AdaGrad [Duchi et al., 2011] were derived using online convex optimization. Thus by using online algorithms we obtain methods that can often be applied with minimal changes to deep neural networks, even if their guarantees are only for convex settings.

Ease of learning multiple parameters simultaneously. In both meta-learning and algorithms with predictions we are often concerned with learning multiple types of parameters simultaneously; for example, in the former we are often interested in both the initialization and step-size while in the latter we want to both find a good predictor and fix a good parameter for the robustness-consistency tradeoff. By optimizing regret-upper-bounds using online learning we are frequently able to obtain algorithms that can learn multiple parameters simultaneously so long as we have separate no-regret algorithms for each. In the case of the OGD step-size, if we have an algorithm that sets η_t that obtains sublinear regret on the sequence $\frac{1}{2\eta}\|\mathbf{x}_t - \mathbf{x}_{t,1}\|_2^2 + \eta G^2 m$ for arbitrary $\mathbf{x}_{t,1}$ then using it in-combination with OGD to set $\mathbf{x}_{t,1}$ as described above yields task-averaged regret

$$\begin{aligned}\overline{\text{Regret}} \leqslant \overline{U} = \frac{1}{T}\sum_{t=1}^{T}\frac{\|\mathbf{x}_t^* - \mathbf{x}_{t,1}\|_2^2}{\eta_t} + \eta_t G^2 m &= o_T(1) + \min_{\eta>0}\sum_{t=1}^{T}\frac{\|\mathbf{x}_t^* - \mathbf{x}_{t,1}\|_2^2}{\eta} + \eta G^2 m \\ &= o_T(1) + \min_{\eta>0,\mathbf{x}\in\mathcal{X}}\sum_{t=1}^{T}\frac{\|\mathbf{x}_t^* - \mathbf{x}\|_2^2}{\eta} + \eta G^2 m\end{aligned} \tag{2.6}$$

which tends to $\mathcal{O}(GV\sqrt{m})$ without needing to know η in advance. We discuss this in detail in subsequent chapters.

2.1.2 Challenges of applying ARUBA

Applying ARUBA on specific does lead to some challenges that we need to resolve on a case-by-case basis.

Enforcing meaningful upper bounds. It can be sometimes difficult to obtain useful upper bounds that meaningfully characterize the performance of a method. In-particular, we can of course always construct trivially learnable upper bounds just from worst-case guarantees, or even due to weakness in a data or hyperparameter-dependent upper bound. Arguing that a specific bound is useful requires knowledge of the application domain and understanding useful notions of task similarity.

Importance of within-algorithm dynamics. While upper bounds can be a useful starting guide, for some applications meta-learning can only be understood fully by studying the per-iteration behavior of the base learner. For example, showing that a linear model must be over-parameterized as a two-layer linear network in order to meta-learn an initialization of SGD when the optimal models of all tasks lie in a one-dimensional subspace required analyzing both within-task and meta-training dynamics [Saunshi et al., 2020]. Upper bounds are not fully be able to characterize the effect of the initialization on the trajectory of iterative algorithms.

Obtaining statistical results. In areas like statistical and bandit learning we generally have access only to empirical measures of the relevant cost function (risk) and so cannot optimize it directly. Thus, while in areas such as algorithms with predictions it is often possible to compare directly with past statistical guarantees, in meta-learning this is sometimes not the case and the online-to-batch conversion may be viewed as lossy.

2.2 Gradient-based meta-learning

In this section we discuss our application of ARUBA to minimizing task-averaged regret when meta-learning to initialize and set the step-size of gradient-based methods across a sequence of online learning tasks. We consider a meta-learner facing a sequence of online learning tasks $t = 1, \dots, T$, each with m_t loss functions $\ell_{t,i} : \Theta \mapsto \mathbb{R}$ over action-space $\Theta \subset \mathbb{R}^d$. The learner has access to a set of learning algorithms parameterized by $\mathbf{x} \in \mathcal{X}$ that can be used to determine the action $\boldsymbol{\theta}_{t,i} \in \Theta$ on each round $i \in [m_t]$ of task t. Thus on each task t the meta-learner chooses $\mathbf{x}_t \in \mathcal{X}$, runs the corresponding algorithm, and suffers regret $\mathrm{Regret}_t = \sum_{i=1}^{m_t} \ell_{t,i}(\boldsymbol{\theta}_{t,i}) - \min_{\boldsymbol{\theta}} \sum_{i=1}^{m_t} \ell_{t,i}(\boldsymbol{\theta})$. For example, we often use online gradient descent as the within-task learning algorithm, as is done by Reptile [Nichol et al., 2018]. OGD can be parameterized by an initialization $\boldsymbol{\phi} \in \Theta$ and a learning rate $\eta > 0$, so that $\mathcal{X} = \{(\boldsymbol{\phi}, \eta) : \boldsymbol{\phi} \in \Theta, \eta > 0\}$. Using the notation $\mathbf{v}_{a:b} = \sum_{i=a}^{b} \mathbf{v}_i$ and $\nabla_{t,j} = \nabla \ell_{t,j}(\boldsymbol{\theta}_{t,j})$, at each round i of task t OGD plays $\boldsymbol{\theta}_{t,i} = \arg\min_{\boldsymbol{\theta} \in \Theta} \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\phi}\|_2^2 + \eta \langle \nabla_{t,1:i-1}, \boldsymbol{\theta} \rangle$.

Our first result is a multi-task extension of Abernethy et al. [2008a, Theorem 4.2] that gives a lower-bound on the task-averaged regret:

Algorithm 1: Generic online algorithm for gradient-based parameter-transfer meta-learning. To run OGD within-task set $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$. To run FTRL within-task substitute $\ell_{t,j}(\boldsymbol{\theta})$ for $\langle \nabla_{t,j}, \boldsymbol{\theta} \rangle$.

Set meta-initialization $\boldsymbol{\phi}_1 \in \Theta$ and learning rate $\eta_1 > 0$.
for task $t \in [T]$ **do**
  `// run` m `steps of online mirror descent (OMD)`
  for round $i \in [m_t]$ **do**
    $\boldsymbol{\theta}_{t,i} \leftarrow \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{B}_R(\boldsymbol{\theta} || \boldsymbol{\phi}_t) + \eta_t \langle \nabla_{t,1:i-1}, \boldsymbol{\theta} \rangle$ Suffer loss $\ell_{t,i}(\boldsymbol{\theta}_{t,i})$
  `// meta-update OMD initialization and learning rate`
  Update $\boldsymbol{\phi}_{t+1}, \eta_{t+1}$

Corollary 2.2.1. Assume $d \geqslant 3$ and that for each $t \in [T]$ an adversary must play a sequence of m convex G-Lipschitz functions $\ell_{t,i} : \Theta \mapsto \mathbb{R}$ whose optimal actions in hindsight $\arg\min_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^m \ell_{t,i}(\theta)$ are contained in some fixed ℓ_2-ball $\Theta^* \subset \Theta$ with center $\boldsymbol{\phi}^*$ and diameter D^*. Then the adversary can force the agent to have task-averaged regret at least $\frac{GD^*}{4}\sqrt{m}$.

Since by definition $D^* \geqslant V$ for V as defined in Section 2.1, this result shows that the example guarantee for OGD discussed before—a simple case of the guarantees shown later in this section—is asymptotically (as $T \to \infty$) optimal up to a constant multiplicative factor.

2.2.1 Adapting to similar tasks and dynamic environments

We now demonstrate the effectiveness of ARUBA for analyzing GBML by using it to prove a general bound for a class of algorithms that can adapt to both *task similarity*, i.e. when the optimal actions $\boldsymbol{\theta}_t^*$ for each task are close to some good initialization, and to *changing environments*, i.e. when this initialization changes over time. The task similarity will be measured using the **Bregman divergence** $\mathcal{B}_R(\boldsymbol{\theta} || \boldsymbol{\phi}) = R(\boldsymbol{\theta}) - R(\phi) - \langle \nabla R(\boldsymbol{\phi}), \boldsymbol{\theta} - \boldsymbol{\phi} \rangle$ of a 1-strongly-convex function $R : \Theta \mapsto \mathbb{R}$ [Bregman, 1967], a generalized notion of distance.[3] Note that for $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ we have $\mathcal{B}_R(\boldsymbol{\theta} || \boldsymbol{\phi}) = \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\phi}\|_2^2$. A changing environment will be studied by analyzing **dynamic regret**, which for a sequence of actions $\{\boldsymbol{\phi}_t\}_t \subset \Theta$ taken by some online algorithm over a sequence of loss functions $\{f_t : \Theta \mapsto \mathbb{R}\}_t$ is defined w.r.t. a reference sequence $\Psi = \{\boldsymbol{\psi}_t\}_t \subset \Theta$ as $\text{Regret}_\Psi = \sum_{t=1}^T f_t(\boldsymbol{\phi}_t) - f_t(\boldsymbol{\psi}_t)$. Dynamic regret measures the performance of an online algorithm taking actions $\boldsymbol{\phi}_t$ relative to a potentially time-varying comparator taking actions ψ_t. Note that when we fix $\boldsymbol{\psi}_t = \boldsymbol{\psi}^* \in \arg\min_{\boldsymbol{\psi} \in \Theta} \sum_{t=1}^T f_t(\boldsymbol{\psi})$ we recover the standard **static regret**, in which the comparator always uses the same action.

Putting these together, we seek to define variants of Algorithm 1 for which as $T \to \infty$ the average regret scales with V_Ψ, where $V_\Psi^2 = \frac{1}{T}\sum_{t=1}^T \mathcal{B}_R(\boldsymbol{\theta}_t^* || \boldsymbol{\psi}_t)$, without knowing this quantity in advance. Note for fixed $\boldsymbol{\psi}_t = \bar{\boldsymbol{\theta}}^* = \frac{1}{T}\boldsymbol{\theta}_{1:T}^*$ this measures the empirical standard deviation of the optimal task actions $\boldsymbol{\theta}_t^*$. Thus achieving our goal implies that average performance improves with task similarity.

[3] See Appendix B.2 for more properties of Bregman divergences.

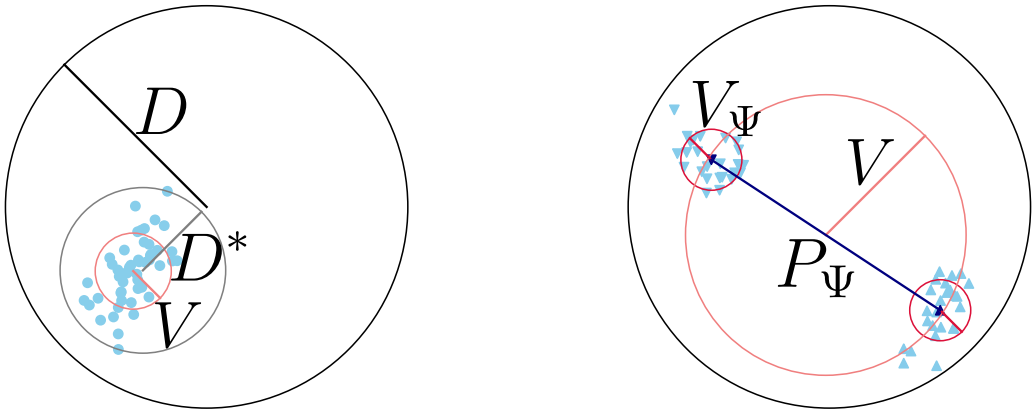

Figure 2.1: Illustrations comparing different notions of task similarity. The left plot depicts notions in the static setting, including the average deviation V on which Theorem 2.2.2 depends, the maximal deviation D^* from the meta-learning lower bound in Corollary 2.2.1, and the radius D of the entire action space on which worst-case bounds depend. The right plot shows a setting where Theorem 2.2.3 yields a strong task similarity-based guarantee via a dynamic comparator Ψ, despite the average deviation V being large due to tasks being in far-away clusters.

On each task t Algorithm 1 runs online mirror descent with regularizer R, initialization $\boldsymbol{\phi}_t \in \Theta$, and learning rate $\eta_t > 0$. It is well-known that OMD and the related Follow-the-Regularized-Leader (FTRL), for which our results also hold, generalize many important online methods, e.g. OGD and multiplicative weights [Hazan, 2015]. For m_t convex losses with mean squared Lipschitz constant G_t^2 they also share a convenient, data-dependent regret-upper-bound for any $\boldsymbol{\theta}_t^* \in \Theta$ [Shalev-Shwartz, 2011, Theorem 2.15]:

$$\text{Regret}_t \leqslant U_t(\boldsymbol{\phi}_t, \eta_t) = \frac{1}{\eta_t}\mathcal{B}_R(\theta_t^* || \boldsymbol{\phi}_t) + \eta_t G_t^2 m_t \tag{2.7}$$

All that remains is to come up with update rules for the meta-initialization $\boldsymbol{\phi}_t \in \Theta$ and the learning rate $\boldsymbol{\eta}_t > 0$ in Algorithm 1 so that the average over T of these upper-bounds $U_t(\boldsymbol{\phi}_t, \eta_t)$ is small. While this can be viewed as a single online learning problem to determine actions $x_t = (\boldsymbol{\phi}_t, \eta_t) \in \Theta \times (0, \infty)$, it is easier to decouple $\boldsymbol{\phi}$ and η by first defining two function sequences $\{f_t^{\text{init}}\}_t$ and $\{f_t^{\text{sim}}\}_t$:

$$f_t^{\text{init}}(\boldsymbol{\phi}) = \mathcal{B}_R(\boldsymbol{\theta}_t^* || \boldsymbol{\phi}) G_t \sqrt{m_t} \qquad f_t^{\text{sim}}(v) = \left(\frac{\mathcal{B}_R(\boldsymbol{\theta}_t^* || \boldsymbol{\phi}_t)}{v} + v\right) G_t \sqrt{m_t} \tag{2.8}$$

We show in Theorem 2.2.1 that to get an adaptive algorithm it suffices to specify two algorithms, `INIT` and `SIM`, such that the actions $\boldsymbol{\phi}_t = \texttt{INIT}(t)$ achieve low (dynamic) regret over f_t^{init} and the actions $v_t = \texttt{SIM}(t)$ achieve low (static) regret over f_t^{sim}; these actions then determine the update rules of $\boldsymbol{\phi}_t$ and $\eta_t = v_t/(G_t\sqrt{m_t})$. We will specialize Theorem 2.2.1 to derive algorithms that adapt to task similarity (Theorem 2.2.2) and to dynamic environments (Theorem 2.2.3).

To understand the formulation of f_t^{init} and f_t^{sim}, first note that $f_t^{\text{sim}}(v) = U_t(\boldsymbol{\phi}_t, v/(G_t\sqrt{m_t}))$, so the online algorithm `SIM` over f_t^{sim} corresponds to an online algorithm over the regret-upper-bounds U_t when the sequence of initializations $\boldsymbol{\phi}_t$ is chosen adversarially. Once we have shown

that SIM is low-regret we can compare its losses $f_t^{\text{sim}}(v_t)$ to those of an arbitrary fixed $v > 0$; this is the first line in the proof of Theorem 2.2.1 (below). For fixed v, each $f_t^{\text{init}}(\boldsymbol{\phi}_t)$ is an affine transformation of $f_t^{\text{sim}}(v)$, so the algorithm INIT with low dynamic regret over f_t^{init} corresponds to an algorithm with low dynamic regret over the regret-upper-bounds U_t when $\eta_t = v/(G_t\sqrt{m_t})\ \forall\ t$. Thus once we have shown a dynamic regret guarantee for INIT we can compare its losses $f_t^{\text{init}}(\boldsymbol{\phi}_t)$ to those of an arbitrary comparator sequence $\{\boldsymbol{\psi}_t\}_t \subset \Theta$; this is the second line in the proof of Theorem 2.2.1.

Theorem 2.2.1. Assume $\Theta \subset \mathbb{R}^d$ is convex, each task $t \in [T]$ is a sequence of m_t convex losses $\ell_{t,i} : \Theta \mapsto \mathbb{R}$ with mean squared Lipschitz constant G_t^2, and $R : \Theta \mapsto \mathbb{R}$ is 1-strongly-convex.

- Let INIT be an algorithm whose dynamic regret over functions $\{f_t^{\text{init}}\}_t$ w.r.t. any reference sequence $\Psi = \{\boldsymbol{\psi}_t\}_{t=1}^T \subset \Theta$ is upper-bounded by $U_T^{\text{init}}(\Psi)$.
- Let SIM be an algorithm whose static regret over functions $\{f_t^{\text{sim}}\}_t$ w.r.t. any $v > 0$ is upper-bounded by a non-increasing function $U_T^{\text{sim}}(v)$ of v.

If Algorithm 1 sets $\boldsymbol{\phi}_t = \texttt{INIT}(t)$ and $\eta_t = \frac{\texttt{SIM}(t)}{G_t\sqrt{m_t}}$ then for $V_\Psi^2 = \frac{\sum_{t=1}^T \mathcal{B}_R(\boldsymbol{\theta}_t^* || \boldsymbol{\psi}_t) G_t\sqrt{m_t}}{\sum_{t=1}^T G_t\sqrt{m_t}}$ it achieves

$$\overline{\text{Regret}} \leqslant \overline{U} \leqslant \frac{U_T^{\text{sim}}(V_\Psi)}{T} + \frac{1}{T}\min\left\{\frac{U_T^{\text{init}}(\Psi)}{V_\Psi}, 2\sqrt{U_T^{\text{init}}(\Psi)\sum_{t=1}^T G_t\sqrt{m_t}}\right\} + \frac{2V_\Psi}{T}\sum_{t=1}^T G_t\sqrt{m_t} \tag{2.9}$$

Proof. For $\sigma_t = G_t\sqrt{m_t}$ we have by the regret bound on OMD/FTRL (1.1) that

$$\begin{aligned}
\overline{U}T = \sum_{t=1}^T \left(\frac{\mathcal{B}_R(\boldsymbol{\theta}_t^* || \boldsymbol{\phi}_t)}{v_t} + v_t\right)\sigma_t &\leqslant \min_{v>0} U_T^{\text{sim}}(v) + \sum_{t=1}^T \left(\frac{\mathcal{B}_R(\theta_t^* || \boldsymbol{\phi}_t)}{v} + v\right)\sigma_t \\
&\leqslant \min_{v>0} U_T^{\text{sim}}(v) + \frac{U_T^{\text{init}}(\Psi)}{v} + \sum_{t=1}^T \left(\frac{\mathcal{B}_R(\boldsymbol{\theta}_t^* || \boldsymbol{\psi}_t)}{v} + v\right)\sigma_t \\
&\leqslant U_T^{\text{sim}}(V_\Psi) + \min\left\{\frac{U_T^{\text{init}}(\Psi)}{V_\Psi}, 2\sqrt{U_T^{\text{init}}(\Psi)\sigma_{1:T}}\right\} + 2V_\Psi\sigma_{1:T}
\end{aligned} \tag{2.10}$$

where the last line follows by substituting $v = \max\left\{V_\Psi, \sqrt{U_T^{\text{init}}(\Psi)/\sigma_{1:T}}\right\}$. □

Similar tasks in static environments

By Theorem 2.2.1, if we can specify algorithms INIT and SIM with sublinear regret over f_t^{init} and f_t^{sim} (2.8), respectively, then the average regret will converge to $\mathcal{O}(V_\Psi\sqrt{m})$ as desired. We first show an approach in the case when the optimal actions $\boldsymbol{\theta}_t^*$ are close to a fixed point in Θ, i.e. for fixed $\boldsymbol{\psi}_t = \bar{\boldsymbol{\theta}}^* = \frac{1}{T}\boldsymbol{\theta}_{1:T}^*$. Henceforth we assume the Lipschitz constant G and number of rounds m are the same across tasks; detailed statements are in the supplement.

Note that if $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ then $\{f_t^{\text{init}}\}_t$ are quadratic functions, so playing $\boldsymbol{\phi}_{t+1} = \frac{1}{t}\boldsymbol{\theta}_{1:t}^*$ has logarithmic regret [Shalev-Shwartz, 2011, Corollary 2.2]. We use a novel strongly convex coupling argument to show that this holds for any such sequence of Bregman divergences, *even*

for nonconvex $\mathcal{B}_R(\boldsymbol{\theta}_t^*||\cdot)$. The second sequence $\{f_t^{\text{sim}}\}_t$ is harder because it is not smooth near 0 and not strongly convex if $\boldsymbol{\theta}_t^* = \boldsymbol{\phi}_t$. We study a regularized sequence $\tilde{f}_t^{\text{sim}}(v) = f_t^{\text{sim}}(v) + \varepsilon^2/v$ for $\varepsilon \geqslant 0$. Assuming a bound of D^2 on the Bregman divergence and setting $\varepsilon = 1/\sqrt[4]{T}$, we achieve $\tilde{\mathcal{O}}(\sqrt{T})$ regret on the original sequence by running the exponentially-weighted online-optimization (EWOO) algorithm of Hazan et al. [2007] on the regularized sequence:

$$v_t = \frac{\int_0^{\sqrt{D^2+\varepsilon^2}} v \exp(-\gamma \sum_{s<t} \tilde{f}_s^{\text{sim}}(v))dv}{\int_0^{\sqrt{D^2+\varepsilon^2}} \exp(-\gamma \sum_{s<t} \tilde{f}_s^{\text{sim}}(v))dv} \qquad \text{for} \qquad \gamma = \frac{2}{DG\sqrt{m}} \min\left\{\frac{\varepsilon^2}{D^2}, 1\right\} \tag{2.11}$$

Note that while EWOO is inefficient in high dimensions, we require only single-dimensional integrals. In the supplement we also show that simply setting $v_{t+1}^2 = \varepsilon^2 t + \sum_{s\leqslant t} \mathcal{B}_R(\boldsymbol{\theta}_s^*||\boldsymbol{\phi}_t)$ has only a slightly worse regret of $\tilde{\mathcal{O}}(T^{3/5})$. These guarantees suffice to show the following:

Theorem 2.2.2. Under the assumptions of Theorem 2.2.1 and boundedness of $\mathcal{B}_R$ over Θ, if `INIT` plays $\boldsymbol{\phi}_{t+1} = \frac{1}{t}\boldsymbol{\theta}_{1:t}^*$ and `SIM` uses ε-EWOO (2.11) with $\varepsilon = 1/\sqrt[4]{T}$ then Algorithm 1 achieves average regret

$$\overline{\text{Regret}} \leqslant \overline{U} = \tilde{\mathcal{O}}\left(\min\left\{\frac{1+\frac{1}{V}}{\sqrt{T}}, \frac{1}{\sqrt[4]{T}}\right\} + V\right)\sqrt{m} \qquad \text{for} \qquad V^2 = \min_{\boldsymbol{\phi}\in\Theta} \frac{1}{T}\sum_{t=1}^{T} \mathcal{B}_R(\boldsymbol{\theta}_t^*||\boldsymbol{\phi}) \tag{2.12}$$

Observe that if V, the average deviation of $\boldsymbol{\theta}_t^*$, is $\Omega_T(1)$ then the bound becomes $\mathcal{O}(V\sqrt{m})$ at rate $\tilde{\mathcal{O}}(1/\sqrt{T})$, while if $V = o_T(1)$ the bound tends to zero.

Related tasks in changing environments

In many settings we have a changing environment and so it is natural to study dynamic regret. This has been widely analyzed by the online learning community [Cesa-Bianchi et al., 2012, Jadbabaie et al., 2015], often by showing a dynamic regret bound consisting of a sublinear term plus a bound on the variation in the action or function space. Using Theorem 2.2.1 we can show dynamic guarantees for GBML via reduction to such bounds. We provide an example in the Euclidean geometry using the popular path-length-bound $P_\Psi = \sum_{t=2}^{T} \|\boldsymbol{\psi}_t - \boldsymbol{\psi}_{t-1}\|_2$ for reference actions $\Psi = \{\boldsymbol{\psi}_t\}_{t=1}^T$ [Zinkevich, 2003]. We use a result showing that OGD with learning rate $\eta \leqslant 1/\beta$ over α-strongly-convex, β-strongly-smooth, and L-Lipschitz functions has a bound of $\mathcal{O}(L(1+P_\Psi))$ on its dynamic regret [Mokhtari et al., 2016, Corollary 1]. Observe that in the case of $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ the sequence f_t^{init} in Theorem 2.2.1 consists of $DG\sqrt{m}$-Lipschitz quadratic functions. Thus using Theorem 2.2.1 we achieve the following:

Theorem 2.2.3. Under the assumptions of Theorem 2.2.1, bounded Θ, and $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$, if `INIT` is OGD with learning rate $\frac{1}{G\sqrt{m}}$, `SIM` uses ε-EWOO (2.11) with $\varepsilon = 1/\sqrt[4]{T}$, and $\Psi = \{\boldsymbol{\psi}_t\}_{t\in[T]} \subset \Theta$ is a comparator sequence, then by using OGD within-task Algorithm 1 achieves

$$\overline{\text{Regret}} \leqslant \overline{U} = \tilde{\mathcal{O}}\left(\min\left\{\frac{1+\frac{1}{V_\Psi}}{\sqrt{T}}, \frac{1}{\sqrt[4]{T}}\right\} + \min\left\{\frac{1+P_\Psi}{V_\Psi T}, \sqrt{\frac{1+P_\Psi}{T}}\right\} + V_\Psi\right)\sqrt{m} \tag{2.13}$$

for $V_\Psi^2 = \frac{1}{2T}\sum_{t=1}^{T} \|\boldsymbol{\theta}_t^* - \boldsymbol{\psi}_t\|_2^2$ and $P_\Psi = \sum_{t=2}^{T} \|\boldsymbol{\psi}_t - \boldsymbol{\psi}_{t-1}\|_2$.

This bound controls task-averaged regret using the deviation V_Φ of the optimal parameters $\boldsymbol{\theta}_t^*$ from some reference sequence Φ, which is assumed to vary slowly or sparsely so that the path length P_Φ is small. Figures 2.1 illustrates when such a guarantee improves over Theorem 2.2.2. Note that Theorem 2.2.3 specifies OGD as the meta-update algorithm `INIT`, so under the approximation that each task t's last iterate is close to $\boldsymbol{\theta}_t^*$ it suggests that simple GBML methods such as Reptile [Nichol et al., 2018] or FedAvg [McMahan et al., 2017] are adaptive. The generality of ARUBA also allows for incorporating other dynamic regret bounds [Hall and Willet, 2016, Zhang et al., 2017] and other non-static notions of regret [Hazan and Seshadri, 2009].

2.2.2 Adapting to the inter-task geometry

Previously we gave guarantees for learning OMD under a simple notion of task similarity: closeness of the optimal actions $\boldsymbol{\theta}_t^*$. We now turn to new algorithms that can adapt to a more sophisticated task similarity structure. Specifically, we study a class of learning algorithms parameterized by an initialization $\boldsymbol{\phi} \in \Theta$ and a symmetric positive-definite matrix $\mathbf{H} \in \mathbb{S}_+^d$ which plays

$$\boldsymbol{\theta}_{t,i} = \arg\min_{\boldsymbol{\theta}\in\Theta} \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\phi}\|_{\mathbf{H}^{-1}}^2 + \langle \nabla_{t,1:i-1}, \boldsymbol{\theta} \rangle \tag{2.14}$$

This corresponds $\boldsymbol{\theta}_{t,i+1} = \boldsymbol{\theta}_{t,i} - \mathbf{H}\nabla_{t,i}$, so if the optimal actions $\boldsymbol{\theta}_t^*$ vary strongly in certain directions, a matrix emphasizing those directions improves within-task performance. By strong-convexity of $\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\phi}\|_{\mathbf{H}^{-1}}^2$ w.r.t. $\|\cdot\|_{\mathbf{H}^{-1}}$, the regret-upper-bound is $U_t(\boldsymbol{\phi}, \mathbf{H}) = \frac{1}{2}\|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}\|_{\mathbf{H}^{-1}}^2 + \sum_{i=1}^m \|\nabla_{t,i}\|_{\mathbf{H}}^2$ [Shalev-Shwartz, 2011, Theorem 2.15]. We first study the diagonal case, i.e. learning a per-coordinate learning rate $\boldsymbol{\eta} \in \mathbb{R}^d$ to get iteration $\boldsymbol{\theta}_{t,i+1} = \boldsymbol{\theta}_{t,i} - \boldsymbol{\eta}_t \odot \nabla_{t,i}$. We propose to set $\boldsymbol{\eta}_t$ at each task t as follows:

$$\boldsymbol{\eta}_t = \sqrt{\frac{\sum_{s<t} \varepsilon_s^2 + \frac{1}{2}(\boldsymbol{\theta}_s^* - \boldsymbol{\phi}_s)^2}{\sum_{s<t} \zeta_s^2 + \sum_{i=1}^{m_s} \nabla_{s,i}^2}} \quad \text{for } \varepsilon_t^2 = \frac{\varepsilon^2}{(t+1)^p},\ \zeta_t^2 = \frac{\zeta^2}{(t+1)^p}\ \forall\, t \geqslant 0, \text{ where } \varepsilon, \zeta, p > 0 \tag{2.15}$$

Observe the similarity between this update AdaGrad [Duchi et al., 2011], which is also inversely related to the sum of the element-wise squares of all gradients seen so far. Our method adds multi-task information by setting the numerator to depend on the sum of squared distances between the initializations $\boldsymbol{\phi}_t$ set by the algorithm and that task's optimal action $\boldsymbol{\theta}_t^*$. This algorithm has the following guarantee:

Theorem 2.2.4. Let Θ be a bounded convex subset of $\mathbb{R}^d$, let $\mathbb{D}_+^d \subset \mathbb{R}^{d\times d}$ be the set of positive definite diagonal matrices, and let each task $t \in [T]$ consist of a sequence of m convex Lipschitz losses $\ell_{t,i} : \Theta \mapsto \mathbb{R}$. Suppose for each task t we run the iteration in Equation 2.14 with $\boldsymbol{\phi} = \frac{1}{t-1}\boldsymbol{\theta}_{1:t-1}^*$ and setting $\mathbf{H} = \operatorname{diag}(\boldsymbol{\eta}_t)$ via Equation 2.15 for $\varepsilon = 1, \zeta = \sqrt{m}$, and $p = \frac{2}{5}$. Then

$$\begin{aligned}\overline{\text{Regret}} &\leqslant \overline{U} \\ &= \min_{\substack{\boldsymbol{\phi}\in\Theta \\ \mathbf{H}\in\mathbb{D}_+^d}} \tilde{\mathcal{O}}\left(\sum_{j=1}^d \min\left\{\frac{\frac{1}{\mathbf{H}_{[j,j]}} + \mathbf{H}_{[j,j]}}{T^{\frac{2}{5}}}, \frac{1}{\sqrt[5]{T}}\right\}\right)\sqrt{m} + \frac{1}{T}\sum_{t=1}^T \frac{\|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}\|_{\mathbf{H}^{-1}}^2}{2} + \sum_{i=1}^m \|\nabla_{t,i}\|_{\mathbf{H}}^2\end{aligned} \tag{2.16}$$

Algorithm 2: Methods for modifying a generic GBML method to learn a per-coordinate step-size, with two variants: (1) the "ARUBA++" variant starts with $\boldsymbol{\eta}_{T,1} = \boldsymbol{\eta}_T$ and $\mathbf{g}_{T,1} = \mathbf{g}_T$, adaptively resets the learning rate by setting $\hat{\mathbf{g}}_{T,i+1} \leftarrow \hat{\mathbf{g}}_{T,i} + c\nabla_i^2$ for some $c > 0$, and then updates $\boldsymbol{\eta}_{T,i+1} \leftarrow \sqrt{\mathbf{b}_T/\mathbf{g}_{T,i+1}}$; (2) the "Isotropic" variant sets $\mathbf{b}_t$ and $\mathbf{g}_t$ to be scalars multiples of $\mathbf{1}_d$ that track the sum of squared distances and sum of squared gradient norms, respectively.

Input: T tasks, update method for meta-initialization, within-task descent method, settings $\varepsilon, \zeta, p > 0$

Initialize $\mathbf{b}_1 \leftarrow \varepsilon^2 \mathbf{1}_d$, $\mathbf{g}_1 \leftarrow \zeta^2 \mathbf{1}_d$

for task $t = 1, 2, \ldots, T$ **do**

Set $\boldsymbol{\phi}_t$ according to update method, $\boldsymbol{\eta}_t \leftarrow \sqrt{\mathbf{b}_t/\mathbf{g}_t}$

Run descent method from ϕ_t with learning rate η_t:

observe gradients $\nabla_{t,1}, \ldots, \nabla_{t,m_t}$

obtain within-task parameter $\hat{\boldsymbol{\theta}}_t$

$\mathbf{b}_{t+1} \leftarrow \mathbf{b}_t + \frac{\varepsilon^2 \mathbf{1}_d}{(t+1)^p} + \frac{1}{2}(\boldsymbol{\phi}_t - \hat{\boldsymbol{\theta}}_t)^2$

$\mathbf{g}_{t+1} \leftarrow \mathbf{g}_t + \frac{\zeta^2 \mathbf{1}_d}{(t+1)^p} + \sum_{i=1}^{m_t} \nabla_{t,i}^2$

Result: initialization $\boldsymbol{\phi}_T$, learning rate $\boldsymbol{\eta}_T = \sqrt{\mathbf{b}_T/\mathbf{g}_T}$

As $T \to \infty$ the bound converges to the minimum over $\boldsymbol{\phi}, \mathbf{H}$ of the last two terms, corresponding to using the optimal initialization and per-coordinate learning rate on every task. The $\mathcal{O}(T^{-2/5})$ convergence is slightly slower than the usual $\mathcal{O}(1/\sqrt{T})$ rate achieved in the previous section; this is due to the algorithm's adaptivity to within-task gradients, whereas previously we simply assumed a known Lipschitz bound G_t to set $\boldsymbol{\eta}_t$. This adaptivity makes the algorithm much more practical, leading to a method for adaptively learning a within-task learning rate using multi-task information; this is outlined in Algorithm 2 and shown to improve GBML performance in Section 2.2.4. Note also the per-coordinate separation of the left term, which shows that the algorithm converges more quickly on non-degenerate coordinates. The per-coordinate specification of $\boldsymbol{\eta}_t$ (2.15) can be further generalized to learning a full-matrix adaptive regularizer, for which we show guarantees in Theorem 2.2.5. However, the rate is much slower, and without further assumptions such methods will have $\Omega(d^2)$ computation and memory requirements.

Theorem 2.2.5. Let $\Theta \subset \mathbb{R}^d$ be bounded and convex, and let each task $t \in [T]$ be a sequence of m convex Lipschitz losses $\ell_{t,i} : \Theta \mapsto \mathbb{R}$. Suppose for each t we run the iteration in Equation 2.14 with $\boldsymbol{\phi} = \frac{1}{t-1}\boldsymbol{\theta}^*_{1:t-1}$ and $\mathbf{H}$ the unique positive definite solution of $\mathbf{B}_t^2 = \mathbf{H}\mathbf{G}_t^2\mathbf{H}$ for

$$\mathbf{B}_t^2 = t\varepsilon^2\mathbf{I}_d + \frac{1}{2}\sum_{s<t}(\boldsymbol{\theta}^*_s - \boldsymbol{\phi}_s)(\boldsymbol{\theta}^*_s - \boldsymbol{\phi}_s)^\top \qquad \text{and} \qquad \mathbf{G}_t^2 = t\zeta^2\mathbf{I}_d + \sum_{s<t}\sum_{i=1}^{m}\nabla_{s,i}\nabla_{s,i}^\top \tag{2.17}$$

for $\varepsilon = 1/\sqrt[8]{T}$ and $\zeta = \sqrt{m}/\sqrt[8]{T}$. Then for $\lambda_j(\mathbf{H})$ the jth largest eigenvalue of $\mathbf{H}$ we have

$$\overline{\text{Regret}} \leqslant \overline{U} = \tilde{\mathcal{O}}\left(\frac{1}{\sqrt[8]{T}}\right)\sqrt{m} + \min_{\substack{\boldsymbol{\phi}\in\Theta \\ \mathbf{H}>0}} \frac{2\lambda_1^2(\mathbf{H})}{\lambda_d(\mathbf{H})}\frac{1+\log T}{T} + \sum_{t=1}^{T}\frac{\|\boldsymbol{\theta}^*_t - \boldsymbol{\phi}^*\|^2_{\mathbf{H}^{-1}}}{2} + \sum_{i=1}^{m}\|\nabla_{t,i}\|^2_{\mathbf{H}} \tag{2.18}$$

2.2.3 Fast rates and high probability bounds for statistical meta-learning

Transfer risk bounds in the distributional setting have been an important motivation for studying LTL via online learning [Alquier et al., 2017, Denevi et al., 2019]. If the regret-upper-bounds are convex, which is true for most practical variants of OMD/FTRL, ARUBA yields several new results in the classical distribution over task distributions setup of Baxter [2000]. In Theorem 2.2.6 we present bounds on the risk $\ell_{\mathcal{P}}(\bar{\boldsymbol{\theta}})$ of the parameter $\bar{\boldsymbol{\theta}}$ obtained by running OMD/FTRL on i.i.d. samples from a new task distribution $\mathcal{P}$ and averaging the iterates.

Theorem 2.2.6. Assume $\Theta, \mathcal{X}$ are convex subsets of a Euclidean vector space. Let convex losses $\ell_{t,i} : \Theta \mapsto [0,1]$ be drawn i.i.d. $\mathcal{P}_t \sim \mathcal{Q}, \{\ell_{t,i}\}_i \sim \mathcal{P}_t^m$ for distribution $\mathcal{Q}$ over tasks. Suppose they are passed to an algorithm with average regret upper-bound $\overline{U}$ after T tasks that at each t picks $\mathbf{x}_t \in \mathcal{X}$ to initialize a within-task method with convex regret upper-bound $U_t : \mathcal{X} \mapsto [0, B\sqrt{m}]$, for $B \geqslant 0$. If the within-task algorithm is initialized by $\bar{\mathbf{x}} = \frac{1}{T}\mathbf{x}_{1:T}$ and it takes actions $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m$ on m i.i.d. losses from new task $\mathcal{P} \sim \mathcal{Q}$ then $\bar{\boldsymbol{\theta}} = \frac{1}{m}\boldsymbol{\theta}_{1:m}$ satisfies the following transfer risk bounds for any $\boldsymbol{\theta}^* \in \Theta$ (all w.p. $1-\delta$):

1. **general case:** $\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\ell_{\mathcal{P}}(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\ell_{\mathcal{P}}(\boldsymbol{\theta}^*) + \mathcal{L}_T$ for $\mathcal{L}_T = \frac{\overline{U}}{m} + B\sqrt{\frac{8}{mT}\log\frac{1}{\delta}}$.
2. **ρ-self-bounded losses ℓ:** if $\exists\ \rho > 0$ s.t. $\rho\mathbb{E}_{\ell\sim\mathcal{P}}\Delta\ell(\boldsymbol{\theta}) \geqslant \mathbb{E}_{\ell\sim\mathcal{P}}(\Delta\ell(\boldsymbol{\theta}) - \mathbb{E}_{\ell\sim\mathcal{P}}\Delta\ell(\boldsymbol{\theta}))^2$ for all distributions $\mathcal{P} \sim \mathcal{Q}$, where $\Delta\ell(\boldsymbol{\theta}) = \ell(\boldsymbol{\theta}) - \ell(\boldsymbol{\theta}^*)$ for any $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}\in\Theta}\ell_{\mathcal{P}}(\boldsymbol{\theta})$, then for $\mathcal{L}_T$ as above we have $\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\ell_{\mathcal{P}}(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\ell_{\mathcal{P}}(\boldsymbol{\theta}^*) + \mathcal{L}_T + \sqrt{\frac{2\rho\mathcal{L}_T}{m}\log\frac{2}{\delta}} + \frac{3\rho+2}{m}\log\frac{2}{\delta}$.
3. **α-strongly-convex, G-Lipschitz regret-upper-bounds U_t:** in parts 1 and 2 above we can substitute $\mathcal{L}_T = \frac{\overline{U}+\min_{\mathbf{x}}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}U(\mathbf{x})}{m} + \frac{4G}{T}\sqrt{\frac{\overline{U}}{\alpha m}\log\frac{8\log T}{\delta}} + \frac{\max\{16G^2, 6\alpha B\sqrt{m}\}}{\alpha mT}\log\frac{8\log T}{\delta}$.

In the **general case**, Theorem 2.2.6 provides bounds on the excess transfer risk decreasing with $\overline{U}/m$ and $1/\sqrt{mT}$. Thus if $\overline{U}$ improves with task similarity so will the transfer risk as $T \to \infty$. Note that the second term is $1/\sqrt{mT}$ rather than $1/\sqrt{T}$ as in some past most-analyses [Denevi et al., 2019]; this is because regret is m-bounded but the OMD regret-upper-bound is $\mathcal{O}(\sqrt{m})$-bounded. The results also demonstrate ARUBA's ability to utilize specialized results from the online-to-batch conversion literature. This is witnessed by the guarantee for **self-bounded losses**, a class which Zhang [2005] shows includes linear regression; we use a result by the same author to obtain high-probability bounds, whereas previous GBML bounds are in-expectation [Denevi et al., 2019]. We also apply a result due to Kakade and Tewari [2008] for the case of **strongly-convex regret-upper-bounds**, enabling fast rates in the number of tasks T. The strongly-convex case is especially relevant for GBML since it holds for OGD with fixed learning rate.

We present two consequences of these results for the algorithms from Section 2.2.1 when run on i.i.d. data. To measure task similarity we use the **variance** $V_{\mathcal{Q}}^2 = \min_{\boldsymbol{\phi}\in\Theta}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\|\boldsymbol{\theta}^* - \boldsymbol{\phi}\|_2^2$ of the empirical risk minimizer $\boldsymbol{\theta}^*$ of an m-sample task drawn from $\mathcal{Q}$. If $V_{\mathcal{Q}}$ is known we can use strong-convexity of the regret-upper-bounds to obtain a fast rate for learning the initialization, as shown in the first part of Corollary 2.2.2. The result can be loosely compared to Denevi et al. [2019], who provide a similar asymptotic improvement but with a slower rate of $\mathcal{O}(1/\sqrt{T})$ in the second term. However, their task similarity measures the deviation of the true, not empirical, risk minimizers, so the results are not directly comparable. Corollary 2.2.2 also gives a guarantee for when we do *not* know $V_{\mathcal{Q}}$ and must learn the learning rate η in addition to the initialization;

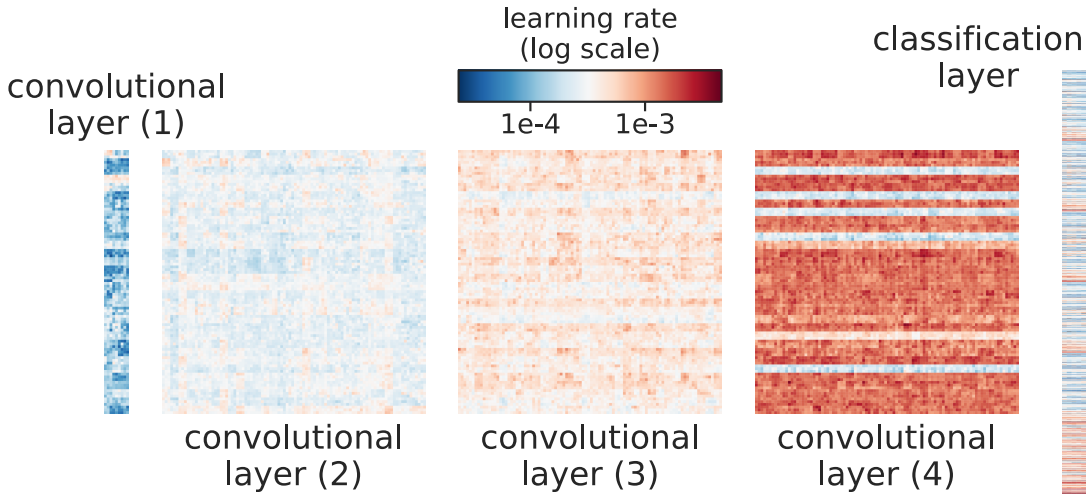

Figure 2.2: Learning rate variation across layers of a convolutional net trained on Mini-ImageNet using Algorithm 2. Following intuition outlined in Section 2.2.4, shared feature extractors are not updated much if at all compared to higher layers.

here we match the rate of Denevi et al. [2019], who do not learn η, up to some additional fast $o(1/\sqrt{m})$ terms.

Corollary 2.2.2. In the setting of Theorems 2.2.2 and 2.2.6, if $\delta \leqslant 1/e$ and Algorithm 1 uses within-task OGD with initialization $\boldsymbol{\phi}_{t+1} = \frac{1}{t}\boldsymbol{\theta}^*_{1:t}$ and step-size $\eta_t = \frac{V_{\mathcal{Q}}+1/\sqrt{T}}{G\sqrt{m}}$ for $V_{\mathcal{Q}}$ as above, then w.p. $1-\delta$

$$\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\ell_{\mathcal{P}}(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\ell_{\mathcal{P}}(\boldsymbol{\theta}^*) + \tilde{\mathcal{O}}\left(\frac{V_{\mathcal{Q}}}{\sqrt{m}} + \left(\frac{1}{\sqrt{mT}} + \frac{1}{T}\right)\log\frac{1}{\delta}\right) \tag{2.19}$$

If η_t is set adaptively using ε-EWOO as in Theorem 2.2.2 for $\varepsilon = 1/\sqrt[4]{mT} + 1/\sqrt{m}$ then w.p. $1-\delta$

$$\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\ell_{\mathcal{P}}(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\ell_{\mathcal{P}}(\boldsymbol{\theta}^*) + \tilde{\mathcal{O}}\left(\frac{V_{\mathcal{Q}}}{\sqrt{m}} + \min\left\{\frac{\frac{1}{\sqrt{m}}+\frac{1}{\sqrt{T}}}{V_{\mathcal{Q}}m}, \frac{1}{\sqrt[4]{m^3T}} + \frac{1}{m}\right\} + \sqrt{\frac{1}{T}\log\frac{1}{\delta}}\right) \tag{2.20}$$

2.2.4 Empirical results for few-shot and federated learning

A generic GBML method does the following at iteration t: (1) initialize a descent method at $\boldsymbol{\phi}_t$; (2) take gradient steps with learning rate η to get task parameter $\hat{\boldsymbol{\theta}}_t$; (3) update meta-initialization to $\boldsymbol{\phi}_{t+1}$. Motivated by Section 2.2.2, in Algorithm 2 we outline a generic way of replacing η by a per-coordinate rate learned on-the-fly. This entails keeping track of two quantities: (1) $\mathbf{b}_t \in \mathbb{R}^d$, a per-coordinate sum over $s < t$ of the squared distances from the initialization $\boldsymbol{\phi}_s$ to within-task parameter $\hat{\boldsymbol{\theta}}_s$; (2) $\mathbf{g}_t \in \mathbb{R}^d$, a per-coordinate sum of the squared gradients seen so far. At task t we set η to be the element-wise square root of $\mathbf{b}_t/\mathbf{g}_t$, allowing multi-task information to inform the trajectory. For example, if along coordinate j the $\hat{\boldsymbol{\theta}}_{t[j]}$ is usually not far from initialization

Table 2.1: Meta-test-time performance of GBML algorithms on few-shot classification benchmarks. 1st-order and 2nd-order results obtained from Nichol et al. [2018] and Li et al. [2017], respectively.

		20-way Omniglot		5-way Mini-ImageNet	
		1-shot	5-shot	1-shot	5-shot
1st Order	1st-Order MAML	89.4 ± 0.5	$\underline{97.9} \pm 0.1$	48.07 ± 1.75	63.15 ± 0.91
	Reptile w. Adam	89.43 ± 0.14	97.12 ± 0.32	49.97 ± 0.32	$\underline{\mathbf{65.99}} \pm 0.58$
	Reptile w. ARUBA	86.67 ± 0.17	96.61 ± 0.13	$\underline{\mathbf{50.73}} \pm 0.32$	65.69 ± 0.61
	Reptile w. ARUBA++	$\underline{89.66} \pm 0.3$	97.49 ± 0.28	50.35 ± 0.74	65.89 ± 0.34
2nd Order	2nd-Order MAML	95.8 ± 0.3	98.9 ± 0.2	48.7 ± 1.84	63.11 ± 0.92
	Meta-SGD [Li et al., 2017]	$\underline{\mathbf{95.93}} \pm 0.38$	$\underline{\mathbf{98.97}} \pm 0.19$	$\underline{50.47} \pm 1.87$	$\underline{64.03} \pm 0.94$

then $\mathbf{b}_{[j]}$ will be small and thus so will $\boldsymbol{\eta}_{[j]}$; then if on a new task we get a high noisy gradient along coordinate j the performance will be less adversely affected because it will be down-weighted by the learning rate. Single-task algorithms such as AdaGrad [Duchi et al., 2011] and Adam [Kingma and Ba, 2015] also work by reducing the learning rate along frequent directions. However, in meta-learning some coordinates may be frequently updated during meta-training because good task weights vary strongly from the best initialization along them, and thus their gradients should not be downweighted; ARUBA encodes this intuition in the numerator using the distance-traveled per-task along each direction, which increases the learning rate along high-variance directions. We show in Figure 2.2 that this is realized in practice, as ARUBA assigns a faster rate to deeper layers than to lower-level feature extractors, following standard intuition in parameter-transfer meta-learning. As described in Algorithm 2, we also consider two variants: ARUBA++, which updates the meta-learned learning-rate at meta-test-time in a manner similar to AdaGrad, and Isotropic ARUBA, which only tracks scalar quantities and is thus useful for communication-constrained settings.

Few-shot classification

We first examine if Algorithm 2 can improve performance on Omniglot [Lake et al., 2017] and Mini-ImageNet [Ravi and Larochelle, 2017], two standard few-shot learning benchmarks, when used to modify Reptile, a simple meta-learning method [Nichol et al., 2018]. In its serial form Reptile is roughly the algorithm we study in Section 2.2.1 when OGD is used within-task and η is fixed. Thus we can set Reptile+ARUBA to be Algorithm 2 with $\hat{\boldsymbol{\theta}}_t$ the last iterate of OGD and the meta-update a weighted sum of $\hat{\boldsymbol{\theta}}_t$ and $\boldsymbol{\phi}_t$. In practice, however, Reptile uses Adam [Kingma and Ba, 2015] to exploit multi-task gradient information. As shown in Table 2.1, ARUBA matches or exceeds this baseline on Mini-ImageNet, although on Omniglot it requires the additional within-task updating of ARUBA++ to show improvement.

It is less clear how ARUBA can be applied to MAML [Finn et al., 2017], as by only taking one step the distance traveled will be proportional to the gradient, so η will stay fixed. We also do not find that ARUBA improves multi-step MAML, which is perhaps not surprising as it is further removed from our theory due to its use of held-out data. In Table 2.1 we compare to Meta-SGD [Li et al., 2017], which does learn a per-coordinate learning rate for MAML by auto-

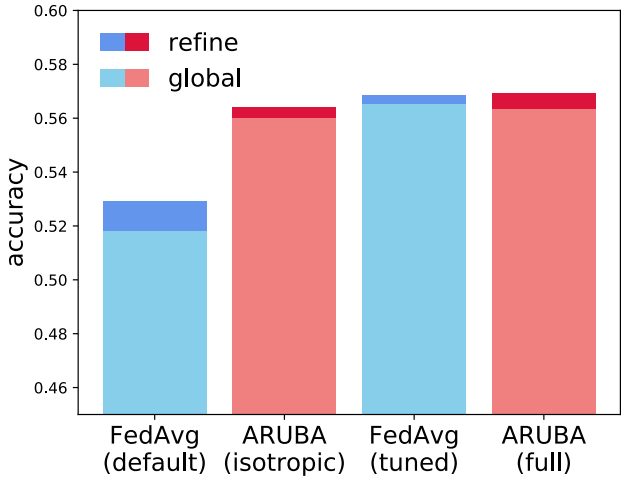

Figure 2.3: Next-character prediction performance for recurrent networks trained on the Shakespeare dataset [Caldas et al., 2018] using FedAvg [McMahan et al., 2017] and its modifications by Algorithm 2. Note that the two ARUBA methods require no learning rate tuning when personalizing the model (refine), unlike *both* FedAvg methods; this is a critical improvement in federated settings. Furthermore, isotropic ARUBA has negligible overhead by only communicating scalars.

matic differentiation. This requires more computation but does lead to consistent improvement. As with the original Reptile, our modification performs better on Mini-ImageNet but worse on Omniglot compared to MAML and its modification Meta-SGD.

Federated learning

A main goal in this setting is to use data on heterogeneous nodes to learn a global model without much communication; leveraging this to get a personalized model is an auxiliary goal [Smith et al., 2017], with a common application being next-character prediction on mobile devices. A popular method is FedAvg [McMahan et al., 2017], where at each communication round r the server sends a global model $\boldsymbol{\phi}_r$ to a batch of nodes, which then run local OGD; the server then sets $\boldsymbol{\phi}_{r+1}$ to the average of the returned models. This can be seen as a GBML method with each node a task, making it easy to apply ARUBA: each node simply sends its accumulated squared gradients to the server together with its model. The server can use this information and the squared difference between $\boldsymbol{\phi}_r$ and $\boldsymbol{\phi}_{r+1}$ to compute a learning rate η_{r+1} via Algorithm 2 and send it to each node in the next round. We use FedAvg with ARUBA to train a character LSTM [Hochreiter and Schmidhuber, 1997] on the Shakespeare dataset, a standard benchmark of a thousand users with varying amounts of non-i.i.d. data [McMahan et al., 2017, Caldas et al., 2018]. Figure 2.3 shows that ARUBA significantly improves over non-tuned FedAvg and matches the performance of FedAvg with a tuned learning rate schedule. Unlike both baselines we also do not require step-size tuning when refining the global model for personalization. This reduced need for hyperparameter optimization is crucial in federated settings, where the number of user-data accesses are extremely limited.

2.2.5 Conclusion

In this section we have demonstrated the application of ARUBA for analyzing gradient-based meta-learning, yielding new guarantees and algorithms for adaptive, dynamic, and statistical LTL via online learning. ARUBA has significant potential to yield many other new LTL methods in a similar manner, but so far our results were for convex, Lipschitz, and full-information loss functions, assumptions that do not hold in many potential applications where we might want to learn-to-learn across tasks. We thus devote the next two sections to exploring how these assumptions might be avoided, first by showing guarantees for the meta-learning of online algorithms over nonconvex piecewise-Lipschitz functions and second by studying adversarial bandit algorithms. In doing so we go beyond *gradient-based* learning algorithms as well, demonstrating that the idea of targeting surrogate performance bounds can be useful even when moving to more probabilistic algorithms.

2.3 Learning-to-learn piecewise-Lipschitz functions

Our first direction is the meta-learning of online learners of piecewise-Lipschitz functions, which can be nonconvex and highly discontinuous. As no-regret learning over such functions is impossible in-general, we study the case of piecewise-Lipschitz functions whose discontinuities are *dispersed*, i.e. which do not concentrate in any small compact subset of the domain [Balcan et al., 2018b]. Such functions arise frequently in *data-driven algorithm design*, in which the goal is to learn the optimal parameter settings of algorithms for difficult (often NP-hard) problems over a distribution or sequence of instances [Balcan, 2021]; for example, a small change to the metric used in cluster linkage can lead to a discontinuous change in classification error [Balcan et al., 2019]. Such problems are often solved across many time periods or problem domains, resulting in natural multi-task structure that we might hope to use to improve performance.

In the single-task setting the problem of learning dispersed functions can be solved using simple methods such as the exponentially-weighted forecaster. To design an algorithm for learning to initialize online learners in this setting, we apply ARUBA to optimize a sequence of data-dependent upper-bounds on the within-task regret. The result is an averaged bound that improves upon the regret of the single-task exponential forecaster so long as there exists an initial distribution that can compactly contain many of the within-task optima of the different tasks. Designing the meta-procedure is especially challenging in our setting because it involves online learning over a set of distributions on the domain. To handle this we study a "prescient" form of the classic follow-the-regularized leader (FTRL) scheme that is run over an unknown discretization; we then show the existence of another algorithm that plays the same actions but uses only known information, thus attaining the same regret while being practical to implement.

As an application, we consider data-driven tuning of the parameters of combinatorial optimization algorithms for hard problems such as knapsack and clustering. The likely intractability of these problems on worst case instances have led to several approaches to study them in more realistic settings, such as smoothed analysis [Spielman and Teng, 2004] and data-driven algorithm configuration [Balcan, 2021]. Our setting is more realistic than those considered in prior work. It is more challenging than learning from i.i.d. instances [Gupta and Roughgar-

den, 2017], but at the same time less pessimistic than online learning over adversarial problem instances [Balcan et al., 2018b], as it allows us to leverage similarity of problem instances coming from different but related distributions. We instantiate our bounds theoretically on several problems where the cost functions are piecewise-constant in the tuned parameters, allowing our meta-procedure to learn the right initial distribution for exponential forecasters. This includes well-known combinatorial optimization problems like finding the maximum weighted independent set (MWIS) of vertices on a graph, solving quadratic programs with integer constraints using algorithms based on the celebrated Goemans-Williamson algorithm, and mechanism design for combinatorial auctions. Then we consider experimentally the problem of tuning the right α for the α-Lloyd's family of clustering algorithms [Balcan et al., 2018c]. In experimental evaluations on two datasets—a synthetic Gaussian mixture model and the well-known Omniglot dataset from meta-learning [Lake et al., 2017]—our meta-procedure leads to improved clustering accuracy compared to single-task learning to cluster. The results holds for both one-shot and five-shot clustering tasks. We also study our results for a family of greedy algorithms for the knapsack problem introduced by Gupta and Roughgarden [2017] and obtain similar results.

2.3.1 Related work

Most learning-theoretic results for initialization-based meta-learning have been in the convex Lipschitz setting, with work on inherently *nonconvex* modeling approaches usually focuseing on multi-task representation learning [Balcan et al., 2015, Maurer et al., 2016, Du et al., 2021b, Tripuraneni et al., 2021] or targeting optimization, e.g. stationary point convergence [Fallah et al., 2020]. An exception is a study of linear models over Gaussian data showing that nonconvexity is critical to meta-learning an initialization that exploits low-rank task structure [Saunshi et al., 2020]. There is also work extending results from the neural tangent kernel literature to meta-learning [Zhou et al., 2021], but in this case the objective becomes convex. On the other hand, we study initializations for learning a class of functions that can be highly nonconvex and have numerous discontinuities.

2.3.2 Initialization-dependent learning of dispersed functions

In this section we introduce our setup for online learning of piecewise-Lipschitz functions in a multi-task setting. We then generalize existing results for the single-task setting in order to obtain within-task regret bounds that depend on both the initialization and the task data. This is critical for both defining a notion of task similarity and devising a meta-learning procedure.

Meta-learning setup

As before, for some $T, m > 0$ and all $t \in [T]$ and $i \in [m]$ we consider a meta-learner faced with a sequence of Tm loss functions $\ell_{t,i} : C \mapsto [0, 1]$ over a compact subset $C \subset \mathbb{R}^d$ that lies within a ball $\mathbb{B}(\boldsymbol{\rho}, R)$ of radius R around some point $\boldsymbol{\rho} \in \mathbb{R}^d$. Before each loss function $\ell_{t,i}$ the meta-learner must pick an element $\boldsymbol{\rho}_{t,i} \in C$ before then suffering a loss or cost $\ell_{t,i}(\boldsymbol{\rho}_{t,i})$. As in the previous section, the subsequence $\ell_{t,1}, \ldots, \ell_{t,m}$ defines a task, but a key point now is that the tasks we consider can have numerous global optima. We will assume, after going through the

m rounds of task t, that we have oracle access to a single fixed optimum for t, which we will refer to using $\boldsymbol{\rho}_t^*$ and use in both our algorithm and to define the task similarity. Note that in the types of applications we are interested in—piecewise-Lipschitz functions—the complexity of computing optima scales with the number of discontinuities. In the important special case of piecewise-constant functions, this dependency becomes logarithmic [Cohen-Addad and Kanade, 2017]. Thus this assumption does not affect the usefulness of the result.

Our goal will be to improve the guarantees for regret in the single-task case by using information obtained from solving multiple tasks. As before, we will do this by proving task similarity-dependent bounds on the task-averaged regret $\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m}\ell_{t,i}(\boldsymbol{\rho}_{t,i}) - \ell_{t,i}(\boldsymbol{\rho}_t^*)$ and claim improvement over single-task learning if in the limit of $T \to \infty$ it is better than the best available bounds on the single-task regret. Note that for simplicity we assume all tasks have the same number of rounds within-task, but as with past work our results are straightforward to extend to the more general setting.

Learning piecewise-Lipschitz functions

We now turn to our target functions and within-task algorithms for learning them: piecewise-Lipschitz losses, i.e. functions that are L-Lipschitz w.r.t. the Euclidean norm everywhere except on measure zero subsets of the space; here they may have arbitrary jump discontinuities so long they still bounded between $[0, 1]$. Apart from being a natural setting of interest due to its generality compared to past work on meta-learning, this class of functions has also been shown to have important applications in data-driven algorithm configuration [Balcan et al., 2018b]; there these functions represent the cost, e.g. an objective value or time-complexity, of algorithms for difficult problems such as integer programming, auction design, and clustering.

This literature has also shown lower bounds demonstrating that no-regret learning piecewise-Lipschitz function is impossible in general, necessitating assumptions about the sequence. One such condition is *dispersion*, which requires that the discontinuities are not too concentrated.

Definition 2.3.1 (Balcan et al. [2018b])**.** The sequence of random loss functions $\ell_1, \ldots, \ell_m$ is said to be β-**dispersed** with Lipschitz constant L if, for all m and for all $\epsilon \geqslant m^{-\beta}$, we have that, in expectation over the randomness of the functions, at most $\tilde{O}(\epsilon m)$ functions (the soft-O notation suppresses dependence on quantities beside ϵ, m and β, as well as logarithmic terms) are not L-Lipschitz for any pair of points at distance ϵ in the domain $\mathcal{C}$. That is, for all m and for all $\varepsilon \geqslant m^{-\beta}$,

$$\mathbb{E}\left[\max_{\substack{\boldsymbol{\rho},\boldsymbol{\rho}'\in\mathcal{C}\\ \|\boldsymbol{\rho}-\boldsymbol{\rho}'\|_2\leqslant\epsilon}} \left|\left\{i \in [m] \mid \ell_i(\boldsymbol{\rho}) - \ell_i(\boldsymbol{\rho}') > L\|\boldsymbol{\rho}-\boldsymbol{\rho}'\|_2\right\}\right|\right] \leqslant \tilde{O}(\epsilon m) \tag{2.21}$$

Assuming a sequence of m β-dispersed loss functions and initial distribution w_1 set to the uniform distribution over C and optimize the step-size parameter, the exponential forecaster presented in Algorithm 3 achieves sublinear regret $\tilde{O}(\sqrt{dm\log(Rm)} + (L+1)m^{1-\beta})$. While this result achieves a no-regret procedure, its lack of dependence on both the task data and on the chosen initialization makes it difficult to meta-learn. In the following theorem, we generalize the regret bound for the exponential forecaster to make it data-dependent and hyperparameter-dependent:

Algorithm 3: Exponential Forecaster

Input: step-size parameter $\lambda \in (0, 1]$, initialization $w : C \to \mathbb{R}_{\geqslant 0}$.
Initialize $w_1 \leftarrow w$ **for** $i = 1, 2, \ldots, m$ **do**
  $W_i \leftarrow \int_C w_i(\boldsymbol{\rho}) d\boldsymbol{\rho}$
  Sample $\boldsymbol{\rho}_i$ with probability proportional to $w_i(\boldsymbol{\rho}_i)$, i.e. with probability $p_i(\boldsymbol{\rho}_i) = \frac{w_i(\boldsymbol{\rho}_i)}{W_i}$
  Suffer $\ell_i(\boldsymbol{\rho}_i)$ and observe $\ell_i(\cdot)$
  For each $\boldsymbol{\rho} \in C$, set $w_{i+1}(\boldsymbol{\rho}) = e^{-\lambda \ell_i(\boldsymbol{\rho})} w_i(\boldsymbol{\rho})$

Theorem 2.3.1. Let $\ell_1, \ldots, \ell_m : C \mapsto [0, 1]$ be any sequence of piecewise L-Lipschitz functions that are β-dispersed. Suppose $C \subset \mathbb{R}^d$ is contained in a ball of radius R. The exponentially weighted forecaster (Algorithm 3) has expected regret $R_m \leqslant m\lambda + \frac{\log(1/Z)}{\lambda} + \tilde{O}((L+1)m^{1-\beta})$, where $Z = \frac{\int_{\mathbb{B}(\boldsymbol{\rho}^*, m^{-\beta})} w(\boldsymbol{\rho}) d\boldsymbol{\rho}}{\int_C w(\boldsymbol{\rho}) d\boldsymbol{\rho}}$ for $\boldsymbol{\rho}^*$ the optimal action in hindsight.

The proof of this result adapts past analyses of Algorithm 3; setting step-size λ appropriately recovers the previously mentioned bound. The new bound is useful due to its explicit dependence on both the initialization w and the optimum in hindsight via the $\log(1/Z)$ term. Assuming w is a (normalized) distribution, this effectively measures the overlap between the chosen initialization and a small ball around the optimum; we thus call

$$-\log Z = -\log \frac{\int_{\mathbb{B}(\boldsymbol{\rho}^*, m^{-\beta})} w(\boldsymbol{\rho}) d\boldsymbol{\rho}}{\int_C w(\boldsymbol{\rho}) d\rho} \tag{2.22}$$

the **negative log-overlap** of initialization $w(\cdot)$ with the optimum $\boldsymbol{\rho}^*$.

We also obtain an asymptotic lower bound of $\tilde{\Omega}(m^{1-\beta})$ on the expected regret of any algorithm by extending the argument of Balcan et al. [2020b] to the multi-task setting; this shows a limit on any improvement we can hope to achieve from task similarity.

Theorem 2.3.2. There is a sequence of piecewise L-Lipschitz β-dispersed losses $\ell_{i,j} : [0, 1] \mapsto [0, 1]$ whose optimal actions in hindsight $\arg\min_\rho \sum_{i=1}^m l_{t,i}(\rho)$ are contained in some fixed ball of diameter D^*, for which any algorithm has expected regret $R_m \geqslant \tilde{\Omega}(m^{1-\beta})$.

Task similarity

Before proceeding to our discussion of meta-learning, we first discuss what we might hope to achieve with it; specifically, we consider what a reasonable notion of task similarity is in this setting. Note that the Theorem 2.3.1 regret bound has three terms, of which two depend on the hyperparameters and the last is due to dispersion and cannot be improved via better settings. Our focus will thus be on improving the first two terms, which are the dominant ones due to the dependence on the dimensionality and the distance from the initialization encoded in the negative log overlap. In particular, when the initialization is the uniform distribution then this quantity depends inversely on the size of a small ball around the optimum, which may be quite small. Via meta-learning we hope to assign more of the probability mass of the initializer to areas close

to the optimum, which will decrease these terms. On average, rather than a dependence on the volume of a small ball we aim to achieve a dependence on the **average negative log-overlap**

$$V^2 = -\min_{w:C\mapsto\mathbb{R}_{\geqslant 0}, \int_C w(\boldsymbol{\rho})d\boldsymbol{\rho}=1} \frac{1}{T}\sum_{t=1}^{T} \log \int_{\mathbb{B}(\boldsymbol{\rho}_t^*, m^{-\beta})} w(\boldsymbol{\rho})d\boldsymbol{\rho} \tag{2.23}$$

which can be much smaller if the task optima $\boldsymbol{\rho}_t^*$ are close together; for example, if they are the same then $V = 0$, corresponding to assigning all the initial weight within the common ball $\mathbb{B}(\boldsymbol{\rho}^*, m^{-\beta})$ around the shared optima. This is also true if $\mathrm{vol}(\cap_{t\in T}\mathbb{B}(\boldsymbol{\rho}_t^*, m^{-\beta})) > 0$, as one can potentially initialize with all the weight in the intersection of the balls. On the other hand if $\mathrm{vol}(\cap_{t\in T}\mathbb{B}(\boldsymbol{\rho}_t^*, m^{-\beta})) = 0$, $V > 0$. For example, if a p-fraction of tasks have optima $\boldsymbol{\rho}_0$ and the remaining at $\boldsymbol{\rho}_1$ with $||\boldsymbol{\rho}_0 - \boldsymbol{\rho}_1|| > 2m^{-\beta}$ the task similarity is given by the binary entropy function $V = H_b(p) = -p\log p - (1-p)\log(1-p)$.

The settings of Algorithm 3 that achieve the minimum in the definition of V are directly related to V itself: the optimal initializer is the distribution achieving V and the optimal step-size is $V/\sqrt{m}$. Note that while the explicit definition requires computing a minimum over a set of functions, the task similarity can be computed using the discretization constructed in Section 2.3.3.

2.3.3 An algorithm for meta-learning the initialization and step-size

Having established a single-task algorithm whose regret depends on the initialization and step-size, we move on to meta-learning these hyperparameters. Recall that the goal is to make the task-averaged regret (2.2) small and improve upon the single-task baseline of repeatedly running Algorithm 3 from the uniform distribution, up to $o_T(1)$ terms that vanish as we see more tasks. In this section, we use the ARUBA strategy of online learning our data-dependent regret guarantees; if we can do so with regret sublinear in T then we will improve upon the single-task guarantees up to $o_T(1)$ terms, as desired. Specifically, we are faced with a sequence of regret-upper-bounds $U_t(w, v) = (v + f_t(w)/v)\sqrt{m} + g(m)$ on nonnegative functions w over C and positive scalars $v > 0$. As $g(m)$ cannot be improved via meta-learning, we will focus on learning w and v. To do so, we run two online algorithms, one over the functions f_t and the other over $h_t(v) = v + f_t(w_t)/v$, where w_t is set by the first procedure. The following shows that if both procedures have sublinear regret then our task-averaged regret will have the desired properties:

Theorem 2.3.3. Assume each task $t \in [T]$ consists of a sequence of m β-dispersed piecewise L-Lipschitz functions $\ell_{t,i} : C \mapsto [0, 1]$. Let f_t and g be functions such that the regret of Algorithm 3 run with step-size $\lambda = v\sqrt{m}$ for $v > 0$ and initialization $w : C \mapsto \mathbb{R}_{\geqslant 0}$ is bounded by $U_t(w, v) = (v + f_t(w)/v)\sqrt{m} + g(m)$. Suppose we have a procedure that achieves $F_T(w)$ regret w.r.t. any $w : C \mapsto \mathbb{R}_{\geqslant 0}$ by playing actions $w_t : C \mapsto \mathbb{R}_{\geqslant 0}$ on f_t and another procedure that achieves $H_T(v)$ regret w.r.t. any $v > 0$ by playing actions $v_t > 0$ on $h_t(v) = v + f_t(w_t)/v$, where H_T is non-increasing on the positive reals. Then by setting $\boldsymbol{\rho}_{t,i}$ using Algorithm 3 with step-size $v_t/\sqrt{m}$ and initialization w_t at each task t we get task-averaged regret bounded by

$$\left(\frac{H_T(V)}{T} + \min\left\{\frac{F_T(w^*)}{VT}, 2\sqrt{F_T(w^*)/T}\right\} + 2V\right)\sqrt{m} + g(m) \tag{2.24}$$

for $w^* = \arg\min_{w:C\mapsto\mathbb{R}_{\geqslant 0}} \sum_{t=1}^{T} f_t(w)$ the optimal initialization and V the task similarity (2.23).

This result is an analog of Theorem 2.2.2 and follows by manipulating the definition of regret. It reduces the problem of obtaining a small task-averaged regret to solving two online learning problems, one to set the initialization and one to set the step-size. So long as both have sublinear regret then we will improve over single-task learning. In the next two sections we derive suitable procedures.

Meta-learning the initialization

We now come to the most technically challenging component of our meta-learning procedure: learning the initialization. As discussed above, we can accomplish this by obtaining a no-regret procedure for the function sequence

$$f_t(w) = -\log \frac{\int_{\mathbb{B}(\boldsymbol{\rho}_t^*, m^{-\beta})} w(\boldsymbol{\rho}) d\boldsymbol{\rho}}{\int_C w(\boldsymbol{\rho}) d\boldsymbol{\rho}} \tag{2.25}$$

This is nontrivial as the optimization domain is a set of nonnegative functions, effectively measures on the domain C. To handle this, we first introduce some convenient notation and abstractions. At each task t we are faced with some function f_t associated with an unknown closed subset $C_t \subset C$—in particular $C_t = \mathbb{B}(\boldsymbol{\rho}_t^*, m^{-\beta})$—with positive volume $\mathrm{vol}(C_t) > 0$ that is revealed after choosing $w_t : C \mapsto \mathbb{R}_{\geqslant 0}$. For each time t define the discretization

$$\mathcal{D}_t = \{D = \bigcap_{s \leqslant t} C_s^{(\mathbf{c}_{[s]})} : \mathbf{c} \in \{0,1\}^t, \mathrm{vol}(D) > 0\} \tag{2.26}$$

of C, where $C_t^{(0)} = C_t$ and $C_t^{(1)} = C \backslash C_t$. We will use elements of these discretizations to index nonnegative vectors in $\mathbb{R}_{\geqslant 0}^{|\mathcal{D}_t|}$; specifically, for any measure $w : C \mapsto \mathbb{R}_{\geqslant 0}$ let $\mathbf{w}(t) \in \mathbb{R}_{\geqslant 0}^{|\mathcal{D}_t|}$ denote the vector with entries $\mathbf{w}(t)_{[D]} = \int_D w(\boldsymbol{\rho}) d\boldsymbol{\rho}$ for $D \in \mathcal{D}_t$. Note that we will exclusively use p, q, v, w for measures, with v specifically referring to the uniform measure, i.e. $\mathbf{v}(t)_{[D]} = \mathrm{vol}(D)$. For convenience, for all real vectors $\mathbf{x}$ we will use $\hat{\mathbf{x}}$ to denote $\mathbf{p}/\|\mathbf{p}\|_1$. Finally, we abuse notation and remove the parentheses to refer those vectors associated with the final discretization, i.e. $\mathbf{v} = \mathbf{v}(T)$ and $\mathbf{w} = \mathbf{w}(T)$.

Now that we have this notation we can turn back to the functions we are interested in: $f_t(w) = -\log \frac{\int_{C_t} w(\boldsymbol{\rho}) d\boldsymbol{\rho}}{\int_C w(\boldsymbol{\rho}) d\boldsymbol{\rho}}$, where $C_t = \mathbb{B}(\boldsymbol{\rho}_t^*, m^{-\beta})$. Observe that we can equivalently write this as $f_t(\mathbf{w}) = -\log\langle \mathbf{w}_t^*, \hat{\mathbf{w}} \rangle$, where $\mathbf{w}_{t[D]}^* = 1_{D \subset C_t}$; this translates our online learning problem from the domain of measures on C to the simplex on $|\mathcal{D}_T|$ elements. However, we cannot play in this domain explicitly as we do not have access to the final discretization $\mathcal{D}_T$, nor do we get access to $\mathbf{w}_t^*$ after task t, except implicitly via C_t. In this section we design a method that implicitly run an online convex optimization procedure over $\mathbb{R}_{\geqslant 0}^{|\mathcal{D}_T|}$ while explicitly playing probability measures $w : C \mapsto \mathbb{R}_{\geqslant 0}$.

As the functions f_t are exp-concave, one might first consider applying a method attaining logarithmic regret on such losses [Hazan et al., 2007, Orabona et al., 2012]; however, such algorithms have regret that depends linearly on the dimension, which in our case is $\mathrm{poly}(T)$. We thus turn to the the follow-the-regularized-leader (FTRL) family of algorithms, which in the case of entropic regularization are well-known to have regret logarithmic in the dimension [Shalev-Shwartz, 2011]. In Algorithm 4 we display the pseudo-code of a modification with regularizer

Algorithm 4: Follow-the-Regularized-Leader (prescient form)

Input: discretization $\mathcal{D}_T$ of C, mixture parameter $\gamma \in [0,1]$, step-size $\eta > 0$
Initialize $\mathbf{w}_1 = \hat{\mathbf{v}}$ **for** $t = 1, 2, \ldots, T$ **do**
  Play $\mathbf{w}_t$
  Suffer $f_t(\mathbf{w}_t) = -\log\langle \mathbf{w}_t^*, \mathbf{w}_t \rangle$
  Observe f_t
  Update $\mathbf{w}_{t+1} = \arg\min_{\|\mathbf{w}\|_1 = 1, \mathbf{w} \geqslant \gamma\hat{\mathbf{v}}} D_{\mathrm{KL}}(\mathbf{w}||\hat{\mathbf{v}}) + \eta \sum_{s \leqslant t} f_s(\mathbf{w})$

$D_{\mathrm{KL}}(\cdot||\hat{\mathbf{v}})$, where recall $\mathbf{v}$ is the vector of volumes of the discretization $\mathcal{D}_T$ of C, and we constrain the played distribution to have measure at least $\gamma\hat{\mathbf{v}}_{[D]}$ over every set $D \in \mathcal{D}_T$.

While Algorithm 4 explicitly requires knowing the discretization $\mathcal{D}_T$ of C in advance, the following key lemma shows that we can run the procedure knowing only the discretization $\mathcal{D}_t$ after task t by simply minimizing the same objective over probability distributions discretized on $\mathcal{D}_t$. This crucially depends on the re-scaling of the entropic regularizer by $\hat{\mathbf{v}}$ (which notably corresponds to the uniform distribution over C) and the fact that $\mathbf{w}_t^* \in \{0,1\}^{|\mathcal{D}_T|}$.

Lemma 2.3.1. Let $w : C \mapsto \mathbb{R}_{\geqslant 0}$ be the probability measure corresponding to the minimizer

$$\mathbf{w} = \operatorname*{arg\,min}_{\|\mathbf{q}\|_1 = 1, \mathbf{q} \geqslant \gamma\hat{\mathbf{v}}} D_{\mathrm{KL}}(\mathbf{q}||\hat{\mathbf{v}}) - \eta \sum_{s \leqslant t} \log\langle \mathbf{w}_s^*, \mathbf{q} \rangle \tag{2.27}$$

and let $\tilde{w} : C \mapsto \mathbb{R}_{\geqslant 0}$ be the probability measure corresponding to the minimizer

$$\tilde{\mathbf{w}}(t) = \operatorname*{arg\,min}_{\|\mathbf{q}\|_1 = 1, \mathbf{q} \geqslant \gamma\hat{\mathbf{v}}(t)} D_{\mathrm{KL}}(\mathbf{q}||\hat{\mathbf{v}}(\mathbf{t})) - \eta \sum_{s \leqslant t} \log\langle \mathbf{w}_s^*(t), \mathbf{q} \rangle \tag{2.28}$$

Then $\mathbf{w} = \tilde{\mathbf{w}}$.

We can thus move on to proving a regret guarantee for Algorithm 4. This follows from Jensen's inequality together with standard results for FTRL once we show that the loss functions are $\frac{1}{\gamma \operatorname{vol}(C_t)}$-Lipschitz over the constrained domain, yielding the following guarantee for Algorithm 4:

Theorem 2.3.4. Algorithm 4 has regret bounded by

$$\frac{1-\gamma}{\eta} D_{\mathrm{KL}}(\mathbf{w}^*||\hat{\mathbf{v}}) + \frac{\eta}{\gamma^2} \sum_{t=1}^{T} \frac{1}{(\operatorname{vol}(C_t))^2} + \gamma \sum_{t=1}^{T} \log \frac{1}{\operatorname{vol}(C_t)} \tag{2.29}$$

w.r.t. the optimum in hindsight $\mathbf{w}^* \in \arg\min_{\|\mathbf{w}\|_1 = 1, \mathbf{w} \geqslant \mathbf{0}} \sum_{t=1}^{T} f_t(\mathbf{w})$ of the functions f_t. Setting $\gamma^2 = GB/\sqrt{T}$ and $\eta^2 = \frac{B^2\gamma^2}{TG^2}$, where $B^2 = D_{\mathrm{KL}}(\mathbf{w}^*||\hat{\mathbf{v}})$ and $G^2 = \frac{1}{T}\sum_{t=1}^{T} \frac{1}{(\operatorname{vol}(C_t))^2}$, yields sublinear regret $\tilde{\mathcal{O}}(\sqrt{BG}T^{\frac{3}{4}})$.

Proof. Algorithm 4 is standard FTRL with regularizer $\frac{1}{\eta} D_{\mathrm{KL}}(\cdot||\hat{\mathbf{v}})$, which has the same Hessian as the standard entropic regularizer over the simplex and is thus $\frac{1}{\eta}$-strongly-convex w.r.t. $\|\cdot\|_1$ [Shalev-Shwartz, 2011, Example 2.5]. Applying Jensen's inequality, the standard regret

bound for FTRL [Shalev-Shwartz, 2011, Theorem 2.11] together with the Lipschitz guarantee of Claim 2.A.9, and Jensen's inequality again yields the result:

$$\begin{aligned}
\sum_{t=1}^{T} f_t(\mathbf{w}_t) - f_t(\mathbf{w}^*) &= \sum_{t=1}^{T} f_t(\mathbf{w}_t) - (1-\gamma) f_t(\mathbf{w}^*) - \gamma f_t(\hat{\mathbf{v}}) + \gamma(f_t(\hat{\mathbf{v}}) - f_t(\mathbf{w}^*)) \\
&\leqslant \sum_{t=1}^{T} f_t(\mathbf{w}_t) - f_t(\gamma\hat{\mathbf{v}} + (1-\gamma)\mathbf{w}^*) + \gamma \log \frac{\langle \mathbf{w}_t^*, \mathbf{w}^* \rangle}{\langle \mathbf{w}_t^*, \hat{\mathbf{v}} \rangle} \\
&\leqslant \frac{1}{\eta} D_{\text{KL}}(\gamma\hat{\mathbf{v}} + (1-\gamma)\mathbf{w}^* || \hat{\mathbf{v}}) + \frac{\eta}{\gamma^2} \sum_{t=1}^{T} \frac{1}{(\text{vol}(C_t))^2} + \gamma \sum_{t=1}^{T} \log \frac{1}{\text{vol}(C_t)} \\
&\leqslant \frac{1-\gamma}{\eta} D_{\text{KL}}(\mathbf{w}^* || \hat{\mathbf{v}}) + \frac{\eta}{\gamma^2} \sum_{t=1}^{T} \frac{1}{(\text{vol}(C_t))^2} + \gamma \sum_{t=1}^{T} \log \frac{1}{\text{vol}(C_t)}
\end{aligned} \tag{2.30}$$

□

Since the regret is sublinear in T, this result satisfies our requirement for attaining asymptotic improvement over single-task learning via Theorem 2.3.3. However, there are several aspects of this bound that warrant some discussion. The first is the rate of $\mathcal{O}(T^{\frac{3}{4}})$, which is less sublinear than the standard $\mathcal{O}(\sqrt{T})$ and certainly the $\mathcal{O}(\log T)$ regret of exp-concave functions. However, the functions we face are (a) non-Lipschitz and (b) over a domain that has dimensionality $\Omega(T)$; both violate conditions for good rates in online convex optimization [Hazan et al., 2007, Shalev-Shwartz, 2011], making our problem much more difficult.

A more salient aspect is the dependence on $B^2 = D_{\text{KL}}(\mathbf{w}^* || \hat{\mathbf{v}})$, effectively the negative entropy of the optimal initialization. This quantity is in-principle unbounded but is analogous to standard online convex optimization bounds that depend on the norm of the optimum, which in e.g. the Euclidean case are also unbounded. In our case, if the optimal distribution is highly concentrated on a very small subset of the space it will be difficult to compete with. Note that our setting of η depends on knowing or guessing B; this is also standard but is certainly a target for future work to address. For example, past work on parameter-free algorithms has solutions for optimization over the simplex [Orabona and Pal, 2016]; however, it is unclear whether this is straightforward to do while preserving the property given by Lemma 2.3.1 allowing us to implicitly work with an unknown discretization. A more reasonable approach may be to compete only with smooth measures that only assign probability at most $\kappa \,\text{vol}(D)$ to any subset $D \subset C$ for some constant $\kappa \geqslant 1$; in this case we will simply have B bounded by $\log \kappa$.

A final issue is the dependence on $\sqrt{G}$, which is bounded by the reciprocal of the smallest volume $\text{vol}(C_t)$, which in the dispersed case is roughly $\mathcal{O}(m^{\beta d})$; this means that the task-averaged regret will have a term that, while decreasing as we see additional tasks, is *increasing* in the number of within-task iterations and the dispersion parameter, which is counter-intuitive. It is also does so exponentially in the dimension. Note that in the common algorithm configuration setting of $\beta = 1/2$ and $d = 1$ this will simply mean that for each task we suffer an extra $o_T(1)$ loss at each within-task round, a quantity which vanishes asymptotically.

Meta-learning the step-size

In addition to learning the initialization, Theorem 2.3.3 requires learning the task similarity to set the within-task step-size $\lambda > 0$. This involves optimizing functions of form $h_t(v) = v + f_t(w_t)/v$. Since we know that the measures w_t are lower-bounded in terms of γ, we can use our previous approach that solves this by running the EWOO algorithm [Hazan et al., 2007] on the modified sequence $v + \frac{f_t(w_t)+\varepsilon^2}{v}$:

Corollary 2.3.1. For any $\varepsilon > 0$, running the EWOO algorithm on the modified sequence $v + \frac{f_t(w)+\varepsilon^2}{v}$ over the domain $[\varepsilon, \sqrt{D^2 - \log\gamma + \varepsilon^2}]$, where $D^2 \geqslant \frac{1}{T}\sum_{t=1}^{T} \log\frac{1}{\mathrm{vol}(C_t)}$, attains regret

$$\min\left\{\frac{\varepsilon^2}{v^*}, \varepsilon\right\} T + \frac{\sqrt{D^2 - \log\gamma}}{2} \max\left\{\frac{D^2 - \log\gamma}{\varepsilon^2}, 1\right\} (1 + \log(T+1)) \tag{2.31}$$

on the original sequence $h_t(v) = v + f_t(w)/v$ for all $v^* > 0$.

Setting $\varepsilon = 1/\sqrt[4]{T}$ gives a guarantee of form $\tilde{\mathcal{O}}((\min\{1/v^*, \sqrt[4]{T}\})\sqrt{T})$. Note this rate might be improvable by using the fact that v is lower-bounded due to the γ-constraint; however, we do not focus on this since this component is not the dominant term in the regret. In fact, because of this we can adapt our approach from Section 2.2.1 that simply runs follow-the-leader (FTL) on the same modified sequence without affecting the dominant terms in the regret:

Corollary 2.3.2. For any $\varepsilon > 0$, running the FTL algorithm on the modified sequence $v + \frac{f_t(w)+\varepsilon^2}{v}$ over the domain $[\varepsilon, \sqrt{D^2 - \log\gamma + \varepsilon^2}]$, where $D^2 \geqslant \frac{1}{T}\sum_{t=1}^{T} \log\frac{1}{\mathrm{vol}(C_t)}$, attains regret

$$\min\left\{\frac{\varepsilon^2}{v^*}, \varepsilon\right\} T + 2\sqrt{D^2 - \log\gamma} \max\left\{\frac{(D^2 - \log\gamma)^{\frac{3}{2}}}{\varepsilon^3}, 1\right\} (1 + \log(T+1)) \tag{2.32}$$

on the original sequence $h_t(v) = v + f_t(w)/v$ for all $v^* > 0$.

Setting $\varepsilon = 1/\sqrt[5]{T}$ gives a guarantee of form $\tilde{\mathcal{O}}((\min\{1/v^*, \sqrt[5]{T}\})T^{\frac{3}{5}})$. The alternatives are described in pseudocode at the bottom of Algorithm 5; while the guarantee of the FTL-based approach is worse, it is almost as simple to compute as the task similarity and does not require integration, making it easier to implement.

Putting the two together

Now that we have an algorithm for both the initialization and the step-size, we can combine the two in Algorithm 5 to meta-learn the parameter of the exponential forecaster. Then we can obtain a bound on the task-averaged regret from Theorem 2.3.3 to attain our final result.

Theorem 2.3.5. Define $B^2 = D_{\mathrm{KL}}(\mathbf{w}^*||\hat{\mathbf{v}})$, $G^2 = \frac{1}{T}\sum_{t=1}^{T} \frac{1}{(\mathrm{vol}(C_t))^2}$, $D^2 \geqslant \frac{1}{T}\sum_{t=1}^{T} \log\frac{1}{\mathrm{vol}(C_t)} = O(\beta d \log m)$, and the task similarity V as in (2.23). Then Algorithm 5 with η, γ set as in Theorem 2.3.4 and $\varepsilon = 1/\sqrt[4]{T}$ (if using EWOO) or $1/\sqrt[5]{T}$ (otherwise) yields task-averaged regret

$$\tilde{\mathcal{O}}\left(\min\left\{\frac{\sqrt{BG}}{V\sqrt[4]{T}}, \frac{\sqrt[4]{BG}}{\sqrt[8]{T}}\right\} + 2V\right)\sqrt{m} + g(m) \tag{2.33}$$

Algorithm 5: Meta-learning the parameters of the exponential forecaster (Algorithm 3). Recall that $\mathbf{p}(t)$ refers to the time-t discretization of the measure $p : C \mapsto \mathbb{R}_{\geqslant 0}$ (c.f. Section 2.3.3).

Input: domain $C \subset \mathbb{R}^d$, dispersion $\beta > 0$, step-size $\eta > 0$, constraint parameter $\gamma \in [0, 1]$, offset parameter $\varepsilon > 0$, domain parameter $D > 0$

Initialize w_1 to the uniform measure on C and set $\lambda_1 = \frac{\varepsilon + \sqrt{D^2 + \varepsilon^2 - \log \gamma}}{2\sqrt{m}}$

for *task* $t = 1, 2, \ldots, T$ **do**

Run Algorithm 3 with initialization w_t and step-size λ_t; get task t optimum $\boldsymbol{\rho}_t^* \in C$

Set $w_t^* = 1_{\mathbb{B}(\boldsymbol{\rho}_t^*, m^{-\beta})}$ to be 1 in an $m^{-\beta}$-ball at $\boldsymbol{\rho}_t^*$ and 0 elsewhere

Set w_{t+1} to $\mathbf{w}_{t+1}(t) = \arg\min_{\|\mathbf{w}\|_1 = 1, \mathbf{w} \geqslant \gamma \hat{\mathbf{v}}(t)} D_{\mathrm{KL}}(\mathbf{w} || \hat{\mathbf{v}}(t)) - \eta \sum_{s \leqslant t} \log \langle \mathbf{w}_s^*(t), \mathbf{w} \rangle$

if *using EWOO* **then**

Define $\mu_t(x) = \exp\left(-\alpha\left(tx + \frac{t\varepsilon^2 - \sum_{s \leqslant t} \log \langle \mathbf{w}_s^*(s), \mathbf{w}_s(s) \rangle}{x}\right)\right)$ for $\alpha = \frac{2}{D} \min\left\{\frac{\varepsilon^2}{D^2}, 1\right\}$

Set $\lambda_{t+1} = \frac{\int_\varepsilon^{\sqrt{D^2 + \varepsilon^2 - \log \gamma}} x \mu_t(x) dx}{\sqrt{m} \int_\varepsilon^{\sqrt{D^2 + \varepsilon^2 - \log \gamma}} \mu_t(x) dx}$

else

Set $\lambda_{t+1} = \sqrt{\frac{\sum_{s \leqslant t} \varepsilon^2 - \log \langle \mathbf{w}_s^*(s), \mathbf{w}_s(s) \rangle}{tm}}$

So as in the previous section, this achieves the meta-learning goal of adapting to the task similarity by attaining asymptotic regret of $2V\sqrt{m} + \mathcal{O}(m^{-\beta})$ on-average, where here we substitute the dispersion term for g and V^2 is the task similarity encoding the average probability mass assigned to the different task balls by the optimal initialization distribution. We include the minimum of two rates in the bound, with the rate being $\mathcal{O}(1/\sqrt[4]{T})$ is the task similarity is a constant $\Theta_T(1)$ and $\mathcal{O}(1/\sqrt[8]{T})$ if it is extremely small. As discussed in above, this rate reflects the difficulty of our meta-problem, in which we are optimizing nonsmooth functions over a space of distributions; in contrast, our meta-update procedures in the previous section took advantage of nice properties of Bregman divergences to obtain faster rates.

2.3.4 Meta-learning for data-driven algorithm design

We demonstrate the utility of our bounds in a series of applications in data-driven algorithm design, demonstrating how our results imply guarantees for meta-learning the tuning of solvers for several difficult combinatorial problems arising from the theory of computing. We also demonstrate the practical utility of our approach for tuning clustering algorithms on real and synthetic datasets.

Instantiations for tuning combinatorial optimization algorithms

Algorithm configuration for combinatorial optimization algorithms involves learning algorithm parameters from multiple instances of combinatorial problems [Gupta and Roughgarden, 2017,

Balcan et al., 2018c]. For problems like maximum weighted independent set (MWIS), integer quadratic programming (IQP), and auction design, the an algorithm's performance on a fixed instance is typically a piecewise-Lipschitz function of its parameters. Prior work has looked at learning these parameters in the distributional setting (i.e. assuming i.i.d. draws of problem instances) [Gupta and Roughgarden, 2017, Balcan et al., 2018b] or the online setting where the problem instances may be adversarially drawn [Balcan et al., 2018b, 2020b]. On the other hand, instantiating our results for these problems provide upper bounds for much more realistic settings where different tasks may be related and our bounds improve with this relatedness. We demonstrate how to apply our results to several combinatorial problems under mild smoothness assumptions. The key idea is to show that if the inputs come from a smooth distribution, the algorithmic performance is dispersed (as a sequence of functions in the algorithm parameters).

We start with the **MWIS** problem, where there is a graph $G = (V, E)$ and a weight $w_v \in \mathbb{R}_{>0}$ for each vertex $v \in V$. The goal is to find a set of non-adjacent vertices with maximum total weight. The problem is NP-hard and in fact does not have any constant factor polynomial time approximation algorithm. Gupta and Roughgarden [2017] propose a greedy heuristic family, which selects vertices greedily based on largest value of $w_v/(1 + \deg(v))^\rho$, where $\deg(v)$ is the degree of vertex v, and removes neighbors of the selected vertex before selecting the next vertex.

For this algorithm family, we can learn the best parameter ρ provided pairs of vertex weights have a joint κ-bounded distribution, and Theorem 2.3.5 implies regret bounds that improve with task similarity. We use the recipe from Balcan et al. [2020a] to establish dispersion:

Theorem 2.3.6. Consider instances of MWIS with all vertex weights in $(0, 1]$ and for each instance, every pair of vertex weights has a κ-bounded joint distribution. Then the asymptotic task-averaged regret for learning the algorithm parameter ρ is $o_T(1) + 2V\sqrt{m} + \mathcal{O}(\sqrt{m})$.

Proof sketch. The loss function is piecewise constant with discontinuities corresponding to ρ such that $w_v/(1 + \deg(v))^\rho = w_u/(1 + \deg(u))^\rho$ for a pair of vertices u, v. Balcan et al. [2018b] show that the discontinuities have $(\kappa \ln n)$-bounded distributions where n is the number of vertices. This implies that in any interval of length ϵ, we have in expectation at most $\epsilon\kappa \ln n$ discontinuities. Using this in dispersion recipe from Balcan et al. [2020a] implies $\frac{1}{2}$-dispersion, which in turn implies the desired regret bound by applying Theorem 2.3.5. □

Similar results may be obtained for other combinatorial problems including knapsack, k-center clustering, IQP and auction design (c.f. Appendix 2.A.5 for full details). We further show instantiations of our results for knapsack and k-center clustering, for which we will empirically validate our proposed methods in the next sections. The first of these, **knapsack**, is a well-known NP-complete problem. We are given a knapsack with capacity $\texttt{cap}$ and items $i \in [m]$ with sizes w_i and values v_i. The goal is to select a subset S of items to add to the knapsack such that $\sum_{i \in S} w_i \leqslant \texttt{cap}$ while maximizing the total value $\sum_{i \in S} v_i$ of selected items. The classic greedy heuristic to add items in decreasing order of v_i/w_i gives a 2-approximation. We consider a generalization to use v_i/w_i^ρ proposed by Gupta and Roughgarden [2017] for $\rho \in [0, 10]$. For example, for the value-weight pairs $\{(0.99, 1), (0.99, 1), (1.01, 1.01)\}$ and capacity $\texttt{cap} = 2$ the classic heuristic $\rho = 1$ gives value 1.01 but using $\rho = 3$ gives the optimal value 1.98. We can learn this optimal value of ρ from similar tasks, and obtain formal guarantees similar to Theorem 2.3.6 (proof in Appendix 2.A.5).

Theorem 2.3.7. Consider instances of the knapsack problem given by bounded weights $w_{i,j} \in [1, C]$ and κ-bounded independent values $v_{i,j} \in [0, 1]$ for $i \in [m], j \in [T]$. Then the asymptotic task-averaged regret for learning the algorithm parameter ρ for the greedy heuristic family described above is $o_T(1) + 2V\sqrt{m} + \mathcal{O}(\sqrt{m})$.

Lastly, for k**-center clustering** we consider the parameterized α-Llyod's algorithm family introduced in Balcan et al. [2018c]. In the seeding phase, each point x is sampled with probability proportional to $\min_{c \in C} d(v, c)^\alpha$, where $d(\cdot, \cdot)$ is the distance metric and C is the set of centers chosen so far. The family contains an algorithm for each $\alpha \in [0, \infty) \cup \infty$, and includes popular clustering heuristics like vanilla k-means (random initial centers, for $\alpha = 0$), k-means++ (corresponding to $\alpha = 2$) and farthest-first traversal ($\alpha = \infty$). The performance of the algorithm is measured using the Hamming distance to the optimal clustering, and is a piecewise constant function of α. Our meta-learning result can be instantiated for this problem even without smoothness assumptions by leveraging the smoothness induced by the internal randomness of the clustering algorithm (proof in Appendix 2.A.5).

Theorem 2.3.8. Consider instances of the k-center clustering problem on n points, with Hamming loss $l_{i,j}$ for $i \in [m], j \in [T]$ against some (unknown) ground truth clustering. Then the asymptotic task-averaged regret for learning the algorithm parameter α for the α-Lloyd's clustering algorithm family of Balcan et al. [2018c] is $o_T(1) + 2V\sqrt{m} + \mathcal{O}(\sqrt{m})$.

In the following section we look at applications of our results through experiments for the knapsack and k-center clustering problems.

Experiments for greedy knapsack and k-center clustering

We design experiments to evaluate our new meta-initialization algorithm for data-driven design for knapsack and clustering problems on real and simulated data. Our experiments show the usefulness of our techniques in learning a sequence of piecewise-Lipschitz functions.

For our experiments, we generate a synthetic dataset of knapsack instances described as follows. For each problem instance of each task, we have $\texttt{cap} = 100$ and $m = 50$. We have 10 'heavy' items with $w_i \sim \mathcal{N}(27, 0.5)$ and $v_i \sim \mathcal{N}(27, 0.5)$, and 40 items with $w_i \sim \mathcal{N}(19 + w_t, 0.5)$ and $v_i \sim \mathcal{N}(18, 0.5)$, where $w_t \in [0, 2]$ is task-dependent.

We also consider the parameterized α-Lloyd's algorithm family from Balcan et al. [2018c]. The performance of the algorithm is measured using the Hamming loss relative to the optimal clustering, and is a piecewise constant function of α. We can compute the pieces of this function for $\alpha \in [0, 10]$ by iteratively computing the subset of parameter values where a candidate point can be the next center. We use the small split of the *Omniglot* dataset [Lake et al., 2017], and create clustering tasks by drawing random samples consisting of five characters each, where four characters are constant throughout. We also create a Gaussian mixture binary classification dataset, with each class a 2D Gaussian distribution consisting of 100 points each, with variance $\operatorname{diag}\left(\begin{pmatrix}\sigma^2 & 4\sigma^2\end{pmatrix}\right)$ and centers $\mathbf{0}_2$ and $d\sigma\mathbf{e}_1$. We pick $d \in [2, 3]$ to create different tasks.

For each dataset we learn using 30 instances each of 10 training tasks and evaluate average loss over 5 test tasks. We perform 100 iterations to average over the randomization of the clustering algorithm and the exponential forecaster algorithm. We perform meta-initialization

Table 2.2: Effect of meta-initialization on few-shot learning of algorithmic parameters. Performance is computed as a fraction of the average value (Hamming accuracy, or knapsack value) of the offline optimum parameter.

Dataset	Omniglot		Gaussian Mixture		Knapsack	
	One-shot	Five-shot	One-shot	Five-shot	One-shot	Five-shot
Single task	$88.67 \pm 0.47\%$	$95.02 \pm 0.19\%$	$90.10 \pm 1.10\%$	$91.43 \pm 0.44\%$	$84.74 \pm 0.29\%$	$98.89 \pm 0.17\%$
Meta-initialized	$89.65 \pm 0.49\%$	$96.05 \pm 0.15\%$	$95.76 \pm 0.60\%$	$96.39 \pm 0.27\%$	$85.66 \pm 0.57\%$	$99.12 \pm 0.15\%$

with parameters $\gamma = \eta = 0.01$ (no hyperparameter search performed). The step-size is set to minimize the regret term in Theorem 2.3.1, and not meta-learned.

The relative improvement in task-averaged regret due to meta-learning in our formal guarantees depend on the task similarity V and how it compares to the dispersion-related $\mathcal{O}(m^{1-\beta})$ term, and can be significant when the latter is small. Our results in Table 2.2 show that meta-learning an initialization, i.e. a distribution over the algorithm parameter, for the exponential forecaster in this setting yields improved performance on each dataset. We observe this for both the one-shot and five-shot settings, i.e. the number of within-task iterations of the test task are one and five respectively. The benefit of meta-learning is most pronounced for the Gaussian mixture case (well-dispersed and similar tasks), and gains for Omniglot may increase with more tasks (dispersed but less similar tasks). For our knapsack dataset, the relative gains are smaller (similar tasks, but less dispersed). See Appendix 2.B.2 for further details.

2.3.5 Conclusion

In this section we extended our approach for analyzing initialization-based meta-learning to the online learning of piecewise-Lipschitz functions, demonstrating how online convex optimization over an adaptive discretization can find an initialization that improves the performance of the exponential forecaster across tasks, assuming the tasks have related optima. We then applied this result to data-driven algorithm design, such as the online configuration of clustering algorithms. Our results demonstrate that ARUBA can be applied even in the face of nonconvex losses; in the next section, we will further show its extension to problems with partial information.

2.4 Meta-learning adversarial bandit algorithms

Thus far we have used ARUBA to understood meta-learning of online learning algorithms in the *full-information* setting, where the loss for every arm is revealed after each round. This assumption is not realistic in many applications, e.g. recommender systems and experimental design, where often partial or *bandit* feedback—only the loss of the action taken—is revealed. Such feedback can be *stochastic*, e.g. the losses are i.i.d. from some distribution, or *adversarial*, i.e. chosen by an adversary. In this section we establish formal guarantees for online-within-online meta-learning with adversarial bandit feedback. As with past full-information meta-learning results, our goal when faced with a sequence of bandit tasks will be to achieve low regret *on average* across them. Specifically, our task-averaged regret should (a) be no worse than that of algorithms

for the single-task setting, e.g. if the tasks are not very similar, and should (b) be much better on tasks that are closely related, e.g. if the same small set of arms do well on all of them. We show that a natural way to achieve both is to initialize and tune online mirror descent (OMD), an algorithm associted with a strictly convex regularizer whose hyperparameters have a significant impact on performance. Our approach works because it can learn the best hyperparameters in hindsight across tasks, which will recover OMD's worst-case optimal performance if the tasks are dissimilar but will take advantage of more optimistic settings if they are related. As generalized distances, the regularizers also induce interpretable measures of similarity between tasks.

To show these results, we extend our ARUBA-based analysis of meta-learning online mirror descent from Section 2.2, which involved online learning of sequences of Bregman divergences. Our core structural result shows that the regularizers ψ_θ of these divergences can be tuned without interfering with meta-learning the initialization and step-size; tuning θ is critical for adapting to settings such as that of a small set of optimal arms in MAB. Doing so depends on several refinements of the original approach, including bounding the task-averaged-regret via the spectral norm of $\nabla^2\psi_\theta$ and expressing the loss of the meta-comparator using only ψ_θ, rather than via its Bregman divergence as in prior work. Applying the structural result also requires setting-specific analysis, e.g. to show regularity w.r.t. θ or to obtain MAB guarantees in terms of the entropy of the true optimal arms. The latter is especially difficult, previous we defined task similarity via full information upper bounds, and involves applying tools from the best-arm-identification literature [Abbasi-Yadkori et al., 2018] to show that a constrained variant of Exp3 finds the optimal arm w.h.p.

Overview of bandit results

We design a meta-algorithm (Algorithm 6) for learning variants of OMD—specifically those with entropic or self-concordant regularizers—that are used for adversarial bandits. This meta-algorithm combines three *full-information* algorithms—follow-the-leader (FTL), exponentially weighted online optimization (EWOO), and multiplicative weights (MW)—to set the initialization, step-size, and regularizer-specific parameters, respectively. It works by optimizing a sequence of functions that each *upper-bound* the regret of OMD on a single task (Theorem 2.4.1), resulting in (a) interesting notions of task similarity because these functions depend on generalized notions of distances (Bregman divergences) and (b) adaptivity, i.e not needing to know how similar the tasks are beforehand.

Our first application is to OMD with the Tsallis regularizer [Abernethy et al., 2015], a relative of Exp3 [Auer et al., 2002] that is optimal for adversarial MAB. We bound the task-averaged regret by the Tsallis entropy of the *estimated* optima-in-hindsight (Corollary 2.4.1), which we further extend to that of the *true* optima by assuming a gap between the best and second-best arms (Corollary 2.4.2). Both results are consequences of Corollary 2.A.1, where we showed the online-learnability of sequences of Bregman divergences, even ones that are *nonconvex* in their second (learned) arguments, which is the case here due to the Tsallis regularizer. As an example, our bound on the m-round regret across T tasks under the gap assumption is

$$o_T(\text{poly}(m)) + 2\min_{\beta\in(0,1]}\sqrt{H_\beta d^\beta m/\beta} + o(\sqrt{m}) \tag{2.34}$$

where d is the number of actions and H_β is the Tsallis entropy [Tsallis, 1988, Abernethy et al., 2015] of the distribution of the optimal actions ($\beta = 1$ recovers the Shannon entropy). This entropy is low if all tasks are usually solved by the same few arms, making it a natural task similarity notion. For example, if only $s \ll d$ of the arms are ever optimal then $H_\beta = \mathcal{O}(s)$, so using $\beta = 1/\log d$ in (2.34) yields an asymptotic task-averaged regret of $\mathcal{O}(\sqrt{sm \log d})$, dropping fast terms. For $s = \mathcal{O}_d(1)$ this beats the minimax optimal rate of $\Theta(\sqrt{dm})$ [Audibert et al., 2011]. On the other hand, since $H_{1/2} = \mathcal{O}(\sqrt{d})$, the same bound recovers this rate in the worst-case of dissimilar tasks.

Lastly, we adapt our meta-algorithm to the adversarial BLO problem by setting the regularizer to be a self-concordant barrier function, as in Abernethy et al. [2008b]. Our bounds yield notions of task similarity that depend on the constraints of the action space, e.g. over the sphere the measure is the closeness of the average of the estimated optima to the sphere's surface (Corollary 2.4.4). We also instantiate BLO on the bandit shortest-path problem (Corollary 2.4.5) [Takimoto and Warmuth, 2003, Kalai and Vempala, 2005].

Related work

While we study the adversarial setting, meta-learning has been analyzed in various *stochastic* bandit settings [Azar et al., 2013, Kveton et al., 2020, Sharaf and Daumé III, 2021, Simchowitz et al., 2021, Kveton et al., 2021, Basu et al., 2021, Cella et al., 2020, Moradipari et al., 2022, Azizi et al., 2022]. The latter three study stochastic bandits under various task generation assumptions, e.g. Azizi et al. [2022] is in a batch-within-online setting where the optimal arms are adversarial. In contrast, we make no distributional assumptions either within or without. Apart from this difference, the results of Azizi et al. [2022] are the ones our MAB results are most easily compared to, which we do in detail in Section 2.4.2. Notably, they assume that only $s \ll d$ of the d arms are ever optimal across T tasks and show (roughly speaking) $\tilde{\mathcal{O}}(\sqrt{sm})$ asymptotic regret; we instead focus on an entropic notion of task similarity that achieves the same asymptotic regret when specialized to their $s \ll d$. However, avoiding their explicit assumption has certain advantages, e.g. robustness in the presence of $o(T)$ outlier tasks (c.f. Section 2.4.2).

A setting that bears some similarity to online-within-online bandits is that of switching bandits [Auer et al., 2002], and more generally online learning with dynamic comparators [Anava and Karnin, 2016, Jadbabaie et al., 2015, Luo et al., 2018, Auer et al., 2019, Zhao et al., 2021]. In such problems, instead of using a static best arm as the comparator we use a piecewise constant sequence of arms, with a limited number of arm switches. The key difference between such work and ours is our assumption that task boundaries are known; this makes the other setting more general. However, while e.g. Exp3.S [Auer et al., 2002] can indeed be applied to online meta-learning, guarantees derived from switching costs *cannot* improve upon just running Tsallis-INF on each task [Marinov and Zimmert, 2021, Table 1]. Furthermore, these approaches usually quantify difficulty by the number of switches, whereas we focus on task similarity. While there exists stochastic-setting work that measures difficulty using a notion of average change in distribution across rounds [Wei and Luo, 2021], it does not lead to improved performance if this average change is $\Omega(T)$, as is the case in e.g. the s-sparse setting discussed above.

2.4.1 Learning the regularizers of bandit algorithms

We consider the problem of meta-learning over bandit tasks $t = 1, \dots, T$ over some fixed set $\mathcal{K} \subset \mathbb{R}^d$, a (possibly improper) subset of which is the action space $\mathcal{A}$. On each round $i = 1, \dots, m$ of task t we play action $\mathbf{x}_{t,i} \in \mathcal{A}$ and receive feedback $\ell_{t,i}(\mathbf{x}_{t,i})$ for some function $\ell_{t,i} : A \mapsto [-1, 1]$. Note that all functions we consider will be linear and so we will also write $\ell_{t,i}(\mathbf{x}) = \langle \boldsymbol{\ell}_{t,i}, \mathbf{x} \rangle$, i.e. $\boldsymbol{\ell}_{t,i[a]} = \ell_{t,i}(a)$. Additionally, we assume the adversary is *oblivious within-task*, i.e. it chooses losses $\ell_{t,1}, \dots, \ell_{t,m}$ at time t. Finally, note that all proofs can be found in Appendices 2.A.6 through 2.A.10.

Recall that in online learning the goal on a single task t is to play actions $\mathbf{x}_{t,1}, \dots \mathbf{x}_{t,m}$ that minimize regret $\sum_{i=1}^m \ell_{t,i}(\mathbf{x}_{t,i}) - \ell_{t,i}(\mathring{\mathbf{x}}_t)$, where $\mathring{\mathbf{x}}_t \in \arg\min_{\mathbf{x} \in \mathcal{K}} \sum_{i=1}^m \ell_{t,i}(\mathbf{x})$ denotes the optimum on task t.[4] Lifting this to the meta-learning setting, our goal as in the previous sections will be to design an algorithm that uses multi-task data to improve the task-averaged regret $\frac{1}{T} \sum_{t=1}^T \sum_{i=1}^m \ell_{t,i}(\mathbf{x}_{i,t}) - \ell_{t,i}(\mathring{\mathbf{x}}_t)$. For example, we wish to attain a task-averaged regret bound of the form $o_T(\mathrm{poly}(m)) + \tilde{\mathcal{O}}(V\sqrt{m}) + o(\sqrt{m})$, where $V \in \mathbb{R}_{\geqslant 0}$ is a measure of task similarity that is small if the tasks are similar but still yields the worst-case single-task performance—$\mathcal{O}(\sqrt{dm})$ for MAB and $\mathcal{O}(d\sqrt{m})$ for BLO—if they are not.

Regret upper bounds for bandit OMD

Previously in Section 2.2.1 we showed guarantees for meta-learning the initialization and step-size of online mirror descent (OMD), which given a strictly convex regularizer $\psi : \mathcal{K}^\circ \mapsto \mathbb{R}$, step-size $\eta > 0$, and initialization $\mathbf{x}_{t,1} \in \mathcal{K}^\circ$ takes a actions

$$\mathbf{x}_{t,i+1} = \operatorname*{arg\,min}_{\mathbf{x} \in \mathcal{K}^\circ} \mathcal{B}(\mathbf{x} || \mathbf{x}_{t,1}) + \eta \sum_{j \leqslant i} \langle \nabla \ell_{t,j}(\mathbf{x}_{t,j}), \mathbf{x} \rangle \tag{2.35}$$

where $\mathcal{B}(\cdot||\cdot)$ is the Bregman divergence of ψ. Our investigation focused on the case of $\psi(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2$, in which case $\mathcal{B}(\mathbf{x}||\mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|_2^2$ and OMD is just online *gradient* descent (OGD). In this setting we derived a GBML method that—if the tasks are similar according to a task similarity measure induced by this Euclidean Bregman divergence—finds an initialization that performs well after only a few steps on a new task.

However, the OMD family includes many methods beyond OGD, e.g. exponentiated gradient when $\psi(\mathbf{p}) = \langle \mathbf{p}, \log \mathbf{p} \rangle$ is the negative Shannon entropy on probability vectors $\mathbf{p} \in \triangle$ and $\mathcal{B}$ is the KL-divergence [Shalev-Shwartz, 2011]. Most importantly for bandit tasks, OMD variants run on **loss estimators** $\hat{\ell}_{t,i}$ constructed via partial feedback are an important class of bandit methods that achieve state-of-the-art guarantees in various theoretical settings [Auer et al., 2002, Abernethy et al., 2008b, 2015]. As a result, we can adapt our ARUBA framework by again constructing and optimizing sequences $U_t(\mathbf{x}, \eta, \theta)$ of affine functions of Bregman divergences that bound the regret of OMD run with initialization $\mathbf{x}$, step-size η, and a new offset parameter we introduce to handle the non-Lipschitzness of bandit regularizers near the boundaries.

To define these upper bounds, first note that the regret of OMD w.r.t. a comparator $\mathbf{y}$ is bounded by $\mathcal{B}(\mathbf{y}||\mathbf{x})/\eta + \mathcal{O}(\eta m)$ [Shalev-Shwartz, 2011, Hazan, 2015]. In our case the comparator is based on the estimated optimum $\hat{\mathbf{x}}_t \in \arg\min_{\mathbf{x} \in \mathcal{K}} \langle \hat{\boldsymbol{\ell}}_t, \mathbf{x} \rangle$, where $\hat{\boldsymbol{\ell}}_t = \sum_{i=1}^m \hat{\boldsymbol{\ell}}_{t,i}$, resulting

[4] In this section we use $\mathring{\mathbf{x}}_t$ instead of $\mathbf{x}_t^*$ to avoid double superscripts in the corresponding proofs.

from running OMD on task t using initialization $\mathbf{x} \in \mathcal{K}$ and hyperparameters η and θ, which we denote $\texttt{OMD}_{\eta,\theta}(\mathbf{x})$. Unlike full-information meta-learning, we use a parameter $\varepsilon > 0$ to constrain this optimum to lie in a subset $\mathcal{K}_\varepsilon \subset \mathcal{K}^\circ$. Formally, we fix a point $\mathbf{x}_1 \in \mathcal{K}^\circ$ to be the "center"—e.g. $\mathbf{x}_1 = \mathbf{1}_d/d$ when $\mathcal{K}$ is the d-simplex $\triangle$—and define the projection $\mathbf{c}_\varepsilon(\mathbf{x}) = \mathbf{x}_1 + \frac{\mathbf{x}-\mathbf{x}_1}{1+\varepsilon}$ mapping from $\mathcal{K}$ to $\mathcal{K}_\varepsilon$. For example, $\mathbf{c}_{\frac{\varepsilon}{1-\varepsilon}}(\mathbf{x}) = (1-\varepsilon)\mathbf{x} + \varepsilon \mathbf{1}_d/d$ on the simplex. This projection allows us to handle regularizers ψ that diverge near the boundary, but also introduces ε-dependent error terms. In the BLO case it also forces us to tune ε itself, as initializing too close to the boundary leads to unbounded regret while initializing too far away does not take advantage of task similarity. Thus the general upper bounds of interest are the following functions of the initialization $\mathbf{x}$, the step-size $\eta > 0$, and a third parameter θ that is either β or ε, depending on the setting (MAB or BLO):

$$U_t(\mathbf{x}, \eta, \theta) = \frac{\mathcal{B}_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)||\mathbf{x})}{\eta} + \eta g(\theta) m + f(\theta) m \tag{2.36}$$

Here $\mathcal{B}_\theta$ is the Bregman divergence of ψ_θ while $g(\theta) \geqslant 1$ and $f(\theta) \geqslant 0$ are tunable constants. We overload θ to be either β or ε for notational simplicity, as we will not tune them simultaneously; if $\theta = \beta$ (for MAB) then $c_\theta(\mathbf{x}) = \mathbf{x}_1 + \frac{\mathbf{x}-\mathbf{x}_1}{1+\varepsilon}$ for fixed ε, while if $\theta = \varepsilon$ (for BLO) then $\mathcal{B}_\theta$ is the Bregman divergence of a fixed ψ. The reason to optimize this sequence of upper bounds U_t is because they directly bound the task-averaged regret while being no worse than the worst-case single-task regret. Furthermore, an average over Bregman divergences is minimized at the average $\bar{\hat{\mathbf{x}}} = \frac{1}{T}\sum_{t=1}^T \hat{\mathbf{x}}_t$, where it attains the value $\hat{V}_\theta^2 = \frac{1}{T}\sum_{t=1}^T \psi_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)) - \psi_\theta(\mathbf{c}_\theta(\bar{\hat{\mathbf{x}}}))$ (c.f. Claim B.2.2). We will show that this quantity leads to intuitive and interpretable notions of task similarity in all the applications we study.

A meta-algorithm for tuning bandit algorithms

To learn these functions $U_t(\mathbf{x}, \eta, \theta)$—and thus to meta-learn $\texttt{OMD}_{\eta,\theta}(\mathbf{x})$—our meta-algorithm sets $\mathbf{x}$ to be the projection $\mathbf{c}_\theta$ of the mean of the estimated optima—i.e. follow-the-leader (FTL) over the Bregman divergences in (2.36)—while simultaneously setting η via EWOO and θ via discrete multiplicative weights (MW). We choose FTL, EWOO, and MW because each is well-suited to the way U_t depends on $\mathbf{x}$, η, and θ, respectively. First, the only effect of $\mathbf{x}$ on U_t is via the Bregman divergence $\mathcal{B}_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)||\mathbf{x})$, over which FTL attains logarithmic regret (c.f. Corollary 2.A.1). For η, U_t is exp-concave on $\eta > 0$ so long as the first term is nonzero, but it is also non-Lipschitz; the EWOO algorithm is one of the few methods with logarithmic regret on exp-concave losses without a dependence on the Lipschitz constant [Hazan et al., 2007], and we ensure the first term is nonzero by *regularizing* the upper bounds as follows for some $\rho > 0$ and $D_\theta^2 = \max_{\mathbf{x},\mathbf{y} \in \mathcal{K}_\theta} \mathcal{B}_\theta(\mathbf{x}||\mathbf{y})$:

$$U_t^{(\rho)}(\mathbf{x}, \eta, \theta) = \frac{\mathcal{B}_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)||\mathbf{x}) + \rho^2 D_\theta^2}{\eta} + \eta g(\theta) m + f(\theta) m \tag{2.37}$$

This function depends only on $\hat{\mathbf{x}}_t$, obtained by running OMD on task t, and so we can use full-information MW to tune θ across the grid Θ_k. Showing low regret w.r.t. any $\theta \in \Theta \supset \Theta_k$ then just requires sufficiently large k and Lipschitzness of U_t w.r.t. θ. Combining all three algorithms together thus yields the guarantee in Theorem 2.4.1, which is our main structural result. It implies

a generic approach for obtaining meta-learning algorithms by (1) bounding the task-averaged regret by an average of functions of the form U_t, (2) applying the theorem to obtain a new bound $o_T(1) + \min_{\theta,\eta} \frac{\hat{V}_\theta^2}{\eta} + \eta g(\theta)m + f(\theta)m$, and (3) bounding the estimated task similarity $\hat{V}_\theta^2$ by an interpretable quantity. Crucially, since we can choose any $\eta > 0$, the asymptotic regret is always as good as the worst-case guarantee for running the base learner separately on each task.

Theorem 2.4.1 (c.f. Thm. 2.A.13)**.** Suppose $\mathbf{x}_1 = \arg\min_{\mathbf{x}\in\mathcal{K}} \psi_\theta(\mathbf{x}) \; \forall \, \theta$ and let D, M, F, and S be maxima over θ of D_θ, $D_\theta\sqrt{g(\theta)m}$, $f(\theta)$, and $\|\nabla^2\psi_\theta\|_\infty$, respectively. For each $\rho \in (0,1)$ we can set $\underline{\eta}$, $\overline{\eta}$, α, and λ s.t. the expected average of the losses $U_t(\mathbf{c}_{\theta_t}(\mathbf{x}_t), \eta_t(\theta_t), \theta_t)$ of Algorithm 6 is at most

$$\min_{\theta\in\Theta,\eta>0} \frac{\mathbb{E}\hat{V}_\theta^2}{\eta} + \eta g(\theta)m + f(\theta)m + \tilde{\mathcal{O}}\left(\frac{\frac{M}{\rho} + Fm}{\sqrt{T}} + \frac{L_\eta}{k} + \frac{M}{\rho^2 T} + \min\left\{\frac{\rho^2 D^2}{\eta}, \rho M\right\} + \frac{S}{\eta T}\right) \tag{2.38}$$

Here $\hat{V}_\theta^2 = \frac{1}{T}\sum_{t=1}^T \psi_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)) - \psi_\theta(\mathbf{c}_\theta(\hat{\bar{\mathbf{x}}}))$ and L_η bounds the Lipschitz constant w.r.t. θ at $\hat{V}_\theta^2/\eta + \eta g(\theta)m + f(\theta)m$. The same bound plus $(M/\rho + Fm)\sqrt{\frac{1}{T}\log\frac{1}{\delta}}$ holds w.p. $\geqslant 1-\delta$.

Proof sketch. First consider online learning $U_t(\cdot,\cdot,\theta)$ for fixed $\theta \in \Theta_k$. To tune η, we online learn the one-dimensional losses $\mathcal{B}_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)||\mathbf{c}_\theta(\mathbf{x}_t))/\eta + \eta g(\theta)$, where $\mathbf{c}_\theta(\hat{\mathbf{x}}_t)$ is the ($\eta_t(\theta)$-independent) action of FTL at time t. As discussed, the corresponding regularized losses $U_t^{(\rho)}$ are exp-concave, and so running EWOO yields $\tilde{\mathcal{O}}\left(M/\rho^2 + \min\left\{\rho^2 D^2/\eta, \rho M\right\} T\right)$ regret w.r.t. the original sequence. At the same time, we show that FTL has logarithmic regret on the sequence $\mathcal{B}_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)||\cdot)$ that scales with the spectral norm S of $\nabla^2\psi_\theta$ (c.f. Corollary 2.A.1), and that the average loss of the optimal comparator is $\hat{V}_\theta^2$ (c.f. Claim B.2.2). Thus, since we only care about a fixed comparator η, dividing by ηT yields the first and last terms (2.38). We run a copy of these algorithms for each $\theta \in \Theta_k$; since their losses are bounded by $\tilde{\mathcal{O}}(M/\rho + Fm)$, textbook results for MW yield $\mathcal{O}(\sqrt{T\log k})$ regret w.r.t. $\theta \in \Theta_k$, which we then extend to $\Theta \supset \Theta_k$ using L_η-Lipschitzness. □

We keep details of the dependence on S and other constants as they are important in applying this result, but in most cases setting $\rho = \frac{1}{\sqrt[4]{T}}$ yields $\tilde{\mathcal{O}}(T^{\frac{3}{4}})$ regret. While a slow rate, the losses U_t are non-Lipschitz and nonconvex in-general, and learning them allows us to tune θ over user-specified intervals and η over all positive numbers, which will be crucial later. At the same time, this tuning is what leads to the slow rate, as without tuning ($k = 1$, $L_\eta = 0$) the same ρ yields $\tilde{\mathcal{O}}(\sqrt{T})$ regret. Lastly, while we focus on learning guarantees, we note that Algorithm 6 is reasonably efficient, requiring a $2k$ single-dimensional integrals per task; this is discussed in more detail in Section 2.4.1.

Computational and space complexity

Algorithm 6 implicitly maintains a separate copy of FTL for each hyperparameter in the continuous space of EWOO and the grid Θ_k over the domain of θ, but explicitly just needs to average the estimated task optima $\hat{\mathbf{x}}_t$; this is due to the mean-as-minimizer property of Bregman divergences and the linearity of $\mathbf{c}_\varepsilon$. Thus the memory it uses is $\mathcal{O}(d+k)$, where k is size of the discretization of Θ and should be viewed as sublinear in T, e.g. for MAB with implicit exploration and BLO

Algorithm 6: Meta-procedure for tuning $\texttt{OMD}_{\eta,\theta}$ with regularizer $\psi_\theta : \mathcal{K}^\circ \mapsto \mathbb{R}$ and step-size $\eta > 0$. Assume OMD takes as input an initialization in $\mathcal{K}$, is run over loss estimators $\hat{\boldsymbol{\ell}}_{t,1}, \ldots, \hat{\boldsymbol{\ell}}_{t,m}$, and returns estimated task optima $\hat{\mathbf{x}}_t = \arg\min_{\mathbf{x}\in\mathcal{K}} \sum_{i=1}^m \langle \hat{\boldsymbol{\ell}}_{t,i}, \mathbf{x}\rangle$.

Input: compact set $\mathcal{K} \subset \mathbb{R}^d$, initialization $\mathbf{x}_1 \in \mathcal{K}$, ordered subset $\Theta_k \subset \mathbb{R}$ also used to index interval bounds $\underline{\eta}, \overline{\eta} \in \mathbb{R}^k_{\geqslant 0}$ and hyperparameters $\alpha \in \mathbb{R}^k_{\geqslant 0}$, scalar hyperparameters $\rho > 0$ and $\lambda \geqslant 0$, learners $\texttt{OMD}_{\eta,\theta} : \mathcal{K} \mapsto \mathbb{R}^d$, projections $\mathbf{c}_\theta : \mathcal{K} \mapsto \mathcal{K}_\theta$

for $\theta \in \Theta_k$ **do**
- $\mathbf{w}_1(\theta) \leftarrow 1$ and $\eta_1(\theta) \leftarrow \frac{\underline{\eta}(\theta)+\overline{\eta}(\theta)}{2}$ // initialize MW and EWOO

for *task* $t = 1, \ldots, T$ **do**
- sample θ_t from Θ_k w.p. $\propto \exp(\mathbf{w}_t)$ // sample from MW distribution
- $\hat{\mathbf{x}}_t \leftarrow \texttt{OMD}_{\eta_t(\theta_t),\theta_t}(\mathbf{c}_{\theta_t}(\mathbf{x}_t))$ // run bandit OMD within-task
- $\mathbf{x}_{t+1} \leftarrow \frac{1}{t}\sum_{s=1}^t \hat{\mathbf{x}}_s$ // FTL update of initialization
- **for** $\theta \in \Theta_k$ **do**
  - $\eta_{t+1}(\theta) \leftarrow \frac{\int_{\underline{\eta}(\theta)}^{\overline{\eta}(\theta)} v \exp\left(-\alpha(\theta)\sum_{s=1}^t U_s^{(\rho)}(\mathbf{x}_s, v, \theta)\right) dv}{\int_{\underline{\eta}(\theta)}^{\overline{\eta}(\theta)} \exp\left(-\alpha(\theta)\sum_{s=1}^t U_s^{(\rho)}(\mathbf{x}_s, v, \theta)\right) dv}$ // EWOO step-size update
  - $\mathbf{w}_{t+1}(\theta) \leftarrow \mathbf{w}_t(\theta) - \lambda U_t(\mathbf{x}_t, \eta_t(\theta), \theta)$ // MW update to tune θ

$k = \mathcal{O}(\sqrt[4]{d}\sqrt{T})$. Computationally, at each timestep t and for each grid point we must compute two single-dimensional integrals; the integrands are sums of upper bounds that just need to be incremented once per round, leading to a total per-iteration complexity of $\mathcal{O}(k)$ (ignoring the running of OMD). Although outside the scope of this thesis, it may be possible to avoid integration by tuning η with MW as well, rather than EWOO, but likely at the cost of worse regret because it would not take advantage of the exp-concavity of $U_t^{(\rho)}$.

2.4.2 Multi-armed bandits

We now turn to our first application: the multi-armed bandit problem, where at each round i of task t we take action $a_{t,i} \in [d]$ and observe loss $\ell_{t,i}(a_{t,i}) \in [0,1]$. As we are sampling actions from distributions $\mathbf{x} \in \mathcal{K} = \triangle$ on the k-simplex, the inner product $\langle \boldsymbol{\ell}_{t,i}, \mathbf{x}_{t,i}\rangle$ is the expected loss and the optimal arm $\mathring{a}_t$ on task t can be encoded as a vector $\mathring{\mathbf{x}}_t$ s.t. $\mathring{\mathbf{x}}_{t[a]} = 1_{a=\mathring{a}_t}$.

We use as a base learner a generalization of Exp3 that uses the negative Tsallis entropy $\psi_\beta(\mathbf{p}) = \frac{1-\sum_{a=1}^d \mathbf{p}_{[a]}^\beta}{1-\beta}$ for some $\beta \in (0,1]$ as the regularizer; this improves regret from Exp3's $\mathcal{O}(\sqrt{dm\log d})$ to the optimal $\mathcal{O}(\sqrt{dm})$ [Abernethy et al., 2015]. Note that $-\psi_\beta$ is the Shannon entropy in the limit $\beta \to 1$ and its Bregman divergence $\mathcal{B}_\beta(\mathbf{x}||\cdot)$ is nonconvex in the second argument. As the Tsallis entropy is non-Lipschitz at the simplex boundary, which is where the estimated and true optima $\hat{\mathbf{x}}_t$ and $\mathring{\mathbf{x}}_t$ lie, we will project them using $\mathbf{c}_{\frac{\varepsilon}{1-\varepsilon}}(\mathbf{x}) = (1-\varepsilon)\mathbf{x} + \varepsilon \mathbf{1}_d/d$ to the set $\mathcal{K}_{\frac{\varepsilon}{1-\varepsilon}} = \{\mathbf{x} \in \triangle : \min_a \mathbf{x}_{[a]} \geqslant \varepsilon/d\}$. We denote the resulting vectors using the superscript (ε), e.g. $\hat{\mathbf{x}}_t^{(\varepsilon)} = \mathbf{c}_{\frac{\varepsilon}{1-\varepsilon}}(\hat{\mathbf{x}}_t)$, and also use $\triangle^{(\varepsilon)} = \mathcal{K}_{\frac{\varepsilon}{1-\varepsilon}}$ to denote the constrained simplex. For MAB we also study two base learners: (1) **implicit exploration** and (2) **guaranteed exploration**. The

former uses low-variance loss *under*-estimators $\hat{\ell}_{t,i}(a) = \frac{\ell_{t,i}(a)1_{a_{t,i}=a}}{\mathbf{x}_{t,i[a]}+\gamma}$ for $\gamma > 0$, where $\mathbf{x}_{t,i[a]}$ is the probability of sampling a on task t round i, to enable high probability bounds [Neu, 2015]. On the other hand, **guaranteed exploration** uses unbiased loss estimators (i.e. $\gamma = 0$) but constrains the action space to $\triangle^{(\varepsilon)}$, which we will use to adapt to a task similarity determined by the *true* optima-in-hindsight.

Adapting to low estimated entropy with high probability using implicit exploration

In our first setting, the base learner runs $\texttt{OMD}_{\eta_t,\beta_t}(\mathbf{x}_{t,1})$ on γ-regularized estimators with Tsallis regularizer ψ_{β_t}, step-size η_t, and initialization $\mathbf{x}_{t,1} \in \triangle^{(\varepsilon)}$. Standard OMD analysis combined with implicit exploration analysis [Neu, 2015] shows (2.165) that the task-averaged regret is bounded w.h.p. by

$$(\varepsilon + \gamma d)m + \tilde{\mathcal{O}}\left(\frac{\sqrt{d}}{\gamma T}\right) + \frac{1}{T}\sum_{t=1}^{T}\frac{\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)}||\mathbf{x}_{t,1})}{\eta_t} + \frac{\eta_t d^{\beta_t} m}{\beta_t} \tag{2.39}$$

The summands have the desired form of $U_t(\mathbf{x}_{t,1}, \eta_t, \beta_t)$, so by Theorem 2.4.1 we can bound their average by

$$\min_{\beta\in[\underline{\beta},\overline{\beta}],\eta>0} \frac{\hat{V}_\beta^2}{\eta} + \frac{\eta d^\beta m}{\beta} + \tilde{\mathcal{O}}\left(\frac{L_\eta}{k} + \frac{\left(\frac{d}{\varepsilon}\right)^{2-\underline{\beta}}}{\eta T} + \left(\rho + \frac{1}{\rho\sqrt{T}} + \frac{1}{\rho^2 T}\right)d\sqrt{m}\right) \tag{2.40}$$

where $\hat{V}_\beta^2 = \frac{1}{T}\sum_{t=1}^{T}\psi_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}) - \psi_\beta(\hat{\bar{\mathbf{x}}}^{(\varepsilon)})$ is the average difference in Tsallis entropies between the (ε-constrained) estimated optima $\hat{\mathbf{x}}_t$ and their empirical distribution $\hat{\bar{\mathbf{x}}} = \frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{x}}_t$, while L_η is the Lipschitz constant of $\frac{\hat{V}_\beta^2}{\eta} + \frac{\eta d^\beta m}{\beta}$ w.r.t. $\beta \in [\underline{\beta}, \overline{\beta}]$. The specific instantiation of Algorithm 6 that (2.40) holds for is to do the following at each time t:

1. sample β_t via the MW distribution $\propto \exp(\mathbf{w}_t)$ over the discretization Θ_k of $[\underline{\beta}, \overline{\beta}] \subset [0, 1]$
2. run $\texttt{OMD}_{\eta_t,\beta_t}$ using the initialization $\mathbf{x}_{t,1} = \frac{1}{t-1}\sum_{s<t}\hat{\mathbf{x}}_t^{(\varepsilon)} = \frac{\varepsilon}{d}\mathbf{1}_d + \frac{1-\varepsilon}{t-1}\sum_{s<t}\hat{\mathbf{x}}_t$ (FTL)
3. update EWOO at each $\beta \in \Theta_k$ with loss $\frac{\mathcal{B}_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}||\mathbf{x}_{t,1})+\rho^2 D_\beta^2}{\eta} + \frac{\eta d^\beta m}{\beta}$, where $D_\beta^2 = \frac{d^{1-\beta}-1}{1-\beta}$
4. update $\mathbf{p}_{t+1}$ using multiplicative weights with expert losses $\frac{\mathcal{B}_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}||\mathbf{x}_{t,1})}{\eta} + \frac{\eta d^\beta m}{\beta}$

(2.41)

The final guarantee for this procedure, given in full in Theorem 2.A.14, follows by two properties of the Tsallis entropy $-\psi_\beta$: (1) its Lipschitzness w.r.t. $\beta \in [0, 1]$ (c.f. Lem 2.A.5) and (2) the fact that $\hat{V}_\beta^2$ is bounded by the entropy $\hat{H}_\beta = -\psi_\beta(\hat{\bar{\mathbf{x}}})$ of the empirical distribution of estimated optima (c.f. Lem 2.A.6), which yields our first notion of task similarity: *multi-armed bandit tasks are similar if the empirical distribution of their (estimated) optimal arms has low entropy*.

We exemplify the implications of Theorem 2.A.14 in Corollary 2.4.1, where we consider three regimes of the lower bound $\underline{\beta}$ on the entropy parameter: $\underline{\beta} = 1$, i.e. always using Exp3; $\underline{\beta} = 1/2$, which corresponds to the optimal worst-case setting [Abernethy et al., 2015]; and $\underline{\beta} = 1/\log d$, below which the OMD regret-upper-bound always worsens (and so it does not make sense to try $\beta < 1/\log d$).

Corollary 2.4.1 (c.f. Cors. 2.A.10, 2.A.11, and 2.A.12)**.** Suppose $\overline{\beta} = 1$ and we set the initialization, step-size, and entropy parameter of Tsallis OMD with implicit exploration via Algorithm 6 as in Theorem 2.A.14.

1. If $\underline{\beta} = 1$ and $T \geqslant \frac{d^2}{m}$ we can ensure that w.h.p.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m}\ell_{t,i}(\mathbf{x}_{t,i}) - \ell_{t,i}(\mathring{\mathbf{x}}_t) \leqslant 2\sqrt{\hat{H}_1 dm} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{2}{3}}m^{\frac{2}{3}}}{\sqrt[3]{T}}\right) \tag{2.42}$$

2. If $\underline{\beta} = \frac{1}{2}$ and $T \geqslant \frac{d^{5/2}}{m}$ we can set $k = \lceil\sqrt[4]{d}\sqrt{T}\rceil$ and ensure w.h.p. that task-averaged regret is

$$\min_{\beta\in[\frac{1}{2},1]} 2\sqrt{\hat{H}_\beta d^\beta m/\beta} + \tilde{\mathcal{O}}\left(\frac{d^{5/7}m^{5/7}}{T^{2/7}} + \frac{d\sqrt{m}}{\sqrt[4]{T}}\right) \tag{2.43}$$

3. If $\underline{\beta} = \frac{1}{\log d}$ and $T \geqslant \frac{d^3}{m}$ we can set $k = \lceil\sqrt[4]{d}\sqrt{T}\rceil$ and ensure w.h.p. that task-averaged regret is

$$\min_{\beta\in(0,1]} 2\sqrt{\hat{H}_\beta d^\beta m/\beta} + \tilde{\mathcal{O}}\left(\frac{d^{3/4}m^{3/4} + d\sqrt{m}}{\sqrt[4]{T}}\right) \tag{2.44}$$

In all three settings, as $T \to \infty$ the regret scales directly with the entropy of the estimated optima-in-hindsight, which is small if most tasks are estimated to be solved by one of a few arms and large if all arms are used roughly equally. Corollary 2.4.1 demonstrates the importance of tuning β: even if tasks are dissimilar, we asymptotically recover the worst-case optimal guarantee $\mathcal{O}(\sqrt{dm})$ in cases two and three because the entropy is at most $\frac{d^{1-\beta}}{1-\beta}$. On the other hand, if a constant $s \ll d$ actions are always minimizers, i.e. the empirical distribution $\hat{\mathring{\mathbf{x}}}$ is s-sparse, then the last bound (2.44) implies that Algorithm 6 can achieve task-averaged regret $o_T(md) + \mathcal{O}(\sqrt{sm\log d})$. At the same time, this tuning is costly, with the last two results having an extra $\tilde{\mathcal{O}}\left(\frac{d\sqrt{m}}{\sqrt[4]{T}}\right)$ term because of it. Furthermore, the bound of $\underline{\beta} = \frac{1}{2}$ has a slightly better dependence on d, m, and T compared to that of $\underline{\beta} = \frac{1}{\log d}$ due to the $\left(\frac{d}{e}\right)^{2-\underline{\beta}}$ term in the bound (2.40) returned for MAB by our structural result.

We can compare the s-sparse result to Azizi et al. [2022], who achieve task-averaged regret $\tilde{\mathcal{O}}(m/\sqrt[3]{T} + \sqrt{sm\log T})$ for *stochastic* MAB. Despite our adversarial setting and no stipulations on how tasks are related, our bounds are asymptotically comparable if the estimated and true optima are roughly equivalent (ignoring their $\mathcal{O}(\sqrt{\log T})$-factor), as we also have $\tilde{\mathcal{O}}(\sqrt{sm})$ average regret as $T \to \infty$. Their rate in the number of tasks is better, but at a cost of runtime exponential in s. Apart from generality, we believe a great strength of our results is their adaptiveness; unlike Azizi et al. [2022], we do not need to know how many optimal arms there are to adapt to there being few of them.

Adapting to the entropy of the true optima-in-hindsight using guaranteed exploration

While the entropy of estimated optima-in-hindsight may be useful in some cases where we wish to actually *compute* the task similarity, it is otherwise generally more desirable to adapt to an intrinsic and algorithm-independent measure, e.g. the entropy of the *true* optima-in-hindsight. However, doing so is difficult without further assumptions, as the optima are both hard to identify

and the measure itself may not be well-defined in case of ties. Thus in this section we study the setting where we have a nonzero performance gap $\Delta > 0$ between the best and second-best arms:

Assumption 2.4.1. For some $\Delta > 0$ and all tasks $t \in [T]$, $\frac{1}{m}\sum_{i=1}^{m} \ell_{t,i}(a) - \ell_{t,i}(\mathring{a}_t) \geqslant \Delta \; \forall \, a \neq \mathring{a}_t$.

This assumption is common in the best-arm identification literature [Jamieson and Talwalkar, 2015, Abbasi-Yadkori et al., 2018], which we adapt to show that the estimated optimal arms match the true optima, and thus so do their entropies. To do so, we switch to *unbiased* loss estimators, i.e. $\gamma = 0$, and control their variance by lower-bounding the probability of selecting an arm to be at least $\frac{\varepsilon}{d}$; this can alternatively be expressed as running OMD using the regularizer $\psi_\beta + I_{\triangle(\varepsilon)}$, where for any $\mathcal{C} \subset \mathbb{R}^d$ the function $I_{\mathcal{C}}(\mathbf{x}) = 0$ if $\mathbf{x} \in \mathcal{C}$ and ∞ otherwise. Guaranteed exploration allows us extend the analysis of Abbasi-Yadkori et al. [2018] to show that the estimated arm is optimal w.h.p.:

Lemma 2.4.1 (c.f. Lem 2.A.8)**.** Suppose for $\varepsilon > 0$ and any $\beta \in (0, 1]$ we run OMD on task $t \in [T]$ with regularizer $\psi_\beta + I_{\triangle(\varepsilon)}$. If $m = \tilde{\Omega}(\frac{d}{\varepsilon\Delta^2})$ then $\hat{\mathbf{x}}_t = \mathring{\mathbf{x}}_t$ w.p. $\geqslant 1 - d\exp(-\Omega(\varepsilon\Delta^2 m/d))$.

However, the constraint that the probabilities are at least $\frac{\varepsilon}{d}$ does lead to εm additional error on each task, with the upper bound on the task-averaged expected regret becoming

$$\mathbb{E}\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(a_{t,i}) - \ell_{t,i}(\mathring{a}_t) \leqslant \varepsilon m + \frac{1}{T}\sum_{t=1}^{T} \frac{\mathbb{E}\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}_{t,1})}{\eta_t} + \frac{\eta_t d^{\beta_t} m}{\beta_t} \tag{2.45}$$

Moreover, we will no longer set $\varepsilon = o_T(1)$, as this would require m to be *increasing* in T for the best-arm identification result of Lemma 2.A.8 to hold. Thus, unlike in the previous section, our results will contain "fast" terms—terms in the task-averaged regret that are $o(\sqrt{m})$ but not decreasing in T nor affected by the task similarity. They will still improve upon the $\Omega(\sqrt{dm})$ MAB lower bound if tasks are similar, but the task-averaged regret will not converge to zero as $T \to \infty$ if the tasks are identical.

Nevertheless, the tuning-dependent component of the upper bounds in (2.45) has the appropriate form for our structural result—in fact we can use the same meta-algorithm (2.41) as for implicit exploration—and so we can again apply Theorem 2.4.1 to get a bound on the task-averaged regret in terms of the average difference $\hat{V}_\beta^2 = \frac{1}{T}\sum_{t=1}^{T} \psi_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}) - \psi_\beta(\hat{\bar{\mathbf{x}}}^{(\varepsilon)})$ of the entropies of the ε-constrained estimated task optima $\hat{\mathbf{x}}_t^{(\varepsilon)}$ and their mean $\hat{\bar{\mathbf{x}}}^{(\varepsilon)}$. The easiest way to apply Lemma 2.A.8 to bound $\hat{V}_\beta^2$ in terms of $H_\beta = \frac{1}{T}\sum_{t=1}^{T} \psi_\beta(\mathring{\mathbf{x}}_t) - \psi_\beta(\mathring{\bar{\mathbf{x}}})$ is via union bound on all T tasks to show that $\hat{\mathbf{x}}_t = \mathring{\mathbf{x}}_t \; \forall \, t$ w.p. $\geqslant 1 - dT\exp(-\Omega(\varepsilon\Delta^2 m/d))$; however, setting a constant failure probability leads to m growing, albeit only logarithmically, in T. Instead, by analyzing the worst-case best-arm identification probabilities, we show in Lemma 2.A.9 that the expectation of $\hat{V}_\beta^2$ is bounded by $H_\beta + 3\beta\frac{(d/\varepsilon)^{1-\beta}-1}{1-\beta}\exp\left(-\frac{3\varepsilon\Delta^2 m}{28d}\right)$ without resorting to $m = \omega_T(1)$. Assuming $m \geqslant \frac{75d}{\varepsilon\Delta^2}\log\frac{d}{\varepsilon\Delta^2}$ is enough (2.190) to bound the second term by $\frac{56}{dm}$. Then the final result (c.f. Theorem 2.A.15) bounds the expected task-averaged regret as follows (ignoring terms that become $o_T(1)$ after setting ρ and k):

$$\varepsilon m + \min_{\beta \in [\underline{\beta}, \overline{\beta}], \eta > 0} \frac{h_\beta(\Delta)}{\eta} + \frac{\eta d^\beta m}{\beta} \quad \text{for} \quad h_\beta(\Delta) = \begin{cases} H_\beta + \frac{56}{md} & \text{if} \quad m \geqslant \frac{75d}{\varepsilon\Delta^2}\log\frac{d}{\varepsilon\Delta^2} \\ \frac{d^{1-\beta}-1}{1-\beta} & \text{otherwise} \end{cases} \tag{2.46}$$

If the gap Δ is known and sufficiently large, then we can set $\varepsilon = \Theta(\frac{d}{\Delta^2 m})$ to obtain an asymptotic task-averaged regret that scales only with the entropy H_β and a fast term that is logarithmic in m:

Corollary 2.4.2 (c.f. Cor. 2.A.15)**.** Suppose we set the initialization, step-size, and entropy parameter of Tsallis OMD with guaranteed exploration via Algorithm 6 as specified in Theorem 2.A.15. If $[\underline{\beta}, \overline{\beta}] = [\frac{1}{\log d}, 1]$ and $m \geqslant \frac{75d}{\Delta^2} \log \frac{d}{\Delta^2}$, then setting $\varepsilon = \tilde{\Theta}\left(\frac{d}{\Delta^2 m}\right)$, $\rho = \frac{1}{\sqrt[3]{d}\sqrt[6]{mT}}$, and $k = \lceil \sqrt[3]{d^2 mT} \rceil$ ensures that the expected task-averaged regret is at most

$$\min_{\beta \in (0,1]} 2\sqrt{H_\beta d^\beta m/\beta} + \tilde{\mathcal{O}}\left(\frac{d}{\Delta^2} + \frac{d^{\frac{4}{3}} m^{\frac{2}{3}}}{\sqrt[3]{T}} + \frac{d^{\frac{5}{3}} m^{\frac{5}{6}}}{T^{\frac{2}{3}}} + \frac{d\Delta^4 m^3}{T}\right) \tag{2.47}$$

Knowing the gap Δ is a strong assumption, as ideally we could set ε without it. Note that if $\varepsilon = \Omega(\frac{1}{m^p})$ for some $p \in (0,1)$ then the condition $m \geqslant \frac{75d}{\varepsilon\Delta^2} \log \frac{d}{\varepsilon\Delta^2}$ only fails if $m \leqslant \mathrm{poly}(\frac{1}{\Delta})$, i.e. for gap decreasing in m. We can use this together with the fact that minimizing over η and β in our bound allows us to replace them with any value, even a gap-dependent one, to derive a gap-*independent* setting of ε that ensures a task similarity-adaptive bound when Δ is not too small and falls back to the worst-case optimal guarantee otherwise. Specifically, for indicator $\iota_\Delta = 1_{m \geqslant \frac{75d}{\varepsilon\Delta^2} \log \frac{d}{\varepsilon\Delta^2}}$, setting $\eta = \Theta\left(\sqrt{\frac{h_\beta(\Delta)}{d^\beta m/\beta}}\right)$ in (2.46) and using $\beta = \frac{1}{2}$ if the condition ι_Δ fails yields asymptotic regret at most

$$\begin{aligned} & \varepsilon m + \min_{\beta \in (0,1]} \mathcal{O}\left(\iota_\Delta \sqrt{\tfrac{H_\beta d^\beta m}{\beta}} + (1-\iota_\Delta)\sqrt{dm}\right) \\ \leqslant \quad & \varepsilon m + \tilde{\mathcal{O}}\left(\min\left\{\min_{\beta \in (0,1]} \sqrt{\tfrac{H_\beta d^\beta m}{\beta}} + \frac{d}{\Delta\sqrt{\varepsilon}}, \sqrt{dm}\right\}\right) \end{aligned} \tag{2.48}$$

Setting $\varepsilon = \Theta(\sqrt{d}/m^{\frac{2}{3}})$ yields the desired dependence on the entropy H_β and a fast term in m:

Corollary 2.4.3 (c.f. Cor. 2.A.16)**.** In the setting of Corollary 2.4.2 but with $m = \Omega(d^{\frac{3}{4}})$ and unknown Δ, using $\varepsilon = \Theta(\sqrt{d}/m^{\frac{2}{3}})$ ensures expected task-averaged regret at most

$$\min\left\{\min_{\beta \in (0,1]} 2\sqrt{H_\beta d^\beta m/\beta} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{3}{4}}\sqrt[3]{m}}{\Delta}\right), 8\sqrt{dm}\right\} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{4}{3}} m^{\frac{2}{3}}}{\sqrt[3]{T}} + \frac{d^{\frac{5}{3}} m^{\frac{5}{6}}}{T^{\frac{2}{3}}} + \frac{d^2 m^{\frac{7}{3}}}{T}\right) \tag{2.49}$$

While not logarithmic, the gap-dependent term is still $o(\sqrt{m})$, and moreover the asymptotic regret is no worse than the worst-case optimal $\mathcal{O}(\sqrt{dm})$. Note that the latter is only needed if $\Delta = o(1/\sqrt[6]{m})$.

The main improvement in this section is in using the entropy of the true optima, which can be much smaller than that of the estimated optima if there are a few good arms but large noise. Our use of the gap assumption for this seems difficult to avoid for this notion of task similarity. We can also compare to Corollary 2.4.1 (2.44), which did not require $\Delta > 0$ and had no fast terms but had a worse rate in T; in contrast, the $\mathcal{O}(\frac{1}{\sqrt[3]{T}})$ rates above match that of the closest stochastic bandit result [Azizi et al., 2022]. As before, for $s \ll d$ "good" arms we obtain $\mathcal{O}(\sqrt{sm \log d})$ asymptotic regret, assuming the gap is not too small. Finally, we can also compare to the classic

shifting regret bound for Exp3.S [Auer et al., 2002], which translated to task-averaged regret is $\mathcal{O}(\sqrt{dm\log(dmT)})$. This is worse than even running OMD separately on each task, albeit under weaker assumptions (not knowing task boundaries). It also cannot take advantage of repeated optimal arms, e.g. the case of $s \ll d$ good arms.

Adapting to entropic task similarity implies robustness to outliers

While we considered mainly the s-sparse setting as a way of exemplifying our results and comparing to other work such as Azizi et al. [2022], the fact that our approach can adapt to the Tsallis entropy $\min_\beta H_\beta$ of the optimal arms implies meaningful guarantees for any low-entropy distribution over the optimal arms, not just sparsely-supported ones. One way to illustrate the importance of this is through an analysis of robustness to outlier tasks. Specifically, suppose that the s-sparsity assumption—that optima $\mathring{a}_t$ lie in a subset of $[T]$ of size $s \ll d$—only holds for all but $\mathcal{O}(T^p)$ of the tasks $t \in [T]$, where $p \in [0, 1)$. Then the best we can do using an asymptotic bound of $\tilde{\mathcal{O}}(\sqrt{sm})$—e.g. that of Azizi et al. [2022] in the stochastic case or from naively applying $\min_{\beta\in(0,1]} H_\beta d^\beta m/\beta \leqslant esm\log d$ to any of our previous results—is to substitute $s + T^p$ instead of s, which will only improve over the single-task bound if $d = \omega(T^p)$, i.e. in the regime where the number of arms increases with the number of tasks.

However, our notion of task similarity allows us to do much better, as we can show (c.f. Proposition 2.A.6) that in the same setting $H_\beta = \mathcal{O}(s+\frac{d^{1-\beta}}{T^{\beta(1-p)}})$ for any $\beta \in [\frac{1}{\log d}, \frac{1}{2}]$. Substituting this result into e.g. Corollary 2.4.3 yields the same asymptotic result of $\mathcal{O}(\sqrt{sm\log d})$, although the rate in T is a very slow $\mathcal{O}(\sqrt{dm}/T^{\frac{1-p}{2\log d}})$. This demonstrates how our entropic notion of task similarity simultaneously yields strong results in the s-sparse setting and is meaningful in more general settings.

2.4.3 Bandit linear optimization

Our last application is bandit linear optimization, in which at task t round i we play $\mathbf{x}_{t,i} \in \mathcal{K}$ in some convex $\mathcal{K} \subset \mathbb{R}^d$ and observe loss $\langle \boldsymbol{\ell}_{t,i}, \mathbf{x}_{t,i}\rangle \in [-1, 1]$. We will again use a variant of mirror descent, using a **self-concordant barrier** for ψ and the specialized loss estimators of Abernethy et al. [2008b, Algorithm 1]. More information on such regularizers can be found in the literature on interior point methods [Nesterov and Nemirovskii, 1994]. We pick this class of algorithms because of their optimal dependence on the number of rounds and their applicability to any convex domain $\mathcal{K}$ via specific barriers ψ, which will yield interesting notions of task similarity. Our ability to handle nonsmooth regularizers via the structural result (Theorem 2.4.1) is even more important here, as barriers are infinite at the boundaries. Indeed, we will *not* learn a β parameterizing the regularizer and instead focus on tuning a boundary offset $\varepsilon > 0$. Here we make use of notation from Section 2.4.1, where $\mathbf{c}_\varepsilon$ maps points in $\mathcal{K}$ to a subset $\mathcal{K}_\varepsilon$ defined by the Minkowski function (c.f. Definition 2.A.2) centered at $\mathbf{x}_1 = \arg\min_{\mathbf{x}\in\mathcal{K}} \psi(\mathbf{x})$.

From Abernethy et al. [2008b] we have an upper bound on the expected task-averaged regret of their algorithm run from initializations $\mathbf{x}_{t,1} \in \mathcal{K}^\circ$ with step-sizes $\eta_t > 0$ and offsets $\varepsilon_t > 0$:

$$\mathbb{E}\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m}\langle \boldsymbol{\ell}_{t,i}, \mathbf{x}_{t,i} - \mathring{\mathbf{x}}_t\rangle \leqslant \frac{1}{T}\sum_{t=1}^{T}\frac{\mathbb{E}\mathcal{B}(\mathbf{c}_{\varepsilon_t}(\hat{\mathbf{x}}_t)||\mathbf{x}_{t,1})}{\eta_t} + (32\eta_t d^2 + \varepsilon_t)m \tag{2.50}$$

We can show (2.209) that $D_\varepsilon^2 = \max_{\mathbf{x},\mathbf{y}\in\mathcal{K}_\varepsilon} \mathcal{B}(\mathbf{x}||\mathbf{y}) \leqslant \frac{9\nu^{\frac{3}{2}}K\sqrt{S_1}}{\varepsilon}$, where ν is the self-concordance constant of ψ and $S_1 = \|\nabla^2\psi(\mathbf{x}_1)\|_2$ is the spectral norm of its Hessian at the center $\mathbf{x}_1$ of $\mathcal{K}$. Restricting to tuning $\varepsilon \in [\frac{1}{m}, 1]$—which is enough to obtain constant task-averaged regret above if the estimated optima $\hat{\mathbf{x}}_t$ are identical—we can now apply Algorithm 6 via the following instantiation:

1. sample ε_t via the MW distribution $\propto \exp(\mathbf{w}_t)$ over the discretization Θ_k of $[\frac{1}{m}, 1]$
2. run $\texttt{OMD}_{\eta_t,\varepsilon_t}$ using the initialization $\mathbf{x}_{t,1} = \frac{1}{t-1}\sum_{s<t}\mathbf{c}_{\varepsilon_t}(\hat{\mathbf{x}}_t) = \mathbf{x}_1 + \frac{\sum_{s<t}\hat{\mathbf{x}}_t - \mathbf{x}_1}{(1+\varepsilon_t)(t-1)}$ (FTL)
3. update EWOO at each $\varepsilon \in \Theta_k$ with loss $\frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x}_{t,1})+\rho^2 D_\varepsilon^2}{\eta} + 32\eta d^2$ for $D_\varepsilon^2 = \frac{9\nu^{\frac{3}{2}}K\sqrt{S_1}}{\varepsilon}$
4. update $\mathbf{p}_{t+1}$ using multiplicative weights with expert losses $\frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x}_{t,1})}{\eta} + \varepsilon m$

(2.51)

Note the similarity to the MAB case (2.41), with the difference being the upper bound passed to EWOO and MW. Our structural result bounds the expected task-averaged regret as follows (c.f. Theorem 2.A.16):

$$\begin{aligned}\mathbb{E}\min_{\varepsilon\in[\frac{1}{m},1],\eta>0} &\frac{\hat{V}_\varepsilon^2}{\eta} + (32\eta d^2 + \varepsilon)m \\ &+ \tilde{\mathcal{O}}\left(\frac{\frac{m^2}{T}+\frac{1}{k}}{\eta} + \frac{m}{k} + m\min\left\{\frac{\rho^2}{\eta}, d\rho\right\} + \frac{dm}{\rho}\sqrt{\frac{\log k}{T}} + \frac{dm}{\rho^2 T}\right)\end{aligned} \tag{2.52}$$

For $\rho = o_T(1)$ and $k = \omega_T(1)$ this becomes $o_T(\text{poly}(m)) + \mathbb{E}\min_{\varepsilon\in[\frac{1}{m},1],\eta>0}\frac{\hat{V}_\varepsilon^2}{\eta} + 32\eta d^2 m + \varepsilon m$, where $\hat{V}_\varepsilon^2 = \frac{1}{T}\sum_{t=1}^T \psi(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t) - \psi(\mathbf{c}_\varepsilon(\hat{\bar{\mathbf{x}}}_t)$. Then by tuning η we get an asymptotic ($T\to\infty$) regret of $4d\hat{V}_\varepsilon\sqrt{2m} + \varepsilon m$ for any $\varepsilon \in [\frac{1}{m}, 1]$. Our analysis removes the explicit dependence on $\sqrt{\nu}$ that appears in the single-task regret [Abernethy et al., 2008b]; as an example, ν equals the number of inequalities defining a polytope $\mathcal{K}$, as in the bandit shortest-path application below.

The remaining challenge is to interpret $\hat{V}_\varepsilon^2$, which as we did for MAB we do via specific examples, in this case concrete action domains $\mathcal{K}$. Our first example is for BLO over the unit sphere $\mathcal{K} = \{\mathbf{x}\in\mathbb{R}^d : \|\mathbf{x}\|_2 \leqslant 1\}$ using the appropriate log-barrier regularizer $\psi(\mathbf{x}) = -\log(1-\|\mathbf{x}\|_2^2)$:

Corollary 2.4.4 (c.f. Cor. 2.A.17)**.** For BLO on the sphere, Algorithm 6 has expected task-averaged regret

$$\tilde{\mathcal{O}}\left(\frac{dm^{\frac{3}{2}}}{T^{\frac{3}{4}}} + \frac{dm}{\sqrt[4]{T}}\right) + \min_{\varepsilon\in[\frac{1}{m},1]} 4d\sqrt{2m\log\left(1 + \frac{1-\mathbb{E}\|\hat{\bar{\mathbf{x}}}\|_2^2}{2\varepsilon+\varepsilon^2}\right)} + \varepsilon m \tag{2.53}$$

The bound above is decreasing in $\mathbb{E}\|\hat{\bar{\mathbf{x}}}\|_2^2$, the expected squared norm of the average of the estimated optima $\hat{\mathbf{x}}_t$. We thus say that *bandit linear optimization tasks over the sphere are similar if the norm of the empirical mean of their (estimated) optima is large*. This makes intuitive sense: if the tasks' optima are uniformly distributed, we should expect $\mathbb{E}\|\hat{\bar{\mathbf{x}}}\|_2^2$ to be small, even decreasing in d. On the other hand, in the degenerate case where the estimated optima $\hat{\mathbf{x}}_t$ are the same across all tasks $t\in[T]$, we have $\mathbb{E}\|\hat{\bar{\mathbf{x}}}\|_2^2 = 1$, so the asymptotic task-averaged regret is 1

because we can use $\varepsilon = \frac{1}{m}$. Perhaps slightly more realistically, if it is $\frac{1}{m^p}$-away from 1 for some power $p \geqslant \frac{1}{2}$ then setting $\varepsilon = \frac{1}{\sqrt{m}}$ can remove the logarithmic dependence on m. These two regimes illustrate the importance of tuning ε.

As a last application, we apply our meta-BLO result to the shortest-path problem in online optimization [Takimoto and Warmuth, 2003, Kalai and Vempala, 2005]. In its bandit variant [Awerbuch and Kleinberg, 2004, Dani et al., 2008], at each step $i = 1, \dots, m$ the player must choose a path p_i from a fixed source $u \in V$ to a fixed sink $v \in V$ in a directed graph $G(V, E)$. At the same time the adversary chooses edge-weights $\ell_i \in \mathbb{R}^{|E|}$ and the player suffers the sum $\sum_{e \in p_t} \ell_i(e)$ of the weights in their chosen path p_t. This can be relaxed as BLO over vectors $\mathbf{x}$ in a set $\mathcal{K} \subset [0,1]^{|E|}$ defined by a set $\mathcal{C}$ of $\mathcal{O}(|E|)$ linear constraints $(\mathbf{a}, b)\ \langle \mathbf{a}, \mathbf{x} \rangle \leqslant b$ enforcing flows from u to v; u to v paths can be sampled from any $\mathbf{x} \in \mathcal{K}$ in an unbiased manner [Abernethy et al., 2008b, Proposition 1]. On a single-instance, applying the BLO method of Abernethy et al. [2008b] ensures $\mathcal{O}(|E|^{\frac{3}{2}}\sqrt{m})$ regret on this problem.

In the multi-instance setting, comprising a sequence $t = 1, \dots, T$ of shortest path instances with m adversarial edge-weight vectors $\ell_{t,i}$ each, we can attempt to achieve better performance by tuning the same method across instances. Notably, we can view this as the problem of learning predictions in the algorithms with predictions paradigm (c.f. Part II), with the OMD initialization on each instance being effectively a prediction of its optimal path. Our meta-learner then has the following average performance across bandit shortest-path instances:

Corollary 2.4.5 (c.f. Cor. 2.A.18)**.** For multi-task bandit online shortest path, Algorithm 6 with regularizer $\psi(\mathbf{x}) = -\sum_{\mathbf{a},b \in \mathcal{C}} \log(b - \langle \mathbf{a}, \mathbf{x} \rangle)$ attains the following expected average regret across instances

$$\tilde{\mathcal{O}}\left(\frac{|E|^4 m^{\frac{3}{2}}}{T^{\frac{3}{4}}} + \frac{|E|^{\frac{5}{2}} m^{\frac{5}{6}}}{\sqrt[4]{T}}\right) + \min_{\varepsilon \in [\frac{1}{m}, 1]} 4|E|\mathbb{E}\sqrt{2m \sum_{\mathbf{a},b \in \mathcal{C}} \log\left(\frac{\frac{1}{T}\sum_{t=1}^T b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t) \rangle}{\sqrt[T]{\prod_{t=1}^T b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t) \rangle}}\right)} + \varepsilon m \tag{2.54}$$

Here the asymptotic regret scales with the sum across all constraints $\mathbf{a}, b \in \mathcal{C}$ of the log of the ratio between the arithmetic and geometric means across tasks of the distances $b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t) \rangle$ from the estimated optimum flow $\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)$ to the constraint boundary. As it is difficult to separate the effect of the offset ε, we do not state an explicit task similarity measure like in our previous settings. Nevertheless, since the arithmetic and geometric means are equal exactly when all entries are equal—and otherwise the former is larger—the bound does show that regret is small when the estimated optimal flows $\hat{\mathbf{x}}_t$ for each task are at similar distances from the constraints, i.e. the boundaries of the polytope. Indeed, just as on the sphere, if the estimated optima are all the same then setting $\varepsilon = \frac{1}{m}$ again yields constant averaged regret.

2.4.4 Future work

In this section we applied ARUBA to design a meta-algorithm for learning to initialize and tune bandit algorithms, obtaining task-averaged regret guarantees for both multi-armed and linear bandits that depend on natural, setting-specific notions of task similarity. For MAB, we meta-learn the initialization, step-size, and entropy parameter of Tsallis-entropic OMD and show good

performance if the entropy of the optimal arms is small. For BLO, we use OMD with self-concordant regularizers and meta-learn the initialization, step-size, and boundary-offset, yielding interesting domain-specific task similarity measures. Some natural directions for future work involve overcoming some limitations of our results: can we adapt to a notion of task similarity that depends on the true optima without assuming a gap for MAB, or at all for BLO? Alternatively, can we design meta-learning algorithms that adapt to both stochastic and adversarial bandits, i.e. a "best-of-both-worlds" guarantee? Beyond this, one could explore other partial information settings, such as contextual bandits or bandit convex optimization.

2.5 Conclusion

This chapter introduces ARUBA, a learning-theoretic framework for deriving and analyzing algorithms for learning-to-learn. Our approach works by applying off-the-shelf learning guarantees to nice but meaningful bounds on the performance of learning algorithms; we show that this approach yields meta-learning methods that are similar to practical gradient-based meta-learning methods such as Reptile, scale to large scale models, and provably adapt to natural, setting-specific notions of task similarity. While motivated by gradient-based meta-learning, we also demonstrate that our approach extends to a variety of learning-theoretic settings, including nonconvex meta-learning and bandits. In the next chapter we will highlight its usefulness in showing provable guarantees for a new method for federated hyperparameter optimization, while in the next part of the thesis we extend ARUBA beyond learning algorithms to algorithms with predictions.

2.A Proofs

2.A.1 Strongly convex coupling

Our first result is a simple trick that we believe may be of independent interest. It allows us to bound the regret of FTL on any (possibly nonconvex) sequence of Lipschitz functions so long as the actions played are identical to those played on a different strongly-convex sequence of Lipschitz functions. The result is formalized in Theorem 2.A.1.

Derivation

We start with some standard facts about convex functions.

Claim 2.A.1. Let $f : \mathcal{X} \mapsto \mathbb{R}$ be an everywhere sub-differentiable convex function. Then for any norm $\|\cdot\|$ we have

$$f(\mathbf{x}) - f(\mathbf{y}) \leqslant \|\nabla f(\mathbf{x})\|_* \|\mathbf{x} - \mathbf{y}\| \;\forall\, \mathbf{x}, \mathbf{y} \in \mathcal{X} \tag{2.55}$$

Claim 2.A.2. Let $f : \mathcal{X} \mapsto \mathbb{R}$ be α-strongly-convex w.r.t. $\|\cdot\|$ with minimum $\mathbf{x}^* \in \operatorname*{arg\,min}_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$. Then $\mathbf{x}^*$ is unique and for all $\mathbf{x} \in \mathcal{X}$ we have

$$f(\mathbf{x}) \geqslant f(\mathbf{x}^*) + \frac{\alpha}{2}\|\mathbf{x} - \mathbf{x}^*\|^2 \tag{2.56}$$

Next we state to some technical results, starting with the well-known be-the-leader lemma:

Lemma 2.A.1 (Shalev-Shwartz [2011])**.** Let $\mathbf{x}_1, \ldots, \mathbf{x}_{T+1} \in \mathcal{X}$ be the sequence of actions of FTL on the function sequence $\{\ell_t : \mathcal{X} \mapsto \mathbb{R}\}_{t\in[T]}$. Then

$$\sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell_t(\mathbf{x}^*) \leqslant \sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell_t(\mathbf{x}_{t+1}) \tag{2.57}$$

for all $\mathbf{x}^* \in \mathcal{X}$.

The final result depends on a stability argument for FTL on strongly-convex functions adapted from Saha et al. [2012]:

Lemma 2.A.2. Let $\{\ell_t : \mathcal{X} \mapsto \mathbb{R}\}_{t\in[T]}$ be a sequence of functions that are α_t-strongly-convex w.r.t. $\|\cdot\|$ and let $\mathbf{x}_1, \ldots, \mathbf{x}_{T+1} \in \mathcal{X}$ be the corresponding sequence of actions of FTL. Then

$$\|\mathbf{x}_t - \mathbf{x}_{t+1}\| \leqslant \frac{2\|\nabla_t\|_*}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.58}$$

for all $t \in [T]$.

Proof. The proof slightly generalizes an argument in Saha et al. [2012, Theorem 6]. For each $t \in [T]$ we have by Claim 2.A.2 and the $\alpha_{1:t}$-strong-convexity of $\sum_{s=1}^{t} \ell_s(\cdot)$ that

$$\sum_{s=1}^{t} \ell_s(\mathbf{x}_t) \geqslant \sum_{s=1}^{t} \ell_s(\mathbf{x}_{t+1}) + \frac{\alpha_{1:t}}{2}\|\mathbf{x}_t - \mathbf{x}_{t+1}\|^2 \tag{2.59}$$

We similarly have

$$\sum_{s=1}^{t-1} \ell_s(\mathbf{x}_{t+1}) \geqslant \sum_{s=1}^{t-1} \ell_s(\mathbf{x}_t) + \frac{\alpha_{1:t-1}}{2}\|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 \tag{2.60}$$

Adding these two inequalities and applying Claim 2.A.1 yields

$$\left(\frac{\alpha_t}{2} + \alpha_{1:t-1}\right)\|\mathbf{x}_t - \mathbf{x}_{t+1}\|^2 \leqslant \ell_t(\mathbf{x}_t) - \ell_t(\mathbf{x}_{t+1}) \leqslant \|\nabla_t\|_*\|\mathbf{x}_t - \mathbf{x}_{t+1}\| \tag{2.61}$$

Dividing by $\|\mathbf{x}_t - \mathbf{x}_{t+1}\|$ yields the result. □

Theorem 2.A.1. Let $\{\ell_t : \mathcal{X} \mapsto \mathbb{R}\}_{t\in[T]}$ be a sequence of functions that are G_t-Lipschitz in $\|\cdot\|_A$ and let $\mathbf{x}_1, \ldots, \mathbf{x}_{T+1}$ be the sequence of actions produced by FTL. Let $\{\ell'_t : \mathcal{X} \mapsto \mathbb{R}\}_{t\in[T]}$ be a sequence of functions on which FTL also plays $\mathbf{x}_1, \ldots, \mathbf{x}_{T+1}$ but which are G'_t-Lipschitz and α_t-strongly-convex in $\|\cdot\|_B$. Then

$$\sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell_t(\mathbf{x}^*) \leqslant 2C\sum_{t=1}^{T} \frac{G_t G'_t}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.62}$$

for all $\mathbf{x}^* \in \mathcal{X}$ and some constant C s.t. $\|\mathbf{x}\|_A \leqslant C\|\mathbf{x}\|_B \;\forall\, \mathbf{x} \in \mathcal{X}$. If the functions ℓ_t are also convex then we have

$$\sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell_t(\mathbf{x}^*) \leqslant 2C\sum_{t=1}^{T} \frac{\|\nabla_t\|_{A,*}\|\nabla'_t\|_{B,*}}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.63}$$

or all $\mathbf{x}^* \in \mathcal{X}$

Proof. By Lemma 2.A.2,

$$\|\mathbf{x}_t - \mathbf{x}_{t+1}\|_A \leqslant C\|\mathbf{x}_t - \mathbf{x}_{t+1}\|_B \leqslant \frac{2CG'_t}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.64}$$

for all $t \in [T]$. Then by Lemma 2.A.1 and the G_t-Lipschitzness of ℓ_t we have for all $\mathbf{x}^* \in \mathcal{X}$ that

$$\sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell_t(\mathbf{x}^*) \leqslant \sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell(\mathbf{x}_{t+1}) \leqslant \sum_{t=1}^{T} G_t\|\mathbf{x}_t - \mathbf{x}_{t+1}\|_A \leqslant 2C\sum_{t=1}^{T} \frac{G_t G'_t}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.65}$$

In the convex case we instead apply Claim 2.A.1 and Lemma 2.A.2 to get

$$\sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell_t(\mathbf{x}^*) \leqslant \sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \ell(\mathbf{x}_{t+1}) \leqslant \sum_{t=1}^{T} \|\nabla_t\|_{A,*}\|\mathbf{x}_t - \mathbf{x}_{t+1}\|_A \leqslant 2C\sum_{t=1}^{T} \frac{\|\nabla_t\|_{A,*}\|\nabla'_t\|_{B,*}}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.66}$$

□

Applications

We now show two applications of strongly convex coupling. The first shows logarithmic regret for FTL run on a sequence of Bregman divergences generated by a fixed regularizer $\psi : \mathcal{X} \mapsto \mathbb{R}$. Note that such functions are nonconvex in general.

Proposition 2.A.1. Let $\psi : \mathcal{X} \mapsto \mathbb{R}$ be 1-strongly-convex w.r.t. $\|\cdot\|$ and consider any $\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathcal{X}$. Then when run on the loss sequence $\alpha_1 \mathcal{B}_\psi(\mathbf{x}_1||\cdot), \dots, \alpha_T \mathcal{B}_\psi(\mathbf{x}_T||\cdot)$ for any positive scalars $\alpha_1, \dots, \alpha_T \in \mathbb{R}_{>0}$, FTL obtains regret

$$\text{Regret} \leqslant 2CD \sum_{t=1}^{T} \frac{\alpha_t^2 G_t}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.67}$$

for C s.t. $\|\mathbf{x}\| \leqslant C\|\mathbf{x}\|_2 \ \forall \ \mathbf{x} \in \mathcal{X}$, $D = \max_{\mathbf{x},\mathbf{y}\in\mathcal{X}} \|\mathbf{x} - \mathbf{y}\|_2$ the ℓ_2-diameter of $\mathcal{X}$, and G_t the Lipschitz constant of $\mathcal{B}_\psi(\mathbf{x}_t||\cdot)$ over $\mathcal{X}$ w.r.t. $\|\cdot\|$. Note that for $\|\cdot\| = \|\cdot\|_2$ we have $C = 1$ and $G_t \leqslant D \ \forall \ t \in [T]$.

Proof. Note that $\alpha_t \mathcal{B}_\psi(\mathbf{x}_t||\cdot)$ is $\alpha_t G_t$-Lipschitz w.r.t. $\|\cdot\|$. Let $\psi'(\cdot) = \frac{1}{2}\|\cdot\|_2^2$, so $\mathcal{B}_{\psi'}(\mathbf{x}_t||\mathbf{y}) = \frac{1}{2}\|\mathbf{x}_t - \mathbf{y}\|_2^2 \ \forall \ \mathbf{y} \in \mathcal{X}, t \in [T]$. The function $\alpha_t \mathcal{B}_{\psi'}(\mathbf{x}_t||\cdot)$ is thus α_t-strongly-convex and D-Lipschitz w.r.t. $\|\cdot\|_2$. Now by Claim B.2.1 FTL run on this new sequence plays the same actions as FTL run on the original sequence. Applying Theorem 2.A.1 yields the result. □

We can state a more elegant result using a bound on the Hessian of the regularizer:

Corollary 2.A.1. Let $\psi : \mathcal{X} \mapsto \mathbb{R}$ be a strictly convex function with $\max_{\mathbf{x}\in\mathcal{X}} \|\nabla^2\psi(\mathbf{x})\|_\infty \leqslant S$ over a convex set $\mathcal{X} \subset \mathbb{R}^d$ of size $\max_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x}\|_2 \leqslant K$. Then for any points $\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathcal{X}$ the actions $\mathbf{y}_1 = \arg\min_{\mathbf{x}\in\mathcal{X}} \psi(\mathbf{x})$ and $\mathbf{y}_t = \frac{1}{t-1}\sum_{s<t} \mathbf{x}_s$ have regret

$$\sum_{t=1}^{T} \mathcal{B}_\psi(\mathbf{x}_t||\mathbf{y}_t) - \mathcal{B}_\psi(\mathbf{x}_t||\mathbf{y}_{T+1}) \leqslant \sum_{t=1}^{T} \frac{8SK^2}{2t-1} \leqslant 8SK^2(1 + \log T) \tag{2.68}$$

Proof. Note that

$$\nabla_{\mathbf{y}} \mathcal{B}_\psi(\mathbf{x}||\mathbf{y}) = -\nabla\psi(\mathbf{y}) - \nabla_{\mathbf{y}}\langle\nabla\psi(\mathbf{y}), \mathbf{x}\rangle + \nabla_{\mathbf{y}}\langle\nabla\psi(\mathbf{y}), \mathbf{y}\rangle = \text{diag}(\nabla^2\psi(\mathbf{y}))(\mathbf{y} - \mathbf{x}) \tag{2.69}$$

so $\mathcal{B}_\psi(\mathbf{x}_t||\cdot)$ is $2SK$-Lipschitz w.r.t. $\|\cdot\|_2$. Now if $\psi'(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ then the functions $\mathcal{B}_{\psi'}(\mathbf{x}_t||\cdot)$ are 1-strongly-convex and $2K$-Lipschitz w.r.t. $\|\cdot\|_2$. Therefore, since FTL run on this new sequence plays the same actions as FTL run on the original sequence, we can apply Theorem 2.A.1 to obtain the result. □

In the next application we use coupling to give a $\tilde{\mathcal{O}}(T^{\frac{3}{5}})$-regret algorithm for a sequence of non-Lipschitz convex functions.

Proposition 2.A.2. Let $\{\ell_t : \mathbb{R}_{>0} \mapsto \mathbb{R}\}_{t\geqslant 1}$ be a sequence of functions $\ell_t(x) = \left(\frac{B_t^2}{x} + x\right)\alpha_t$ for any positive scalars $\alpha_1, \dots, \alpha_T \in \mathbb{R}_{>0}$ and adversarially chosen $B_t \in [0, D]$. Then the ε-FTL

algorithm, which for $\varepsilon > 0$ uses the actions of FTL run on the functions $\tilde{\ell}_t(x) = \left(\frac{B_t^2+\varepsilon^2}{x} + x\right)\alpha_t$ over the domain $[\varepsilon, \sqrt{D^2+\varepsilon^2}]$ to determine x_t, achieves regret

$$\text{Regret} \leqslant \min\left\{\frac{\varepsilon^2}{x^*}, \varepsilon\right\}\alpha_{1:T} + 2D\max\left\{\frac{D^3}{\varepsilon^3}, 1\right\}\sum_{t=1}^{T}\frac{\alpha_t^2}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.70}$$

for all $x^* > 0$.

Proof. Define $\tilde{B}_t^2 = B_t^2 + \varepsilon^2$ and note that FTL run on the functions $\tilde{\ell}'_t(x) = \left(\frac{x^2}{2} - \tilde{B}_t^2 \log x\right)\alpha_t$ plays the exact same actions $x_t^2 = \frac{\sum_{s<t}\alpha_s\tilde{B}_s^2}{\alpha_{1:t-1}}$ as FTL run on $\tilde{\ell}_t$. We have that

$$|\partial_x\tilde{\ell}_t| = \alpha_t\left|1 - \frac{\tilde{B}_t^2}{x^2}\right| \leqslant \frac{\alpha_t D^2}{\varepsilon^2} \tag{2.71}$$

$$|\partial_x\tilde{\ell}'_t| = \alpha_t\left|x - \frac{\tilde{B}_t^2}{x}\right| \leqslant \alpha_t\max\left\{D, \frac{D^2}{\varepsilon}\right\} \qquad \partial_{xx}\tilde{\ell}'_t = \alpha_t\left(1 + \frac{\tilde{B}_t^2}{x^2}\right) \geqslant \alpha_t \tag{2.72}$$

so the functions $\tilde{\ell}_t$ are $\frac{\alpha_t D^2}{\varepsilon^2}$-Lipschitz while the functions $\tilde{\ell}'_t$ are $\alpha_t D\max\left\{\frac{D}{\varepsilon}, 1\right\}$-Lipschitz and α_t-strongly-convex. Therefore by Theorem 2.A.1 we have that

$$\sum_{t=1}^{T}\tilde{\ell}_t(x_t) - \tilde{\ell}_t(x^*) \leqslant 2D\max\left\{\frac{D^3}{\varepsilon^3}, 1\right\}\sum_{t=1}^{T}\frac{\alpha_t^2}{\alpha_t + 2\alpha_{1:t-1}} \tag{2.73}$$

for any $x^* \in [\varepsilon, \sqrt{D^2+\varepsilon^2}]$. Since $\sum_{t=1}^{T}\tilde{\ell}_t$ is minimized on $[\varepsilon, \sqrt{D^2+\varepsilon^2}]$, the above also holds for all $x^* > 0$. Therefore we have that

$$\begin{aligned}
\sum_{t=1}^{T}\ell_t(x_t) &\leqslant \sum_{t=1}^{T}\left(\frac{B_t^2+\varepsilon^2}{x_t} + x_t\right)\alpha_t \\
&= \sum_{t=1}^{T}\tilde{\ell}_t(x_t) \\
&\leqslant \min_{x^*>0} 2D\max\left\{\frac{D^3}{\varepsilon^3}, 1\right\}\sum_{t=1}^{T}\frac{\alpha_t^2}{\alpha_t + 2\alpha_{1:t-1}} + \sum_{t=1}^{T}\tilde{\ell}_t(x^*) \\
&= \min_{x^*>0} 2D\max\left\{\frac{D^3}{\varepsilon^3}, 1\right\}\sum_{t=1}^{T}\frac{\alpha_t^2}{\alpha_t + 2\alpha_{1:t-1}} + \sum_{t=1}^{T}\left(\frac{B_t^2+\varepsilon^2}{x^*} + x^*\right)\alpha_t \\
&= \min_{x^*>0}\frac{\varepsilon^2}{x^*}\alpha_{1:T} + 2D\max\left\{\frac{D^3}{\varepsilon^3}, 1\right\}\sum_{t=1}^{T}\frac{\alpha_t^2}{\alpha_t + 2\alpha_{1:t-1}} + \sum_{t=1}^{T}\ell_t(x^*)
\end{aligned} \tag{2.74}$$

Note that substituting $x^* = \sqrt{\frac{\sum_{t=1}^{T}\alpha_t\tilde{B}_t^2}{\alpha_{1:T}}}$ into the second-to-last line yields

$$\min_{x^*>0}\sum_{t=1}^{T}\left(\frac{B_t^2+\varepsilon^2}{x^*} + x^*\right)\alpha_t \leqslant 2\sqrt{\alpha_{1:T}\sum_{t=1}^{T}\alpha_t\tilde{B}_t^2} \leqslant 2\varepsilon\alpha_{1:T} + \min_{x^*>0}\sum_{t=1}^{T}\ell_t(x^*) \tag{2.75}$$

completing the proof. □

2.A.2 Adaptive and dynamic guarantees

Throughout Appendices 2.A.2, 2.A.3, and 2.A.4 we assume that $\arg\min_{\boldsymbol{\theta}\in\Theta}\sum_{\ell\in\mathcal{S}}\ell(\boldsymbol{\theta})$ returns a unique minimizer of the sum of the loss functions in the sequence $\mathcal{S}$. Formally, this can be defined to be the one minimizing an appropriate Bregman divergence $\mathcal{B}_R(\cdot|\boldsymbol{\phi}_R)$ from some fixed $\boldsymbol{\phi}_R \in \Theta$, e.g. the origin in Euclidean space or the uniform distribution over the simplex, which is unique by strong-convexity of $\mathcal{B}_R(\cdot|\boldsymbol{\phi}_R)$ and convexity of the set of optimizers of a convex function.

Theorem 2.A.2. Let each task $t \in [T]$ consist of a sequence of m_t convex loss functions $\ell_{t,i} : \Theta \mapsto \mathbb{R}$ that are $G_{t,i}$-Lipschitz w.r.t. $\|\cdot\|$. For $G_t^2 = G_{1:m_t}^2/m_t$ and $R : \Theta \mapsto \mathbb{R}$ a 1-strongly-convex function w.r.t. $\|\cdot\|$ define the following online algorithms:

1. `INIT`: a method that has dynamic regret bound $U_T^{\text{init}}(\Psi) \geqslant \sum_{t=1}^T f_t^{\text{init}}(\boldsymbol{\phi}_t) - f_t^{\text{init}}(\psi_t)$ w.r.t. reference actions $\Psi = \{\boldsymbol{\psi}_t\}_{t=1}^T \subset \Theta$ over the sequence $f_t^{\text{init}}(\cdot) = \mathcal{B}_R(\boldsymbol{\theta}_t^*||\cdot)G_t\sqrt{m_t}$.
2. `SIM`: a method that has (static) regret bound $U_T^{\text{sim}}(x)$ decreasing in $x > 0$ over the sequence of functions $f_t^{\text{sim}}(x) = \left(\frac{\mathcal{B}_R(\boldsymbol{\theta}_t^*||\boldsymbol{\phi}_t)}{x} + x\right) G_t\sqrt{m_t}$.

Then if Algorithm 1 sets $\phi_t = \texttt{INIT}(t)$ and $\eta_t = \frac{\texttt{SIM}(t)}{G_t\sqrt{m_t}}$ it will achieve

$$\overline{\text{Regret}} \leqslant \overline{U} \leqslant \frac{U_T^{\text{sim}}(V_\Psi)}{T} + \frac{1}{T}\min\left\{\frac{U_T^{\text{init}}(\Psi)}{V_\Psi}, 2\sqrt{U_T^{\text{init}}(\Psi)\sum_{t=1}^T G_t\sqrt{m_t}}\right\} + \frac{2V_\Psi}{T}\sum_{t=1}^T G_t\sqrt{m_t} \tag{2.76}$$

for $V_\Psi^2 = \frac{1}{\sum_{t=1}^T G_t\sqrt{m_t}}\sum_{t=1}^T \mathcal{B}_R(\boldsymbol{\theta}_t^*||\boldsymbol{\psi}_t)G_t\sqrt{m_t}$.

Proof. Letting $x_t = \texttt{SIM}(t)$ be the output of `SIM` at time t, defining $\sigma_t = G_t\sqrt{m_t}$ and $\sigma_{1:T} = \sum_{t=1}^T \sigma_t$, and substituting into the regret-upper-bound of OMD/FTRL (1.1), we have that

$$\begin{aligned}
\overline{U}T &= \sum_{t=1}^T \left(\frac{\mathcal{B}_R(\boldsymbol{\theta}_t^*||\boldsymbol{\phi}_t)}{x_t} + x_t\right)\sigma_t \\
&\leqslant \min_{x>0} U_T^{\text{sim}}(x) + \sum_{t=1}^T \left(\frac{\mathcal{B}_R(\boldsymbol{\theta}_t^*||\boldsymbol{\phi}_t)}{x} + x\right)\sigma_t \\
&\leqslant \min_{x>0} U_T^{\text{sim}}(x) + \frac{U_T^{\text{init}}(\Psi)}{x} + \sum_{t=1}^T \left(\frac{\mathcal{B}_R(\boldsymbol{\theta}_t^*||\boldsymbol{\psi}_t)}{x} + x\right)\sigma_t \\
&\leqslant U_T^{\text{sim}}(V_\Psi) + \min\left\{\frac{U_T^{\text{init}}(\Psi)}{V_\Psi}, 2\sqrt{U_T^{\text{init}}(\Psi)\sigma_{1:T}}\right\} + 2V_\Psi\sigma_{1:T}
\end{aligned} \tag{2.77}$$

where the last line follows by substituting $x = \max\left\{V_\Psi, \sqrt{\frac{U_T^{\text{init}}(\Psi)}{\sigma_{1:T}}}\right\}$. □

Corollary 2.A.2. Under the assumptions of Theorem 2.A.2 and boundedness of $\mathcal{B}_R$ over Θ, if `INIT` uses FTL, or AOGD in the case of $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$, and `SIM` uses ε-FTL as defined in

Proposition 2.A.2, then Algorithm 1 achieves

$$\overline{U}T \leqslant \min\left\{\frac{\varepsilon^2}{V}, \varepsilon\right\}\sigma_{1:T} + 2D\max\left\{\frac{D^3}{\varepsilon^3}, 1\right\}\sum_{t=1}^{T}\frac{\sigma_t^2}{\sigma_{1:t}} + \sqrt{8CD\sigma_{1:T}\sum_{t=1}^{T}\frac{\sigma_t^2}{\sigma_{1:t}}} + 2V\sigma_{1:T} \quad (2.78)$$

for $V^2 = \min_{\boldsymbol{\phi}\in\Theta}\sum_{t=1}^{T}\sigma_t\mathcal{B}_R(\boldsymbol{\theta}_t^*||\boldsymbol{\phi})$ and constant C the product of the constant C from Proposition 2.A.1 and the bound on the gradient of the Bregman divergence. Assuming $\sigma_t = G\sqrt{m}\ \forall\ t$ and substituting $\varepsilon = \frac{1}{\sqrt[5]{T}}$ yields

$$\overline{\text{Regret}} \leqslant \overline{U} = \tilde{\mathcal{O}}\left(\min\left\{\frac{1}{VT^{\frac{2}{5}}} + \frac{1}{\sqrt{T}}, \frac{1}{\sqrt[5]{T}}\right\} + V\right)\sqrt{m} \quad (2.79)$$

Proof. Substitute Propositions 2.A.1 and 2.A.2 into Theorem 2.A.2. □

Proposition 2.A.3. Let $\{\ell_t : \mathbb{R}_{>0} \mapsto \mathbb{R}\}_{t\geqslant 1}$ be a sequence of losses of form $\ell_t(x) = \left(\frac{B_t^2}{x} + x\right)\alpha_t$ for any positive scalars $\alpha_1, \ldots, \alpha_T \in \mathbb{R}_{>0}$ and adversarially chosen $B_t \in [0, D]$. Then the losses $\tilde{\ell}_t(x) = \left(\frac{B_t^2+\varepsilon^2}{x} + x\right)\alpha_t$ on the domain $[\varepsilon, \sqrt{D^2+\varepsilon^2}]$ are $\frac{\alpha_t D^2}{\varepsilon^2}$-Lipschitz and $\frac{2}{\alpha_t D}\min\left\{\frac{\varepsilon^2}{D^2}, 1\right\}$-exp-concave.

Proof. Lipschitzness follows by taking derivatives as in Proposition 2.A.2. Define $\tilde{B}_t^2 = B_t^2 + \varepsilon^2$. We then have

$$\partial_x\tilde{\ell}_t = \alpha_t\left(1 - \frac{\tilde{B}_t^2}{x^2}\right) \qquad \partial_{xx}\tilde{\ell}_t = \frac{2\alpha_t\tilde{B}_t^2}{x^3} \quad (2.80)$$

The γ-exp-concavity of the functions $\tilde{\ell}_t$ can be determined by finding the largest γ satisfying

$$\gamma \leqslant \frac{\partial_{xx}\tilde{\ell}_t}{(\partial_x\tilde{\ell}_t)^2} = \frac{2\tilde{B}_t^2 x}{\alpha_t(\tilde{B}_t^2 - x^2)^2} \quad (2.81)$$

for all $x \in [\varepsilon, \sqrt{D^2+\varepsilon^2}]$ and all $t \in [T]$. We first minimize jointly over choice of $x, \tilde{B}_t \in [\varepsilon, \sqrt{D^2+\varepsilon^2}]$. The derivatives of the objective w.r.t. x and $\tilde{B}_t$, respectively, are

$$\frac{2\tilde{B}_t^2(\tilde{B}_t^2 + 3x^2)}{(\tilde{B}_t^2 - x^2)^3} \qquad -\frac{4\tilde{B}_t x(\tilde{B}_t^2 + x^2)}{(\tilde{B}_t^2 - x^2)^3} \quad (2.82)$$

Note that the objective approaches ∞ as the coordinates approach the line $x = \tilde{B}_t$. For $x < \tilde{B}_t$ the derivative w.r.t. x is always positive while the derivative w.r.t. $\tilde{B}_t$ is always negative. Since we have the constraints $x \geqslant \varepsilon$ and $\tilde{B}_t^2 \leqslant D^2 + \varepsilon^2$, the optimum over $x < \tilde{B}_t$ is thus attained at $x = \varepsilon$ and $\tilde{B}_t^2 = D^2 + \varepsilon^2$. Substituting into the original objective yields

$$\frac{2(D^2+\varepsilon^2)\varepsilon}{\alpha_t D^4} \geqslant \frac{2\varepsilon}{\alpha_t D^2} \quad (2.83)$$

For $x > \tilde{B}_t$ the derivative w.r.t. x is always negative while the derivative w.r.t. $\tilde{B}_t$ is always positive. Since we have the constraints $x \leqslant \sqrt{D^2 + \varepsilon^2}$ and $\tilde{B}_t^2 \geqslant \varepsilon^2$, the optimum over $x > \tilde{B}_t$ is thus attained at $x = \sqrt{D^2 + \varepsilon^2}$ and $\tilde{B}_t^2 = \varepsilon^2$. Substituting into the original objective yields

$$\frac{2\varepsilon^2\sqrt{D^2 + \varepsilon^2}}{\alpha_t D^4} \geqslant \frac{2\varepsilon^2}{\alpha_t D^3} \tag{2.84}$$

Thus we have that the functions $\tilde{\ell}_t$ are $\frac{2}{\alpha_t D}\min\left\{\frac{\varepsilon^2}{D^2}, 1\right\}$-exp-concave. □

Corollary 2.A.3. Let $\{\ell_t : \mathbb{R}_{>0} \mapsto \mathbb{R}\}_{t\geqslant 1}$ be a sequence of functions of form $\ell_t(x) = \left(\frac{B_t^2}{x} + x\right)\alpha_t$ for any positive scalars $\alpha_1, \ldots, \alpha_T \in \mathbb{R}_{>0}$ and adversarially chosen $B_t \in [0, D]$. Then the ε-EWOO algorithm, which for $\varepsilon > 0$ uses the actions of EWOO run on the functions $\tilde{\ell}_t(x) = \left(\frac{B_t^2+\varepsilon^2}{x} + x\right)\alpha_t$ over the domain $[\varepsilon, \sqrt{D^2 + \varepsilon^2}]$ to determine x_t, achieves regret

$$\text{Regret}_t \leqslant \min\left\{\frac{\varepsilon^2}{x^*}, \varepsilon\right\}\alpha_{1:T} + \frac{D\alpha_{\max}}{2}\max\left\{\frac{D^2}{\varepsilon^2}, 1\right\}(1 + \log(T+1)) \tag{2.85}$$

for all $x^* > 0$.

Proof. Since $\sum_{t=1}^T \tilde{\ell}_t$ is minimized on $[\varepsilon, \sqrt{D^2 + \varepsilon^2}]$, we apply Theorem B.3.3 and follow a similar argument to that concluding Proposition 2.A.2 to get

$$\begin{aligned}\sum_{t=1}^T \ell_t(x_t) &\leqslant \frac{D\alpha_{\max}}{2}\max\left\{\frac{D^2}{\varepsilon^2}, 1\right\}(1 + \log(T+1)) + \sum_{t=1}^T \tilde{\ell}_t(x^*) \\ &= \min\left\{\frac{\varepsilon^2}{x^*}, \varepsilon\right\}\alpha_{1:T} + \frac{D\alpha_{\max}}{2}\max\left\{\frac{D^2}{\varepsilon^2}, 1\right\}(1 + \log(T+1)) + \sum_{t=1}^T \ell_t(x^*)\end{aligned} \tag{2.86}$$

□

Corollary 2.A.4. Under the assumptions of Theorem 2.A.2 and boundedness of $\mathcal{B}_R$ over Θ, if `INIT` uses FTL, or AOGD in the case of $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$, and `SIM` uses ε-EWOO as defined in Proposition 2.A.3, then Algorithm 1 achieves

$$\begin{aligned}\overline{U}T \leqslant \min\left\{\frac{\varepsilon^2}{V}, \varepsilon\right\}\sigma_{1:T} + \frac{D\sigma_{\max}}{2}\max\left\{\frac{D^2}{\varepsilon^2}, 1\right\}(1 + \log(T+1)) \\ + \sqrt{8CD\sigma_{1:T}\sum_{t=1}^T \frac{\sigma_t^2}{\sigma_{1:t}} + 2V\sigma_{1:T}}\end{aligned} \tag{2.87}$$

for $V^2 = \min_{\phi\in\Theta}\sum_{t=1}^T \sigma_t \mathcal{B}_R(\theta_t^* || \phi)$ and constant C the product of the constant C from Proposition 2.A.1 and the bound on the gradient of the Bregman divergence. Assuming $\sigma_t = G\sqrt{m}\ \forall\, t$ and substituting $\varepsilon = \frac{1}{\sqrt[4]{T}}$ yields

$$\overline{\text{Regret}} \leqslant \overline{U} = \tilde{\mathcal{O}}\left(\min\left\{\frac{1 + \frac{1}{V}}{\sqrt{T}}, \frac{1}{\sqrt[4]{T}}\right\} + V\right)\sqrt{m} \tag{2.88}$$

Proof. Substitute Proposition 2.A.1 and Corollary 2.A.3 into Theorem 2.A.2. □

Corollary 2.A.5. Under the assumptions of Theorem 2.2.1 and boundedness of Θ, if `INIT` is OGD with learning rate $\frac{1}{\sigma_{\max}}$ and `SIM` uses ε-EWOO as defined in Proposition 2.A.3 then Algorithm 1 achieves

$$\begin{aligned}\overline{U}T \leqslant \min\left\{\frac{\varepsilon^2}{V_\Psi}, \varepsilon\right\}\sigma_{1:T} + \frac{D\sigma_{\max}}{2}\max\left\{\frac{D^2}{\varepsilon^2}, 1\right\}(1+\log(T+1)) \\ + 2D\min\left\{\frac{D\sigma_{\max}}{V_\Psi}(1+P_\Psi), \sqrt{2\sigma_{\max}\sigma_{1:T}(1+P_\Psi)}\right\} + 2V_\Psi\sigma_{1:T}\end{aligned} \tag{2.89}$$

for $P_T(\Psi) = \sum_{t=2}^T \|\psi_t - \psi_{t-1}\|_2$. Assuming $\sigma_t = G\sqrt{m}\ \forall\ t$ and substituting $\varepsilon = \frac{1}{\sqrt[4]{T}}$ yields

$$\overline{\text{Regret}} \leqslant \overline{U} = \tilde{\mathcal{O}}\left(\min\left\{\frac{1+\frac{1}{V_\Psi}}{\sqrt{T}}, \frac{1}{\sqrt[4]{T}}\right\} + \min\left\{\frac{1+P_\Psi}{V_\Psi T}, \sqrt{\frac{1+P_\Psi}{T}}\right\} + V_\Psi\right)\sqrt{m} \tag{2.90}$$

Proof. Substitute Theorem 2.2.3 and Corollary 2.A.3 into Theorem 2.A.2. □

2.A.3 Guarantees for adapting to the inter-task geometry

For any nonnegative $\mathbf{a} \in \mathbb{R}^d$ we will use the notation $\|\cdot\|_{\mathbf{a}} = \langle\sqrt{\mathbf{a}}, \cdot\rangle$; note that if all elements of $\mathbf{a}$ are positive then $\|\cdot\|_{\mathbf{a}}$ is a norm on $\mathbb{R}^d$ with dual norm $\|\cdot\|_{\mathbf{a}^{-1}}$.

Claim 2.A.3. For $t \geqslant 1$ and $p \in (0,1)$ we have

$$\sum_{s=0}^{t-1}\frac{1}{(s+1)^p} \geqslant \sum_{s=1}^{t}\frac{1}{(s+1)^p} \geqslant \underline{c}_p t^{1-p} \qquad \text{and} \qquad \sum_{s=1}^{t}\frac{1}{s^p} \leqslant \overline{c}_p t^{1-p} \tag{2.91}$$

for $\underline{c}_p = \frac{1-\left(\frac{2}{3}\right)^{1-p}}{1-p}$ and $\overline{c}_p = \frac{1}{1-p}$.

Proof.

$$\sum_{s=0}^{t-1}\frac{1}{(s+1)^p} \geqslant \sum_{s=1}^{t}\frac{1}{(s+1)^p} \geqslant \int_1^{t+1}\frac{ds}{(s+1)^p} = \frac{(t+2)^{1-p}-2^{1-p}}{1-p} \geqslant \underline{c}_p(t+2)^{1-p} \geqslant \underline{c}_p t^{1-p} \tag{2.92}$$

$$\sum_{s=1}^{t}\frac{1}{s^p} \leqslant 1 + \int_1^t \frac{ds}{s^p} = 1 + \frac{t^{1-p}-1}{1-p} \leqslant \overline{c}_p t^{1-p} \tag{2.93}$$

□

Claim 2.A.4. For any $\mathbf{x} \in \mathbb{R}^d$ we have $\|\mathbf{x}^2\|_2^2 \leqslant \|\mathbf{x}\|_2^4$.

Proof.

$$\|\mathbf{x}^2\|_2^2 = \sum_{j=1}^d x_j^4 \leqslant \left(\sum_{j=1}^d x_j^2\right)^2 = \|\mathbf{x}\|_2^4 \tag{2.94}$$

□

We now review some facts from matrix analysis. Throughout this section we will use matrices in $\mathbb{R}^{d\times d}$; we denote the subset of symmetric matrices by $\mathbb{S}^d$, the subset of symmetric PSD matrices by $\mathbb{S}^d_+$, and the subset of symmetric positive-definite matrices by $\mathbb{S}^d_{++}$. Note that every symmetric matrix $\mathbf{A} \in \mathbb{S}^d$ has diagonalization $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ for diagonal matrix $\mathbf{\Lambda} \in \mathbb{S}^d$ containing the eigenvalues of $\mathbf{A}$ along the diagonal and a matrix $\mathbf{V} \in \mathbb{R}^{d\times d}$ of orthogonal eigenvectors. For such matrices we will use $\lambda_j(\mathbf{A})$ to denote the jth largest eigenvalue of $\mathbf{A}$ and for any function $f : [\lambda_d(\mathbf{A}), \lambda_1(\mathbf{A})] \mapsto \mathbb{R}$ we will use the notation

$$f(\mathbf{A}) = \mathbf{V}\begin{pmatrix} f(\mathbf{\Lambda}_{[1,1]}) & & \\ & \ddots & \\ & & f(\mathbf{\Lambda}_{[d,d]}) \end{pmatrix}\mathbf{V}^{-1} \tag{2.95}$$

Claim 2.A.5. [Boyd and Vandenberghe, 2004, Section A.4.1] $f(\mathbf{X}) = \log\det\mathbf{X}$ has gradient $\nabla_{\mathbf{X}} f = \mathbf{X}^{-1}$ over $\mathbb{S}^d_{++}$

Claim 2.A.6. [Moridomi et al., 2018, Theorem 3.1] The function $f(\mathbf{X}) = -\log\det\mathbf{X}$ is $\frac{1}{\sigma^2}$-strongly-convex w.r.t. $\|\!|\cdot|\!\|_\infty$ over the set of symmetric positive-definite matrices with spectral norm bounded by σ.

Definition 2.A.1. A function $f : (0,\infty) \mapsto \mathbb{R}$ is **operator convex** if $\forall\ \mathbf{X},\mathbf{Y} \in \mathbb{S}^d_{++}$ and any $t \in [0,1]$ we have

$$f(t\mathbf{X} + (1-t)\mathbf{Y}) \preceq tf(\mathbf{X}) + (1-t)f(\mathbf{Y}) \tag{2.96}$$

Claim 2.A.7. If $\mathbf{A} \in \mathbb{S}^d_+$ and $f : (0,\infty) \mapsto \mathbb{R}$ is operator convex then $\operatorname{Tr}(\mathbf{A}f(\mathbf{X}))$ is convex on $\mathbb{S}^d_{++}$.

Proof. Consider any $\mathbf{X},\mathbf{Y} \in \mathbb{S}^d_{++}$ and any $t \in [0,1]$. By the operator convexity of f, positive semi-definiteness of $\mathbf{A}$, and linearity of the trace functional we have that

$$\begin{aligned} 0 &\le \operatorname{Tr}(\mathbf{A}(tf(\mathbf{X}) + (1-t)f(\mathbf{Y}) - f(t\mathbf{X} + (1-t)\mathbf{Y}))) \\ &= t\operatorname{Tr}(\mathbf{A}(f(\mathbf{X}))) + (1-t)\operatorname{Tr}(\mathbf{A}f(\mathbf{Y})) - \operatorname{Tr}(\mathbf{A}(f(t\mathbf{X} + (1-t)\mathbf{Y}))) \end{aligned} \tag{2.97}$$

□

Corollary 2.A.6. If $\mathbf{A} \in \mathbb{S}^d_+$ then $\operatorname{Tr}(\mathbf{A}\mathbf{X}^{-1})$ and $\operatorname{Tr}(\mathbf{A}\mathbf{X})$ are convex over $\mathbb{S}^d_{++}$.

Proof. By the Löwner-Heinz theorem [Davis, 1963], x^{-1}, x, and x^2 are operator convex. The result follows by applying Claim 2.A.7. □

Corollary 2.A.7. [Lieb, 1973, Corollary 1.1] If $\mathbf{A},\mathbf{B} \in \mathbb{S}^d_+$ then $\operatorname{Tr}(\mathbf{A}\mathbf{X}\mathbf{B}\mathbf{X})$ is convex over $\mathbb{S}^d_+$.

Proposition 2.A.4. Let $\{\ell_t : \mathbb{R}_{>0} \mapsto \mathbb{R}\}_{t\geqslant 1}$ be of form $\ell_t(\mathbf{x}) = \left\|\frac{\mathbf{b}_t^2}{\mathbf{x}} + \mathbf{g}_t^2 \odot \mathbf{x}\right\|_1$ for adversarially chosen $\mathbf{b}_t, \mathbf{g}_t$ satisfying $\|\mathbf{b}_t\|_2 \leqslant D, \|\mathbf{g}_t\|_2 \leqslant G$. Then the (ε,ζ,p)-FTL algorithm, which for $\varepsilon,\zeta > 0$ and $p \in (0,\frac{2}{3})$ uses the actions of FTL run on the functions $\tilde{\ell}_t(\mathbf{x}) = \left\|\frac{\mathbf{b}_t^2 + \varepsilon_t^2\mathbf{1}_d}{\mathbf{x}} + (\mathbf{g}_t^2 + \zeta_t^2\mathbf{1}_d)\odot\mathbf{x}\right\|_1$, where $\varepsilon_t^2 = \varepsilon^2(t+1)^{-p}, \zeta_t^2 = \zeta^2(t+1)^{-p}$ for $t \geqslant 0$ and

$\mathbf{b}_0 = \mathbf{g}_0 = \mathbf{0}_d$, to determine $\mathbf{x}_t$, has regret

$$\begin{aligned}\text{Regret}_t \leqslant C_p \sum_{j=1}^{d} \min\left\{\left(\frac{\varepsilon^2}{\mathbf{x}_j^*} + \zeta^2 \mathbf{x}_j^*\right) T^{1-p}, \sqrt{\zeta^2 \mathbf{b}_{j,1:T}^2 + \varepsilon^2 \mathbf{g}_{j,1:T}^2} T^{\frac{1-p}{2}} + 2\varepsilon\zeta T^{1-p}\right\} \\ + C_p \left(\frac{D+\varepsilon}{\zeta^3} G^4 + \frac{G+\zeta}{\varepsilon^3} D^4\right) T^{\frac{3}{2}p} + C_p(D\zeta + G\varepsilon + \varepsilon\zeta)d\end{aligned} \tag{2.98}$$

for any $\mathbf{x} > 0$ and some constant C_p depending only on p.

Proof. Define $\tilde{\mathbf{b}}_t^2 = \mathbf{b}_t^2 + \varepsilon_t^2 \mathbf{1}_d, \tilde{\mathbf{g}}_t^2 = \mathbf{g}_t^2 + \zeta_t^2 \mathbf{1}_d$ and note that FTL run on the modified functions $\tilde{\ell}_t'(\mathbf{x}) = \left\| \frac{\tilde{\mathbf{g}}_t^2 \odot \mathbf{x}^2}{2} - \tilde{\mathbf{b}}_t^2 \odot \log(\mathbf{x}) \right\|_1$ plays the exact same actions $\mathbf{x}_t^2 = \frac{\tilde{\mathbf{b}}_{0:t-1}^2}{\tilde{\mathbf{g}}_{0:t-1}^2}$ as FTL run $\tilde{\ell}_t$. Since both sequences of loss functions are separable across coordinates, we consider d per-coordinate problems, with loss functions of form $\tilde{\ell}_t(x) = \frac{\tilde{b}_t^2}{x} + \tilde{g}_t^2 x$ and $\tilde{\ell}_t'(x) = \frac{\tilde{g}_t^2 x^2}{2} - \tilde{b}_t^2 \log x$. We have that

$$|\nabla_t| = \left|\tilde{g}_t^2 - \frac{\tilde{b}_t^2}{x_t^2}\right| = \frac{|\tilde{g}_t^2 x_t^2 - \tilde{b}_t^2|}{x_t^2} \quad |\nabla_t'| = \left|\tilde{g}_t^2 x_t - \frac{\tilde{b}_t^2}{x_t}\right| = \frac{|\tilde{g}_t^2 x_t^2 - \tilde{b}_t^2|}{x_t} \quad \partial_{xx}\tilde{\ell}_t' = \tilde{g}_t^2 + \frac{\tilde{b}_t^2}{x^2} \geqslant \tilde{g}_t^2 \tag{2.99}$$

so by Theorem 2.A.1 and substituting the action $x_t^2 = \frac{\tilde{b}_{0:t-1}^2}{\tilde{g}_{0:t-1}^2}$ we have per-coordinate regret

$$\begin{aligned}\sum_{t=1}^{T} \tilde{\ell}_t(x_t) - \tilde{\ell}_t(x^*) \leqslant 2\sum_{t=1}^{T} \frac{|\nabla_t||\nabla_t'|}{\tilde{g}_{1:t}^2} &= 2\sum_{t=1}^{T} \frac{|\tilde{g}_t^2 x_t^2 - \tilde{b}_t^2|^2}{x_t^3 \tilde{g}_{1:t}^2} \\ &\leqslant 2\sum_{t=1}^{T} \frac{\tilde{g}_t^4 x_t}{\tilde{g}_{1:t}^2} + \frac{\tilde{b}_t^4}{x_t^3 \tilde{g}_{1:t}^2} \\ &= 2\sum_{t=1}^{T} \frac{\tilde{g}_t^4 \sqrt{\tilde{b}_{0:t-1}^2}}{\tilde{g}_{1:t}^2 \sqrt{\tilde{g}_{0:t-1}^2}} + \frac{\tilde{b}_t^4}{\tilde{g}_{1:t}^2 \left(\frac{\tilde{b}_{0:t-1}^2}{\tilde{g}_{0:t-1}^2}\right)^{\frac{3}{2}}} \\ &\leqslant 2\sum_{t=1}^{T} \frac{\tilde{g}_t^4 \sqrt{\tilde{b}_{0:t-1}^2}}{\tilde{g}_{1:t}^2 \sqrt{\tilde{g}_{0:t-1}^2}} + \frac{\tilde{b}_t^4 \sqrt{2\tilde{g}_{1:t}^2}}{(\tilde{b}_{0:t-1}^2)^{\frac{3}{2}}} + \frac{\tilde{b}_t^4 \tilde{g}_0^3 \sqrt{2}}{\tilde{g}_{1:t}^2 (\tilde{b}_{0:t-1}^2)^{\frac{3}{2}}}\end{aligned} \tag{2.100}$$

Taking the summation over the coordinates yields

$$
\begin{aligned}
&\sum_{t=1}^{T}\tilde{\ell}_t(\mathbf{x}_t)-\tilde{\ell}_t(\mathbf{x}^*)\\
&\quad\leqslant 4\sum_{t=1}^{T}\left(\frac{(D+\varepsilon)(\|\mathbf{g}_t^2\|_2^2+\zeta_t^4 d)}{\zeta_{1:t}^2\sqrt{2\zeta_{0:t-1}^2}}+\frac{(G+\zeta)(\|\mathbf{b}_t^2\|_2^2+\varepsilon_t^4 d)}{(\varepsilon_{0:t-1}^2)^{\frac{3}{2}}}+\frac{(\|\mathbf{b}_t^2\|_2^2+\varepsilon_t^4 d)\zeta^3}{\tilde{\zeta}_{0:t-1}^2(\tilde{\varepsilon}_{0:t-1}^2)^{\frac{3}{2}}}\right)\sqrt{2t}\\
&\quad\leqslant 4\sum_{t=1}^{T}\left(\frac{(D+\varepsilon)(G^4+\zeta_t^4 d)}{(\underline{\mathfrak{c}}_p\zeta^2 t^{1-p})^{\frac{3}{2}}\sqrt{2}}+\frac{(G+\zeta)(D^4+\varepsilon_t^4 d)}{(\underline{\mathfrak{c}}_p\varepsilon^2 t^{1-p})^{\frac{3}{2}}}+\frac{(D^4+\varepsilon_t^4 d)\zeta}{\varepsilon^3(\underline{\mathfrak{c}}_p t^{1-p})^{\frac{5}{2}}}\right)\sqrt{2t}\\
&\quad\leqslant 4\sqrt{2}\frac{1+\frac{1}{\underline{\mathfrak{c}}_p}}{\underline{\mathfrak{c}}_p^{\frac{3}{2}}}\sum_{t=1}^{T}\left(\frac{D+\varepsilon}{\zeta^3}G^4+\frac{G+\zeta}{\varepsilon^3}D^4\right)t^{\frac{3}{2}p-1}+\frac{D\zeta+G\varepsilon+2\varepsilon\zeta}{t^{1+\frac{p}{2}}}d\\
&\quad\leqslant C_{p,1}\left(\frac{D+\varepsilon}{\zeta^3}G^4+\frac{G+\zeta}{\varepsilon^3}D^4\right)T^{\frac{3}{2}p}+C_{p,2}(D\zeta+G\varepsilon+2\varepsilon\zeta)d
\end{aligned}
\tag{2.101}
$$

for $C_{p,1}=4\overline{c}_{1-\frac{3}{2}p}\sqrt{2}\left(1+\frac{1}{\underline{\mathfrak{c}}_p}\right)/\underline{\mathfrak{c}}_p^{3/2}$ and $C_{p,2}=4\sqrt{2}\left(1+\frac{1}{\underline{\mathfrak{c}}_p}\right)\sum_{t=1}^{\infty}\frac{1}{t^{1+\frac{p}{2}}}/\underline{\mathfrak{c}}_p^{3/2}$. Thus we have

$$
\begin{aligned}
\sum_{t=1}^{T}\ell_t(\mathbf{x}_t)&\leqslant\sum_{t=1}^{T}\tilde{\ell}_t(\mathbf{x}_t)\\
&\leqslant\min_{\mathbf{x}^*>0}C_{p,1}\left(\frac{D+\varepsilon}{\zeta^3}G^4+\frac{G+\zeta}{\varepsilon^3}D^4\right)T^{\frac{3}{2}p}+C_{p,2}(D\zeta+G\varepsilon+2\varepsilon\zeta)d+\sum_{t=1}^{T}\tilde{\ell}_t(\mathbf{x}^*)\\
&=C_{p,1}\left(\frac{D+\varepsilon}{\zeta^3}G^4+\frac{G+\zeta}{\varepsilon^3}D^4\right)T^{\frac{3}{2}p}+C_{p,2}(D\zeta+G\varepsilon+2\varepsilon\zeta)d\\
&\quad+\min_{\mathbf{x}^*>0}\sum_{t=1}^{T}\left\|\frac{\mathbf{b}_t^2+\varepsilon_t^2\mathbf{1}_d}{\mathbf{x}^*}+(\mathbf{g}_t^2+\zeta_t^2\mathbf{1}_d)\odot\mathbf{x}^*\right\|_1\\
&\leqslant C_{p,1}\left(\frac{D+\varepsilon}{\zeta^3}G^4+\frac{G+\zeta}{\varepsilon^3}D^4\right)T^{\frac{3}{2}p}+C_{p,2}(D\zeta+G\varepsilon+2\varepsilon\zeta)d\\
&\quad\min_{\mathbf{x}^*>0}\overline{c}_pT^{1-p}\sum_{j=1}^{d}\frac{\varepsilon^2}{\mathbf{x}_j^*}+\zeta^2\mathbf{x}_j^*+\sum_{t=1}^{T}\ell_t(\mathbf{x}^*)
\end{aligned}
\tag{2.102}
$$

Separating again per-coordinate we have that

$$
\sum_{t=1}^{T}\frac{\tilde{b}_t^2}{x^*}+\tilde{g}_t^2x^*\leqslant\overline{c}_pT^{1-p}\frac{\varepsilon^2}{x^*}+\zeta^2x^*+\sum_{t=1}^{T}\ell_t(x^*)
\tag{2.103}
$$

However, substituting $x^* = \sqrt{\frac{\tilde{b}_{1:T}^2}{\tilde{g}_{1:T}}}$ also yields

$$\begin{aligned}\min_{x^*>0} \sum_{t=1}^{T} \frac{\tilde{b}_t^2}{x^*} + \tilde{g}_t^2 x^* &\leqslant 2\sqrt{\tilde{b}_{1:T}^2 \tilde{g}_{1:T}^2} \\ &\leqslant 2\sqrt{\overline{c}_p \left(\zeta^2 b_{1:T}^2 + \varepsilon^2 g_{1:T}^2\right)} T^{\frac{1-p}{2}} + 2\overline{c}_p \varepsilon \zeta T^{1-p} + \min_{x^*>0} \sum_{t=1}^{T} \ell_t(x^*)\end{aligned} \tag{2.104}$$

completing the proof. □

Theorem 2.A.3. Let Θ be a bounded convex subset of $\mathbb{R}^d$, let $\mathbb{D}_+^d$ be the set of positive definite diagonal matrices, and let each task $t \in [T]$ consist of a sequence of m convex Lipschitz loss functions $\ell_{t,i} : \Theta \mapsto \mathbb{R}$. Suppose for each task t we run the iteration in Equation 2.14 setting $\boldsymbol{\phi} = \frac{1}{t-1}\boldsymbol{\theta}_{1:t-1}^*$ and setting $\mathbf{H} = \operatorname{diag}(\boldsymbol{\eta}_t)$ via Equation 2.15 for $\varepsilon = 1, \zeta = \sqrt{m}$, and $p = \frac{2}{5}$. Then we achieve

$$\begin{aligned}\overline{\operatorname{Regret}} &\leqslant \overline{U} \\ &= \min_{\substack{\boldsymbol{\phi}\in\Theta \\ \mathbf{H}\in\mathbb{D}_+^d}} \tilde{\mathcal{O}}\left(\sum_{j=1}^{d} \min\left\{\frac{\frac{1}{\mathbf{H}_{[j,j]}} + \mathbf{H}_{[j,j]}}{T^{\frac{2}{5}}}, \frac{1}{\sqrt[5]{T}}\right\}\right)\sqrt{m} + \frac{1}{T}\sum_{t=1}^{T} \frac{\|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}\|_{\mathbf{H}^{-1}}^2}{2} + \sum_{i=1}^{m} \|\nabla_{t,i}\|_{\mathbf{H}}^2\end{aligned} \tag{2.105}$$

Proof. Define $\mathbf{b}_t^2 = \frac{1}{2}(\boldsymbol{\theta}_t^* - \boldsymbol{\phi}_t)^2$ and $\mathbf{g}_t^2 = \nabla_{1:m}^2$. Then applying Proposition 2.A.4 yields

$$\begin{aligned}\overline{U}T &= \sum_{t=1}^{T} \frac{\|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}_t\|_{\boldsymbol{\eta}_t^{-1}}^2}{2} + \sum_{i=1}^{m} \|\nabla_{t,i}\|_{\boldsymbol{\eta}_t}^2 \\ &= \sum_{t=1}^{T} \left\| \frac{(\boldsymbol{\theta}_t^* - \boldsymbol{\phi}_t)^2}{2\boldsymbol{\eta}_t} + \boldsymbol{\eta}_t \odot \nabla_{t,1:m}^2 \right\|_1 \\ &\leqslant \min_{\boldsymbol{\eta}\in\mathbb{R}_{>0}^d} \sum_{t=1}^{T} \left\| \frac{(\boldsymbol{\theta}_t^* - \boldsymbol{\phi}_t)^2}{2\boldsymbol{\eta}} + \boldsymbol{\eta} \odot \nabla_{t,1:m}^2 \right\|_1 \\ &\quad + C_p \sum_{j=1}^{d} \min\left\{\left(\frac{\varepsilon^2}{\boldsymbol{\eta}_{[j]}} + \zeta^2 \boldsymbol{\eta}_{[j]}\right) T^{1-p}, \sqrt{\zeta^2 \mathbf{b}_{1:T[j]}^2 + \varepsilon^2 \mathbf{g}_{1:T[j]}^2}\, T^{\frac{1-p}{2}} + 2\varepsilon\zeta T^{1-p}\right\} \\ &\quad + C_p\left(\frac{D+\varepsilon}{\zeta^3} G^4 m^2 + \frac{G\sqrt{m}+\zeta}{\varepsilon^3} D^4\right) T^{\frac{3}{2}p} + C_p(D\zeta + G\sqrt{m}\varepsilon + \varepsilon\zeta)d \\ &\leqslant \min_{\substack{\boldsymbol{\phi}\in\Theta \\ \boldsymbol{\eta}\in\mathbb{R}_{>0}^d}} \sum_{t=1}^{T} \frac{\|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}\|_{\boldsymbol{\eta}^{-1}}^2}{2} + \sum_{i=1}^{m_t} \|\nabla_{t,i}\|_{\boldsymbol{\eta}}^2 + \frac{D_\infty^2}{2}\|\boldsymbol{\eta}^{-1}\|_1(1+\log T) \\ &\quad + C_p \sum_{j=1}^{d} \min\left\{\left(\frac{\varepsilon^2}{\boldsymbol{\eta}_{[j]}} + \zeta^2 \boldsymbol{\eta}_{[j]}\right) T^{1-p}, \sqrt{\zeta^2 \mathbf{b}_{1:T[j]}^2 + \varepsilon^2 \mathbf{g}_{1:T[j]}^2}\, T^{\frac{1-p}{2}} + 2\varepsilon\zeta T^{1-p}\right\} \\ &\quad + C_p\left(\frac{D+\varepsilon}{\zeta^3} G^4 m^2 + \frac{G\sqrt{m}+\zeta}{\varepsilon^3} D^4\right) T^{\frac{3}{2}p} + C_p(D\zeta + G\sqrt{m}\varepsilon + \varepsilon\zeta)d\end{aligned} \tag{2.106}$$

Substituting $\boldsymbol{\eta} + \frac{\mathbf{1}_d}{\sqrt{mT}}$ for the optimum and the values of ε, ζ, p completes the proof. $\square$

Proposition 2.A.5. Let $\{\ell_t : \mathbb{R}_{>0} \mapsto \mathbb{R}\}_{t\geqslant 1}$ be of form $\ell_t(\mathbf{X}) = \operatorname{Tr}(\mathbf{X}^{-1}\mathbf{B}_t^2) + \operatorname{Tr}(\mathbf{X}\mathbf{G}_t^2)$ for adversarially chosen $\mathbf{B}_t, \mathbf{G}_t$ satisfying $\|\mathbf{B}_t\|_\infty \leqslant \sigma_B, \|\mathbf{G}_t\|_\infty \leqslant \sigma_G\sqrt{m}$ for $m \geqslant 1$. Then the (ε, ζ)-FTL algorithm, which for $\varepsilon, \zeta > 0$ uses the actions of FTL on the alternate function sequence $\tilde{\ell}_t(\mathbf{X}) = \operatorname{Tr}((\mathbf{B}^2 + \varepsilon^2\mathbf{I}_d)\mathbf{X}^{-1}) + \operatorname{Tr}((\mathbf{G}^2 + \zeta^2\mathbf{I}_d)\mathbf{X})$, achieves regret

$$\operatorname{Regret}_t \leqslant \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1 + \log T) + ((1 + \sigma_G^2)\varepsilon\sqrt{m} + (1 + \sigma_B^2)\zeta)T \tag{2.107}$$

for constant C_σ depending only on σ_B, σ_G.

Proof. Define $\tilde{\mathbf{B}}_t^2 = \mathbf{B}_t^2 + \varepsilon^2\mathbf{I}_d, \tilde{\mathbf{G}}_t^2 = \mathbf{G}_t^2 + \zeta^2\mathbf{I}_d$ and note that FTL run on modified functions $\tilde{\ell}'_t(\mathbf{X}) = \frac{1}{2}\operatorname{Tr}(\tilde{\mathbf{B}}_t^{-2}\mathbf{X}\tilde{\mathbf{G}}_t^2\mathbf{X}) - \log\det\mathbf{X}$ has the same solution $\tilde{\mathbf{B}}_{1:T}^2 = \mathbf{X}\tilde{\mathbf{G}}_{1:T}^2\mathbf{X}$.

$$\|\nabla_{\mathbf{X}}\tilde{\ell}_t(\mathbf{X})\|_\infty = \|\tilde{\mathbf{G}}_t^2 - \mathbf{X}^{-1}\tilde{\mathbf{B}}_t^2\mathbf{X}^{-1}\|_\infty \leqslant \|\tilde{\mathbf{G}}_t\|_\infty^2 + \|\mathbf{X}^{-1}\|_\infty^2\|\tilde{\mathbf{B}}_t\|_\infty^2 \leqslant \frac{\sigma_B^2}{\varepsilon^2} + m\sigma_G^2 + \zeta^2 \tag{2.108}$$

$$\begin{aligned}\|\nabla_{\mathbf{X}}\tilde{\ell}'_t(\mathbf{X})\|_\infty = \|\tilde{\mathbf{G}}_t^2\mathbf{X}\tilde{\mathbf{B}}_t^{-2} - \mathbf{X}^{-1}\|_\infty &\leqslant \|\tilde{\mathbf{G}}_t\|_\infty^2\|\mathbf{X}\|_\infty\|\tilde{\mathbf{B}}_t^{-1}\|_\infty^2 + \|\mathbf{X}^{-1}\|_\infty \\ &\leqslant \frac{(m\sigma_G^2 + \zeta^2)\sqrt{\sigma_B^2 + \varepsilon^2}}{\varepsilon^2\zeta} + \frac{\sqrt{m\sigma_G^2 + \zeta^2}}{\zeta}\end{aligned} \tag{2.109}$$

Since by Claim 2.A.6 $-\log\det|\mathbf{X}|$ is $\frac{\zeta^2}{\sigma_B^2+\varepsilon^2}$-strongly-convex we have by Theorem 2.A.1 that

$$\sum_{t=1}^T \tilde{\ell}_t(\mathbf{X}_t) - \tilde{\ell}_t(\mathbf{X}^*) \leqslant \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1 + \log T) \tag{2.110}$$

for some C_σ depending on σ_B^2, σ_G^2. Therefore

$$\begin{aligned}\sum_{t=1}^T \ell_t(\mathbf{X}) &\leqslant \sum_{t=1}^T \tilde{\ell}_t(\mathbf{X}) \\ &\leqslant \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1 + \log T) + \min_{\mathbf{X}>\mathbf{0}} \sum_{t=1}^T \tilde{\ell}_t(\mathbf{X}) \\ &\leqslant \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1 + \log T) + \min_{\mathbf{X}>\mathbf{0}} \varepsilon^2 T\operatorname{Tr}(\mathbf{X}^{-1}) + \zeta^2 T\operatorname{Tr}(\mathbf{X}) + \sum_{t=1}^T \ell_t(\mathbf{X}) \\ &\leqslant \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1 + \log T) + (1 + \sigma_G^2)\varepsilon T\sqrt{m} + \min_{\mathbf{X}>\mathbf{0}} \zeta^2 T\operatorname{Tr}(\mathbf{X}) + \sum_{t=1}^T \ell_t(\mathbf{X}) \\ &\leqslant \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1 + \log T) + ((1 + \sigma_G^2)\varepsilon\sqrt{m} + (1 + \sigma_B^2)\zeta)T + \min_{\mathbf{X}>\mathbf{0}} \sum_{t=1}^T \ell_t(\mathbf{X})\end{aligned} \tag{2.111}$$

$\square$

Theorem 2.A.4. Let Θ be a bounded convex subset of $\mathbb{R}^d$ and let each task $t \in [T]$ consist of a sequence of m convex Lipschitz loss functions $\ell_{t,i} : \Theta \mapsto \mathbb{R}$. Suppose for each task t we run the iteration in Equation 2.14 with $\boldsymbol{\phi} = \frac{1}{t-1}\boldsymbol{\theta}^*_{1:t-1}$ and $\mathbf{H}$ the unique positive definite solution of $\mathbf{B}_t^2 = \mathbf{H}\mathbf{G}_t^2\mathbf{H}$ for

$$\mathbf{B}_t^2 = t\varepsilon^2\mathbf{I}_d + \sum_{s<t}(\boldsymbol{\theta}^*_s - \boldsymbol{\phi}_s)(\boldsymbol{\theta}^*_s - \boldsymbol{\phi}_s)^\top \qquad \text{and} \qquad \mathbf{G}_t^2 = t\varepsilon^2\mathbf{I}_d + \sum_{s<t}\sum_{i=1}^m \nabla_{s,i}\nabla_{s,i}^\top \tag{2.112}$$

for $\varepsilon = 1/\sqrt[8]{T}$ and $\zeta = \sqrt{m}/\sqrt[8]{T}$. Then we achieve

$$\overline{\text{Regret}} \leqslant \overline{U} = \tilde{\mathcal{O}}\left(\frac{1}{\sqrt[8]{T}}\right)\sqrt{m} + \min_{\substack{\boldsymbol{\phi}\in\Theta \\ \mathbf{H}>\mathbf{0}}} \frac{2\lambda_1^2(\mathbf{H})}{\lambda_d(\mathbf{H})}\frac{1+\log T}{T} + \sum_{t=1}^T \frac{\|\boldsymbol{\theta}^*_t - \boldsymbol{\phi}^*\|^2_{\mathbf{H}^{-1}}}{2} + \sum_{i=1}^m \|\nabla_{t,i}\|^2_{\mathbf{H}} \tag{2.113}$$

Proof. Let D and G be the diameter of Θ and Lipschitz bound on the losses, respectively. Then applying Proposition 2.A.5 yields

$$\begin{aligned}
\overline{\text{Regret}}T &= \sum_{t=1}^T \frac{\|\boldsymbol{\theta}^*_t - \boldsymbol{\phi}_t\|^2_{\mathbf{H}_t^{-1}}}{2} + \sum_{i=1}^m \|\nabla_{t,i}\|^2_{\mathbf{H}_t} \\
&= \sum_{t=1}^T \frac{1}{2}\operatorname{Tr}\left(\mathbf{H}_t^{-1}(\boldsymbol{\theta}^*_t - \boldsymbol{\phi}_t)(\boldsymbol{\theta}^*_t - \boldsymbol{\phi}_t)^\top\right) + \operatorname{Tr}\left(\mathbf{H}_t\sum_{i=1}^m \nabla_{t,i}\nabla_{t,i}^\top\right) \\
&\leqslant \min_{\mathbf{H}>\mathbf{0}} \sum_{t=1}^T \frac{1}{2}\operatorname{Tr}\left(\mathbf{H}^{-1}(\boldsymbol{\theta}^*_t - \boldsymbol{\phi}_t)(\boldsymbol{\theta}^*_t - \boldsymbol{\phi}_t)^\top\right) + \operatorname{Tr}\left(\mathbf{H}\sum_{i=1}^m \nabla_{t,i}\nabla_{t,i}^\top\right) \\
&\quad + \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1+\log T) + ((1+G^2)\varepsilon\sqrt{m} + (1+D^2)\zeta)T \\
&= \min_{\mathbf{H}>\mathbf{0}} \sum_{t=1}^T \frac{\|\boldsymbol{\theta}^*_t - \boldsymbol{\phi}_t\|^2_{\mathbf{H}^{-1}}}{2} + \operatorname{Tr}\left(\mathbf{H}\sum_{i=1}^m \nabla_{t,i}\nabla_{t,i}^\top\right) \\
&\quad + \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1+\log T) + ((1+G^2)\varepsilon\sqrt{m} + (1+D^2)\zeta)T \\
&\leqslant \min_{\substack{\boldsymbol{\phi}\in\Theta \\ \mathbf{H}>\mathbf{0}}} \frac{2\lambda_1^2(\mathbf{H})}{\lambda_d(\mathbf{H})}\sum_{t=1}^T \frac{1}{t} + \sum_{t=1}^T \frac{\|\boldsymbol{\theta}^*_t - \boldsymbol{\phi}^*\|^2_{\mathbf{H}^{-1}}}{2} + \sum_{i=1}^m \|\nabla_{t,i}\|^2_{\mathbf{H}} \\
&\quad + \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1+\log T) + ((1+G^2)\varepsilon\sqrt{m} + (1+D^2)\zeta)T \\
&= \min_{\substack{\boldsymbol{\phi}\in\Theta \\ \mathbf{H}>\mathbf{0}}} \frac{2\lambda_1^2(\mathbf{H})}{\lambda_d(\mathbf{H})}\sum_{t=1}^T \frac{1}{t} + \sum_{t=1}^T \frac{\|\boldsymbol{\theta}^*_t - \boldsymbol{\phi}^*\|^2_{\mathbf{H}^{-1}}}{2} + \sum_{i=1}^m \|\boldsymbol{\nabla}_{t,i}\|^2_{\mathbf{H}} \\
&\quad + \frac{C_\sigma m^2}{\varepsilon^4\zeta^3}(1+\log T) + ((1+G^2)\varepsilon\sqrt{m} + (1+D^2)\zeta)T
\end{aligned} \tag{2.114}$$

□

2.A.4 Online-to-batch conversion for task-averaged regret

Theorem 2.A.5. Let $\mathcal{Q}$ be a distribution over distributions $\mathcal{P}$ over convex loss functions $\ell : \Theta \mapsto [0, 1]$. A sequence of sequences of loss functions $\{\ell_{t,i}\}_{t\in[T],i\in[m]}$ is generated by drawing m loss functions i.i.d. from each in a sequence of distributions $\{\mathcal{P}_t\}_{t\in[T]}$ themselves drawn i.i.d. from $\mathcal{Q}$. If such a sequence is given to an meta-learning algorithm with task-averaged regret $\overline{\mathrm{Regret}}$ that has states $\{s_t\}_{t\in[T]}$ at the beginning of each task t then we have w.p. $1-\delta$ for any $\boldsymbol{\theta}^* \in \Theta$ that

$$\mathbb{E}_{t\sim\mathcal{U}[T]}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{\overline{\mathrm{Regret}}}{m} + \sqrt{\frac{8}{T}\log\frac{1}{\delta}} \tag{2.115}$$

where $\bar{\boldsymbol{\theta}} = \frac{1}{m}\boldsymbol{\theta}_{1:m}$ is generated by randomly sampling $t \in \mathrm{Unif}[T]$, running the online algorithm with state s_t, and averaging the actions $\{\boldsymbol{\theta}_i\}_{i\in[m]}$. If on each task the meta-learning algorithm runs an online algorithm with regret upper bound $U(s_t)$ a convex, nonnegative, and $B\sqrt{m}$-bounded function of the state $s_t \in \mathcal{X}$, where $\mathcal{X}$ is a convex Euclidean subset, and the total regret-upper-bound is $\overline{U}$, then we also have the bound

$$\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{\overline{U}}{m} + B\sqrt{\frac{8}{mT}\log\frac{1}{\delta}} \tag{2.116}$$

where $\bar{\boldsymbol{\theta}} = \frac{1}{m}\boldsymbol{\theta}_{1:m}$ is generated by running the online algorithm with state $\bar{s} = \frac{1}{T}s_{1:T}$ and averaging the actions $\{\boldsymbol{\theta}_i\}_{i\in[m]}$.

Proof. For the second inequality, applying Proposition B.4.1, Jensen's inequality, and Proposition B.4.2 yields

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) &\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{U(\bar{s})}{m}\right) \\
&\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\frac{U(s_t)}{m}\right) \\
&= \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{2B}{T\sqrt{m}}\sum_{t=1}^{T}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\frac{U(s_t)}{2B\sqrt{m}} + \frac{\sqrt{m}}{2B}\right) - 1 \\
&\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{\overline{U}}{m} + B\sqrt{\frac{8}{mT}\log\frac{1}{\delta}}
\end{aligned} \tag{2.117}$$

The first inequality follows similarly except using Regret_t instead of U, linearity of expectation instead of Jensen's inequality, 1 instead of B, and $\overline{\mathrm{Regret}}$ instead of $\overline{U}$. □

Note that since regret-upper-bounds are nonnegative one can easily replace 8 by 2 in the second inequality by simply multiplying and dividing by $B\sqrt{m}$ in the third line of the above proof.

Claim 2.A.8. In the setup of Theorem 2.A.5, let $\boldsymbol{\theta}_t^* \in \arg\min_{\boldsymbol{\theta}\in\Theta}\sum_{i=1}^{m}\ell_{t,i}(\boldsymbol{\theta})$ and define the quantities $V_{\mathcal{Q}}^2 = \arg\min_{\boldsymbol{\phi}\in\Theta}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\|\boldsymbol{\theta}^* - \boldsymbol{\phi}\|_2^2$ and D the ℓ_2-radius of Θ. Then w.p. $1-\delta$ we have

$$V^2 = \min_{\boldsymbol{\phi}\in\Theta}\frac{1}{T}\sum_{t=1}^{T}\|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}\|_2^2 \leqslant \mathcal{O}\left(V_{\mathcal{Q}}^2 + \frac{D^2}{T}\log\frac{1}{\delta}\right) \tag{2.118}$$

Proof. Define $\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}\in\Theta} \sum_{t=1}^{T} \|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}\|_2^2$ and $\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}\in\Theta} \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\|\boldsymbol{\theta}^* - \boldsymbol{\phi}\|_2^2$. Then by a multiplicative Chernoff's inequality w.p. at least $1-\delta$ we have

$$\begin{aligned} TV^2 = \sum_{t=1}^{T} \|\boldsymbol{\theta}_t^* - \hat{\boldsymbol{\phi}}\|_2^2 &\leqslant \sum_{t=1}^{T} \|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}^*\|_2^2 \\ &\leqslant \left(1 + \max\left\{1, \frac{3D^2}{V_{\mathcal{Q}}^2 T}\log\frac{1}{\delta}\right\}\right) T\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\|\boldsymbol{\theta}^* - \boldsymbol{\phi}^*\|_2^2 \\ &\leqslant 2TV_{\mathcal{Q}}^2 + 3D^2\log\frac{1}{\delta} \end{aligned} \tag{2.119}$$

□

Corollary 2.A.8. Under the assumptions of Theorems 2.2.2 and 2.2.6, if the loss functions are Lipschitz and we use Algorithm 1 with η_t also learned, using ε-EWOO as in Theorem 2.2.2 for $\varepsilon = 1/\sqrt[4]{mT} + 1/\sqrt{m}$, and set the initialization using $\boldsymbol{\phi}_{t+1} = \frac{1}{t}\sum_{s\leqslant t}\boldsymbol{\theta}_s^*$, then w.p. $1-\delta$ we have

$$\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\ell_{\mathcal{P}}(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\ell_{\mathcal{P}}(\boldsymbol{\theta}^*) + \tilde{\mathcal{O}}\left(\frac{V_{\mathcal{Q}}}{\sqrt{m}} + \min\left\{\frac{\frac{1}{\sqrt{T}} + \frac{1}{\sqrt{m}}}{V_{\mathcal{Q}}m}, \frac{1}{\sqrt[4]{m^3T}} + \frac{1}{m}\right\} + \sqrt{\frac{1}{T}\log\frac{1}{\delta}}\right) \tag{2.120}$$

where $V_{\mathcal{Q}}^2 = \min_{\boldsymbol{\phi}\in\Theta}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\|\boldsymbol{\theta}^* - \boldsymbol{\phi}\|_2^2$.

Proof. Substitute Corollary 2.A.4 into Theorem 2.A.5 using the fact the the regret-upper-bounds are $\mathcal{O}(\frac{\sqrt{m}}{\varepsilon})$-bounded. Conclude by applying Claim 2.A.8. □

Theorem 2.A.6. Let $\mathcal{Q}$ be a distribution over distributions $\mathcal{P}$ over convex losses $\ell : \Theta \mapsto [0,1]$ s.t. the functions $\ell(\boldsymbol{\theta}) - \ell(\boldsymbol{\theta}^*)$ are ρ-self-bounded for some $\rho > 0$ and $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}\in\Theta}\mathbb{E}_{\ell\sim\mathcal{P}}(\boldsymbol{\theta})$. A sequence of sequences of loss functions $\{\ell_{t,i}\}_{t\in[T],i\in[m]}$ is generated by drawing m loss functions i.i.d. from each in a sequence of distributions $\{\mathcal{P}_t\}_{t\in[T]}$ themselves drawn i.i.d. from $\mathcal{Q}$. If such a sequence is given to an meta-learning algorithm with task-averaged regret $\overline{\mathrm{Regret}}$ that has states $\{s_t\}_{t\in[T]}$ at the beginning of each task t then we have w.p. $1-\delta$ for any $\boldsymbol{\theta}^* \in \Theta$ that

$$\begin{aligned} \mathbb{E}_{t\sim\mathcal{U}[T]}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{\overline{\mathrm{Regret}}}{m} + \sqrt{\frac{2\rho}{m}\left(\frac{\overline{\mathrm{Regret}}}{m} + \sqrt{\frac{8}{T}\log\frac{2}{\delta}}\right)\log\frac{2}{\delta}} \\ + \sqrt{\frac{8}{T}\log\frac{2}{\delta}} + \frac{3\rho+2}{m}\log\frac{2}{\delta} \end{aligned} \tag{2.121}$$

where $\bar{\boldsymbol{\theta}} = \frac{1}{m}\theta_{1:m}$ is generated by randomly sampling $t \in \mathcal{U}[T]$, running the online algorithm with state s_t, and averaging the actions $\{\boldsymbol{\theta}_i\}_{i\in[m]}$. If on each task the meta-learning algorithm runs an online algorithm with regret upper bound $U(s_t)$ a convex, nonnegative, and $B\sqrt{m}$-bounded function of the state $s_t \in \mathcal{X}$, where $\mathcal{X}$ is a convex Euclidean subset, and the total regret-upper-

bound is $\overline{U}$, then we also have the bound

$$\begin{aligned}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{\overline{U}}{m} + \sqrt{\frac{2\rho}{m}\left(\frac{\overline{U}}{m} + B\sqrt{\frac{8}{mT}\log\frac{2}{\delta}}\right)\log\frac{2}{\delta}} \\ + B\sqrt{\frac{8}{mT}\log\frac{2}{\delta}} + \frac{3\rho+2}{m}\log\frac{2}{\delta}\end{aligned} \tag{2.122}$$

where $\bar{\boldsymbol{\theta}} = \frac{1}{m}\boldsymbol{\theta}_{1:m}$ is generated by running the online algorithm with state $\bar{s} = \frac{1}{T}s_{1:T}$ and averaging the actions $\{\boldsymbol{\theta}_i\}_{i\in[m]}$.

Proof. By Corollary B.4.2 and Jensen's inequality we have w.p. $1-\frac{\delta}{2}$ that

$$\begin{aligned}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) &\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{U(\bar{s})}{m} + \frac{1}{m}\sqrt{2\rho U(\bar{s})\log\frac{1}{\delta}} + \frac{3\rho+2}{m}\log\frac{1}{\delta}\right) \\ &\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\frac{U(s_t)}{m}\right) \\ &\quad + \sqrt{\frac{2\rho}{mT}\sum_{t=1}^{T}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\frac{U(s_t)}{m}\right)\log\frac{2}{\delta}} + \frac{3\rho+2}{m}\log\frac{2}{\delta}\end{aligned} \tag{2.123}$$

As in the proof of Theorem 2.A.5, by Proposition B.4.2 we further have w.p. $1-\frac{\delta}{2}$ that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\frac{U(s_t)}{m}\right) \leqslant \frac{\overline{U}}{m} + B\sqrt{\frac{8}{mT}\log\frac{2}{\delta}} \tag{2.124}$$

Substituting the second inequality into the first yields the second bound. The first bound follows similarly except using Regret_t instead of U, linearity of expectation instead of Jensen's inequality, 1 instead of B, and $\overline{\mathrm{Regret}}$ instead of $\overline{U}$. □

Theorem 2.A.7. Let $\mathcal{Q}$ be a distribution over distributions $\mathcal{P}$ over convex loss functions $\ell : \Theta \mapsto [0,1]$. A sequence of sequences of loss functions $\{\ell_{t,i}\}_{t\in[T],i\in[m]}$ is generated by drawing m loss functions i.i.d. from each in a sequence of distributions $\{\mathcal{P}_t\}_{t\in[T]}$ themselves drawn i.i.d. from $\mathcal{Q}$. If such a sequence is given to an meta-learning algorithm that on each task runs an online algorithm with regret upper bound $U(s_t)$ a nonnegative, $B\sqrt{m}$-bounded, G-Lipschitz w.r.t. $\|\cdot\|$, and α-strongly-convex w.r.t. $\|\cdot\|$ function of the state $s_t \in \mathcal{X}$ at the beginning of each task t, where $\mathcal{X}$ is a convex Euclidean subset, and the total regret upper bound is $\overline{U}$, then we have w.p. $1-\delta$ for any $\boldsymbol{\theta}^* \in \Theta$ that

$$\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \mathcal{L}_T \tag{2.125}$$

for

$$\mathcal{L}_T = \frac{U^* + \overline{U}}{m} + \frac{4G}{T}\sqrt{\frac{\overline{U}}{\alpha m}\log\frac{8\log T}{\delta}} + \frac{\max\{16G^2, 6\alpha B\sqrt{m}\}}{\alpha m T}\log\frac{8\log T}{\delta} \tag{2.126}$$

where $U^* = \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}} U(s^*)$ for any valid s^* and $\bar{\boldsymbol{\theta}} = \frac{1}{m}\boldsymbol{\theta}_{1:m}$ is generated by running the online algorithm with state $\bar{s} = \frac{1}{T}s_{1:T}$ and averaging the actions $\{\boldsymbol{\theta}_i\}_{i\in[m]}$. If we further assume that the functions $\ell(\boldsymbol{\theta}) - \ell(\boldsymbol{\theta}^*)$ are ρ-self-bounded for some $\rho > 0$ and $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{\ell\sim\mathcal{P}}(\theta)$ for all $\mathcal{P}$ in the support of $\mathcal{Q}$ then we also have the bound

$$\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\theta^*) + \mathcal{L}_T + \sqrt{\frac{2\rho\mathcal{L}_T}{m}\log\frac{2}{\delta}} + \frac{3\rho+2}{m}\log\frac{2}{\delta} \tag{2.127}$$

Proof. Applying Proposition B.4.1 and Theorem B.4.1 we have w.p. $1 - \frac{\delta}{2}$ that

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\bar{\boldsymbol{\theta}}) &\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\left(\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{U(\bar{s})}{m}\right) \\
&\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\boldsymbol{\theta}^*) + \frac{1}{m}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}U(s^*) + \frac{\overline{U}}{m} \\
&\quad + \frac{4G}{T}\sqrt{\frac{\overline{U}}{\alpha m}\log\frac{8\log T}{\delta}} + \frac{\max\{16G^2, 6\alpha B\sqrt{m}\}}{\alpha m T}\log\frac{8\log T}{\delta} \\
&\leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\ell\sim\mathcal{P}}\ell(\theta^*) + \mathcal{L}_T
\end{aligned} \tag{2.128}$$

This yields the first bound since. The second bound follows similarly except for the application of Corollary B.4.2 in the second step w.p. $1 - \frac{\delta}{2}$. □

Corollary 2.A.9. Under the assumptions of Theorem 2.2.6 and boundedness of Θ, if the loss functions are G-Lipschitz and we use Algorithm 1 running OGD with fixed $\eta = \frac{V_{\mathcal{Q}}+1/\sqrt{T}}{G\sqrt{m}}$, where we have $V_{\mathcal{Q}}^2 = \min_{\boldsymbol{\phi}\in\Theta}\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\|\boldsymbol{\theta}^* - \boldsymbol{\phi}\|_2^2$, and set the initialization using $\boldsymbol{\phi}_{t+1} = \frac{1}{t}\boldsymbol{\theta}^*_{1:t}$, then w.p. $1-\delta$ we have

$$\mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\mathbb{E}_{\mathcal{P}^m}\ell_{\mathcal{P}}(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\mathcal{P}\sim\mathcal{Q}}\ell_{\mathcal{P}}(\boldsymbol{\theta}^*) + \tilde{\mathcal{O}}\left(\frac{V_{\mathcal{Q}}}{\sqrt{m}} + \left(\frac{1}{T} + \frac{1}{\sqrt{mT}}\right)\max\left\{\log\frac{1}{\delta}, \sqrt{\log\frac{1}{\delta}}\right\}\right) \tag{2.129}$$

Proof. Apply Theorem 2.A.2 with $V_\Phi = V_{\mathcal{Q}} + 1/\sqrt{T}$, $U^{\text{sim}} = 0$ (because the learning rate is fixed), and $U^{\text{init}} = \tilde{\mathcal{O}}\left(\hat{V}\sqrt{m} + 1/\sqrt{T}\right)$ (for $\hat{V}^2 = \min_{\boldsymbol{\phi}\in\Theta}\frac{1}{T}\sum_{t=1}^T\|\boldsymbol{\theta}_t^* - \boldsymbol{\phi}\|_2^2$). Substitute the result into Theorem 2.A.7 using the fact that U is $\mathcal{O}\left(\left(\frac{1}{\varepsilon}+\varepsilon\right)\sqrt{m}\right)$-bounded, $\mathcal{O}\left(\frac{\sqrt{m}}{\varepsilon}\right)$-Lipschitz, and $\Omega\left(\frac{\sqrt{m}}{\varepsilon}\right)$-strongly-convex. Conclude by applying Claim 2.A.8 to bound $\hat{V}$. □

2.A.5 Non-convex meta-learning

Proof of Theorem 2.3.1

Proof. The proof adapts the analysis of the exponential forecaster in Balcan et al. [2018b]. Let $W_t = \int_C w_t(\boldsymbol{\rho})d\boldsymbol{\rho}$ be the normalizing constant and $P_t = \mathbb{E}_{\boldsymbol{\rho}\sim p_t}[u_t(\boldsymbol{\rho})]$ be the expected payoff at round t. Also let $U_t(\boldsymbol{\rho}) = \sum_{j=1}^t u_j(\boldsymbol{\rho})$. We seek to bound $\text{Regret} = OPT - P(T)$, where

$OPT = U_T(\boldsymbol{\rho}^*)$ for optimal parameter $\boldsymbol{\rho}^*$ and $P(T) = \sum_{t=1}^{T} P_t$ is the expected utility of Algorithm 3 in T rounds. We will do this by lower bounding $P(T)$ and upper bounding OPT by analyzing the normalizing constant W_t.

Lower bound for $P(T)$: This follows from standard arguments, included for completeness. Using the definitions in Algorithm 3, it follows that

$$\frac{W_{t+1}}{W_t} = \frac{\int_{\mathcal{C}} e^{\lambda u_t(\boldsymbol{\rho})} w_t(\boldsymbol{\rho}) d\boldsymbol{\rho}}{W_t} = \int_{\mathcal{C}} e^{\lambda u_t(\boldsymbol{\rho})} \frac{w_t(\boldsymbol{\rho})}{W_t} d\boldsymbol{\rho} = \int_{\mathcal{C}} e^{\lambda u_t(\boldsymbol{\rho})} p_t(\boldsymbol{\rho}) d\boldsymbol{\rho} \tag{2.130}$$

Use inequalities $e^{\lambda x} \leqslant 1 + (e^{\lambda} - 1)x$ for $x \in [0, 1]$ and $1 + x \leqslant e^x$ to conclude

$$\frac{W_{t+1}}{W_t} \leqslant \int_{\mathcal{C}} p_t(\boldsymbol{\rho}) \left(1 + (e^{\lambda} - 1) u_t(\boldsymbol{\rho})\right) d\boldsymbol{\rho} = 1 + (e^{H\lambda} - 1) P_t \leqslant \exp\left((e^{\lambda} - 1) P_t\right) \tag{2.131}$$

Finally, we can write W_{T+1}/W_1 as a telescoping product to obtain

$$\frac{W_{T+1}}{W_1} = \prod_{t=1}^{T} \frac{W_{t+1}}{W_t} \leqslant \exp\left((e^{\lambda} - 1) \sum_t P_t\right) = \exp\left(P(T)(e^{\lambda} - 1)\right),$$

or, $W_{T+1} \leqslant \exp\left(P(T)(e^{\lambda} - 1)\right) \int_C w_1(\boldsymbol{\rho}) d\boldsymbol{\rho}$.

Upper bound for OPT: If there are at most k discontinuities in any ball of radius r, we can conclude that for all $\boldsymbol{\rho} \in \mathbb{B}^*(\boldsymbol{\rho}, r)$, $U_T(\boldsymbol{\rho}) \geqslant OPT - k - LTr$. Now, since $W_{T+1} = \int_C w_1(\boldsymbol{\rho}) \exp(\lambda U_T(\boldsymbol{\rho})) d\boldsymbol{\rho}$, we have

$$\begin{aligned} W_{T+1} \geqslant \int_{\mathbb{B}^*(\boldsymbol{\rho}, r)} w_1(\boldsymbol{\rho}) e^{\lambda U_T(\boldsymbol{\rho})} d\boldsymbol{\rho} &\geqslant \int_{\mathbb{B}^*(\boldsymbol{\rho}, r)} w_1(\boldsymbol{\rho}) e^{\lambda(OPT - k - LTr)} d\boldsymbol{\rho} \\ &= e^{\lambda(OPT - k - LTr)} \int_{\mathbb{B}^*(\boldsymbol{\rho}, r)} w_1(\boldsymbol{\rho}) d\boldsymbol{\rho} \end{aligned} \tag{2.132}$$

Putting together with the lower bound, and rearranging, gives

$$\begin{aligned} OPT - P_T &\leqslant \frac{P(T)(e^{\lambda} - 1 - \lambda)}{\lambda} + \frac{\log(1/Z)}{\lambda} + k + LTr \\ &\leqslant T\lambda + \frac{\log(1/Z)}{\lambda} + k + LTr \end{aligned} \tag{2.133}$$

where we use that $P(T) \leqslant T$ and for all $x \in [0, 1], e^x \leqslant 1 + x + (e - 2)x^2$. Take expectation over the sequence of utility functions and apply dispersion to conclude the result. □

Proof of Theorem 2.3.2

We extend the construction in Balcan et al. [2020b] to the multi-task setting. The main difference is that we generalize the construction for any task similarity, and show that we get the same lower bound asymptotically.

Proof. Define $u^{(b,x)}(\rho) = I[b=0] * I[\rho > x] + I[b=1] * I[\rho \leqslant x]$, where $b \in \{0,1\}$, $x, \rho \in [0,1]$ and $I[\cdot]$ is the indicator function. For each iteration the adversary picks $u^{(0,x)}$ or $u^{(1,x)}$ with equal probability for some $x \in [a, a+D^*]$, the ball of diameter D^* containing all the optima.

For each task t, $m - \frac{3}{D^*} m^{1-\beta}$ functions are presented with the discontinuity $x \in [a + D^*/3, a + 2D^*/3]$ while ensuring β-dispersion. The remaining $\frac{3}{D^*} m^{1-\beta}$ are presented with discontinuities located in successively halved intervals (the 'halving adversary') containing the optima in hindsight, any algorithm gets half of these wrong in expectation. It is readily verified that the functions are β-dispersed. The construction works provided m is sufficiently large ($m > \left(\frac{3}{D^*}\right)^{1/\beta}$). The task averaged regret is therefore also $\tilde{\Omega}(m^{1-\beta})$. □

Proof of Theorem 2.3.3

Proof.

$$
\begin{aligned}
&\sum_{t=1}^{T} \sum_{m=1}^{m} \ell_{t,i}(\boldsymbol{\rho}_{t,i}) - \min_{\boldsymbol{\rho}_t^* \in C} \sum_{i=1}^{m} \ell_{t,i}(\boldsymbol{\rho}_t^*) \\
&\quad \leqslant \sum_{t=1}^{T} U_t(w_t, v_t) \\
&\quad \leqslant \min_{v>0} H_T(v)\sqrt{m} + \sum_{t=1}^{T} \left(v + \frac{f_t(w_t)}{v} \right) \sqrt{m} + g(m) \qquad (2.134) \\
&\quad \leqslant \min_{w: C \mapsto \mathbb{R}_{\geqslant 0}, v>0} H_T(v)\sqrt{m} + \frac{F_T(w)\sqrt{m}}{v} + \sum_{t=1}^{T} \left(v + \frac{f_t(w)}{v} \right) \sqrt{m} + g(m) \\
&\quad \leqslant \left(H_T(V) + \min \left\{ \frac{F_T(w^*)}{V}, 2\sqrt{F_T(w^*)T} \right\} + 2TV \right) \sqrt{m} + Tg(m)
\end{aligned}
$$

where the last step is achieved by substituting $w = w^*$ and $v = \max\left\{V, \sqrt{F_T(w^*)/T}\right\}$. □

Proof of Lemma 2.3.1

Proof. Define a probability measure $p : C \mapsto \mathbb{R}_{\geqslant 0}$ that is constant on all elements $\tilde{D} \in \mathcal{D}_t$ of the discretization at time t, taking the value $p(\boldsymbol{\rho}) = \frac{1}{\mathrm{vol}(\tilde{D})} \sum_{D \in D_T, D \subset \tilde{D}} \mathbf{w}_{[D]} \;\forall\; \boldsymbol{\rho} \in \tilde{D}$. Note that for any $D \in \mathcal{D}_T$ that is a subset of $\tilde{D}$ we have that

$$
\mathbf{p}_{[D]} = \int_D \tilde{w}(\boldsymbol{\rho}) d\boldsymbol{\rho} = \frac{\mathbf{v}_{[D]}}{\sum_{D' \in \mathcal{D}_T, D' \subset \tilde{D}} \mathbf{v}_{[D']}} \sum_{D' \in \mathcal{D}_T, D' \subset \tilde{D}} \mathbf{w}_{[D']} \qquad (2.135)
$$

Then

$$
\begin{aligned}
&D_{\mathrm{KL}}(\mathbf{p}||\hat{\mathbf{v}}) - \eta \sum_{s\leqslant t} \log\langle \mathbf{w}_s^*, \mathbf{p}\rangle \\
&\quad= \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D\in\mathcal{D}_T, D\subset\tilde{D}} \mathbf{p}_{[D]} \log \frac{\mathbf{p}_{[D]}}{\hat{\mathbf{v}}_{[D]}} - \eta \sum_{s\leqslant t} \log \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D\in\mathcal{D}_T, D\subset\tilde{D}} \mathbf{w}^*_{s\,[D]} \mathbf{p}_{[D]} \\
&\quad= \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D\in\mathcal{D}_T, D\subset\tilde{D}} \frac{\mathbf{v}_{[D]}}{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{v}_{[D']}} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']} \log \frac{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']}}{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \hat{\mathbf{v}}_{[D']}} \\
&\qquad- \eta \sum_{s\leqslant t} \log \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D\in\mathcal{D}_T, D\subset\tilde{D}} \frac{\mathbf{w}^*_{s\,[D]} \mathbf{v}_{[D]}}{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{v}_{[D']}} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']} \\
&\quad\leqslant \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D\in\mathcal{D}_T, D\subset\tilde{D}} \frac{\mathbf{v}_{[D]}}{\sum_{D'\in\mathcal{D}_T, D'\in\tilde{D}} \mathbf{v}_{[D']}} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{D'} \log \frac{\mathbf{w}_{[D']}}{\hat{\mathbf{v}}_{[D']}} \\
&\qquad- \eta \sum_{s\leqslant t} \log \sum_{\tilde{D}\in\mathcal{D}_t, \tilde{D}\subset C_s} \sum_{D\in\mathcal{D}_T, D\subset\tilde{D}} \frac{\mathbf{v}_{[D]}}{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{v}_{[D']}} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']} \\
&\quad= \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D'\in\mathcal{D}_T, D'\in\tilde{D}} \mathbf{w}_{[D']} \log \frac{\mathbf{w}_{[D']}}{\hat{\mathbf{v}}_{[D']}} - \eta \sum_{s\leqslant t} \log \sum_{\tilde{D}\in\mathcal{D}_t, \tilde{D}\subset C_s} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']} \\
&\quad= D_{\mathrm{KL}}(\mathbf{w}||\hat{\mathbf{v}}) - \eta \sum_{s\leqslant t} \log\langle \mathbf{w}_s^*, \mathbf{w}\rangle
\end{aligned}
\tag{2.136}
$$

where the inequality follows from applying the log-sum inequality to the first term and the fact that $\mathbf{w}^*_{s\,[D]} = \mathbf{1}_{D\subset C_s}$ in the second term. Note that we also have

$$
\|\mathbf{p}\|_1 = \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D\in\mathcal{D}_T, D\subset\tilde{D}} \frac{\mathbf{v}_{[D]}}{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{v}_{[D']}} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']} = \sum_{\tilde{D}\in\mathcal{D}_t} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']} = 1 \tag{2.137}
$$

and

$$
\mathbf{p}_{[D]} = \frac{\mathbf{v}_{[D]}}{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{v}_{[D']}} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{w}_{[D']} \geqslant \frac{\gamma \mathbf{v}_{[D]}}{\sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \mathbf{v}_{[D']}} \sum_{D'\in\mathcal{D}_T, D'\subset\tilde{D}} \hat{\mathbf{v}}_{[D']} = \gamma \hat{\mathbf{v}}_{[D]} \tag{2.138}
$$

so $\mathbf{p}$ satisfies the optimization constraints. Therefore, since $\mathbf{w}$ was defined to be the minimum of the sum of the KL-divergence (a strongly-convex function [Shalev-Shwartz, 2011, Example 2.5]) and a convex function, it is unique and so coincides with $\mathbf{p}$.

On the other hand

$$\begin{aligned} D_{\mathrm{KL}}(\mathbf{p}(t)||\hat{\mathbf{v}}(\mathbf{t})) - \eta \sum_{s \leqslant t} \log\langle \mathbf{w}_s^*(t), \mathbf{p}(t)\rangle &\leqslant D_{\mathrm{KL}}(\mathbf{p}||\hat{\mathbf{v}}) - \eta \sum_{s \leqslant t} \log\langle \mathbf{w}_s^*, \mathbf{p}\rangle \\ &= D_{\mathrm{KL}}(\mathbf{w}||\hat{\mathbf{v}}) - \eta \sum_{s \leqslant t} \log\langle \mathbf{w}_s^*, \mathbf{w}\rangle \\ &\leqslant D_{\mathrm{KL}}(\tilde{\mathbf{w}}||\hat{\mathbf{v}}) - \eta \sum_{s \leqslant t} \log\langle \mathbf{w}_s^*, \tilde{\mathbf{w}}\rangle \\ &= D_{\mathrm{KL}}(\tilde{\mathbf{w}}(t)||\hat{\mathbf{v}}(t)) - \eta \sum_{s \leqslant t} \log\langle \mathbf{w}_s^*(t), \tilde{\mathbf{w}}(t)\rangle \end{aligned} \tag{2.139}$$

where the first inequality follows from above and the second from the optimality of $\mathbf{w}$. Note that by nonnegativity the discretization of p does not affect its measure over C, so $\|\mathbf{p}\|_1 = 1 \implies \|\mathbf{p}(t)\|_1 = 1$. Finally, also from above we have

$$\mathbf{p}(t)_{[D]} = \sum_{D' \in \mathcal{D}_T, D' \subset D} \mathbf{p}_{[D']} \geqslant \gamma \sum_{D' \in \mathcal{D}_T, D' \subset D} \mathbf{p}_{[D']} \hat{\mathbf{v}}_{[D']} = \gamma \hat{\mathbf{v}}(t)_{[D]} \tag{2.140}$$

Thus as before $\mathbf{p}(t)$ satisfies the optimization constraints, which with the previous inequality and the uniqueness of the optimum $\tilde{\mathbf{w}}(t)$ implies that $\mathbf{p}(t) = \tilde{\mathbf{w}}(t)$. Finally, since $\tilde{w}$ is constant on all elements of the discretization $\mathcal{D}_t$ of C this last fact implies that $\mathbf{p} = \tilde{\mathbf{w}}$, which together with $\mathbf{p} = \mathbf{w}$ implies the result. □

Lipschitzness for Algorithm 4

Claim 2.A.9. The loss f_t is $\frac{1}{\gamma \operatorname{vol}(C_t)}$-Lipschitz w.r.t. $\|\cdot\|_1$ over the set $\{\mathbf{w} \in \mathbb{R}^{|\mathcal{D}_T|} : \|\mathbf{w}\|_1 = 1, \mathbf{w} \geqslant \gamma \hat{\mathbf{v}}\}$.

Proof.

$$\max_{\|\mathbf{w}\|_1 = 1, \mathbf{w} \geqslant \gamma \hat{\mathbf{v}}} \|\nabla \log\langle \mathbf{w}_t^*, \mathbf{w}\rangle\|_\infty = \max_{D, \|\mathbf{w}\|_1 = 1, \mathbf{w} \geqslant \gamma \hat{\mathbf{v}}} \frac{\mathbf{w}_{t\,[D]}^*}{\langle \mathbf{w}_t^*, \mathbf{w}\rangle} \leqslant \frac{1}{\langle \mathbf{w}_t^*, \gamma \hat{\mathbf{v}}\rangle} = \frac{1}{\gamma \operatorname{vol}(C_t)} \tag{2.141}$$

□

Proof of Corollary 2.3.1

Proof. Using first-order conditions we have that the optimum in hindsight of the functions h_t satisfies

$$v^2 = \frac{1}{T} \sum_{t=1}^{T} f_t(w_t) = -\frac{1}{T} \sum_{t=1}^{T} \log\langle \mathbf{w}_t^*, \mathbf{w}_t\rangle \leqslant \frac{1}{T} \sum_{t=1}^{T} \log \frac{1}{\gamma \operatorname{vol}(C_t)} \tag{2.142}$$

Applying Corollary 2.A.3 with $\alpha_t = 1$, $B_t^2 = f_t(w_t)$, and $D^2 - \log \gamma$ instead of D^2 yields the result. □

Proof of Corollary 2.3.2

Proof. Using first-order conditions we have that the optimum in hindsight of the functions h_t satisfies

$$v^2 = \frac{1}{T}\sum_{t=1}^{T} f_t(w_t) = -\frac{1}{T}\sum_{t=1}^{T} \log\langle \mathbf{w}_t^*, \mathbf{w}_t\rangle \leqslant \frac{1}{T}\sum_{t=1}^{T} \log \frac{1}{\gamma \operatorname{vol}(C_t)} \tag{2.143}$$

Applying Proposition 2.A.2 with $\alpha_t = 1$, $B_t^2 = f_t(w_t)$, and $D^2 - \log\gamma$ instead of D^2 yields the result. □

Proof of Theorem 2.3.5

Proof. We have $F_T(w^*) = \tilde{\mathcal{O}}(\sqrt{BG}T^{\frac{3}{4}})$ and $H_T(V) = \tilde{\mathcal{O}}(\min\{1/V, \sqrt[5]{T}\}T^{\frac{3}{5}})$ from Corollaries 2.3.1 and 2.3.2. Substituting into Lemma 2.3.1 and simplifying yields

$$\tilde{\mathcal{O}}\left(\frac{\min\left\{\frac{1}{V}, \sqrt[4]{T}\right\}}{\sqrt{T}} + \min\left\{\frac{\sqrt{BG}}{V\sqrt[4]{T}}, \frac{\sqrt[4]{BG}}{\sqrt[8]{T}}\right\} + 2V\right)\sqrt{m} + g(m) \tag{2.144}$$

Simplifying further yields the result. □

Learning algorithmic parameters for combinatorial problems

We discuss implications of our results for several combinatorial problems of widespread interest including integer quadratic programming and auction mechanism design. We will need the following theorem from Balcan [2021], which generalizes the recipe for establishing dispersion given by Balcan et al. [2020a] for $d = 1, 2$ dimensions to arbitrary constant d dimendions. It is straightforward to apply the recipe to establish dispersion for these problems, which in turn implies that our meta-learning results are applicable. We demonstrate this for a few important problems below for completeness.

Theorem 2.A.8 (Balcan [2021])**.** Let $l_1, \ldots, l_m : \mathbb{R}^d \to \mathbb{R}$ be independent piecewise L-Lipschitz functions, each having discontinuities specified by a collection of at most K algebraic hypersurfaces of bounded degree. Let $\mathcal{L}$ denote the set of axis-aligned paths between pairs of points in $\mathbb{R}^d$, and for each $s \in \mathcal{L}$ define $D(m, s) = |\{1 \leqslant t \leqslant m \mid l_t \text{ has a discontinuity along } s\}|$. Then we have $\mathbb{E}[\sup_{s\in\mathcal{L}} D(m, s)] \leqslant \sup_{s\in\mathcal{L}} \mathbb{E}[D(m, s)] + \mathcal{O}(\sqrt{m\log(mK)})$.

Greedy knapsack We are given a knapsack with capacity C and items $i \in [m]$ with sizes w_i and values v_i. The goal is to select a subset S of items to add to the knapsack such that $\sum_{i\in S} w_i \leqslant C$ while maximizing the total value $\sum_{i\in S} v_i$ of selected items. We consider a general greedy heuristic to insert items with largest v_i/w_i^ρ first (due to Gupta and Roughgarden [2017]) for $\rho \in [0, 10]$.

The classic greedy heuristic sets $\rho = 1$ and can be used to provide a 2-approximation for the problem. However other values of ρ can improve the knapsack objective on certain problem instances. For example, for the value-weight pairs $\{(0.99, 1), (0.99, 1), (1.01, 1.01)\}$ and capacity $C = 2$ the classic heuristic $\rho = 1$ gives value 1.01 as the greedy heuristic is maximized for the

third item. However, using $\rho = 3$ (or any $\rho > 1 + \log(1/0.99)/\log(1.01) > 2.01$) allows us to pack the two smaller items giving the optimal value 1.98.

Our result (Theorem 2.3.5) when applied to this problem shows that it is possible to learn the optimal parameter values for the greedy heuristic algorithm family for knapsack from similar tasks.

Theorem 2.A.9. Consider instances of the knapsack problem given by bounded weights $w_{i,j} \in [1, C]$ and κ-bounded independent values $v_{i,j} \in [0, 1]$ for $i \in [m], j \in [T]$. Then the asymptotic task-averaged regret for learning the algorithm parameter ρ for the greedy heuristic family described above is $o_T(1) + 2V\sqrt{m} + \mathcal{O}(\sqrt{m})$.

Proof. Lemma 11 of Balcan et al. [2020a] shows that the loss functions form a $\frac{1}{2}$-dispersed sequence. The result follows by applying Theorem 2.3.5 with $\beta = \frac{1}{2}$. □

k-center clustering We consider the α-Lloyd's clustering algorithm family from Balcan et al. [2018c], where the initial k centers in the procedure are set by sampling points with probability proportional to d^α where d is the distance from the centers selected so far for some $\alpha \in [0, D], D \in \mathbb{R}_{\geqslant 0}$. For example, $\alpha = 0$ corresponds to the vanilla k-means with random initial centers, and $\alpha = 2$ setting is the k-means++ procedure. For this algorithm family, we are able to show the following guarantee. Interestingly, for this family it is sufficient to rely on the internal randomness of the algorithmic procedure and we do not need assumptions on data smoothness.

Theorem 2.A.10. Consider instances of the k-center clustering problem on n points, with Hamming loss $l_{i,j}$ for $i \in [m], j \in [T]$ against some (unknown) ground truth clustering. Then the asymptotic task-averaged regret for learning the algorithm parameter α for the α-Lloyd's clustering algorithm family of Balcan et al. [2018c] is $o_T(1) + 2V\sqrt{m} + \mathcal{O}(\sqrt{m})$.

Proof. We start by applying Theorem 4 from Balcan et al. [2018c] to an arbitrary α-interval $[\alpha_0, \alpha_0 + \epsilon] \subseteq [0, D]$ of length ϵ. The expected number of discontinuities (expectation under the internal randomness of the algorithm when sampling successive centers), is at most

$$\mathcal{O}(nk \log(n) \log(\max\{(\alpha_0 + \epsilon)/\alpha_0), (\alpha_0 + \epsilon) \log R\}) \tag{2.145}$$

where R is an upper bound on the ratio between any pair of non-zero distances. Considering cases $\alpha_0 \lesseqgtr \frac{1}{\log R}$ and using the inequality $\log(1 + x) \leqslant x$ for $x \geqslant 0$ we get that there are, in expectation, at most $\mathcal{O}(\epsilon nk \log n \log R)$ discontinuities in any interval of length ϵ. Theorem 2.A.8 now implies $\frac{1}{2}$-dispersion using the recipe from Balcan et al. [2020a]. The task-averaged regret bound follows from Theorem 2.3.5. □

Integer quadratic programming (IQP) The objective is to maximize a quadratic function $\mathbf{x}^\top \mathbf{A}\mathbf{x}$ for a matrix $\mathbf{A}$ with non-negative diagonal entries, subject to $\mathbf{x} \in \{0, 1\}^n$. In the classic Goemans-Williamson algorithm [Goemans and Williamson, 1995] one solves an SDP relaxation $\mathbf{U}^\top \mathbf{A}\mathbf{U}$ where columns $\mathbf{u}_i$ of U are unit vectors. The entries of $\mathbf{u}_i$ are then rounded to $\{\pm 1\}$ by projecting on a vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}_n, 1)$ and using $\operatorname{sign}(\langle \mathbf{u}_i, \mathbf{x} \rangle)$. A simple parametric family is s-linear rounding where the rounding is as before if $|\langle \mathbf{u}_i, \mathbf{x} \rangle| > s$ but uses probabilistic rounding

to round $\mathbf{u}_i$ to 1 with probability $\frac{1+(\langle \mathbf{u}_i, \mathbf{z} \rangle)/s}{2}$. The dispersion analysis of the problem from Balcan et al. [2018b] and the general recipe from Balcan et al. [2020a] imply that our results yield low task-averaged regret for learning the parameter of the s-linear rounding algorithms.

Theorem 2.A.11. Consider instances of IQP given by appropriate matrices $\mathbf{A}_{t,i}$ and rounding vectors $\mathbf{x}_{t,i} \sim \mathcal{N}(\mathbf{0}_n, 1)$ for $t \in [T], i \in [m]$. Then the asymptotic task-averaged regret for learning the algorithm parameter s for s-linear rounding is $o_T(1) + 2V\sqrt{m} + \mathcal{O}(\sqrt{m})$.

Proof. As noted in Balcan et al. [2018b], since $\mathbf{x}_{i,j}$ are normal, the local of discontinuities $s = |\langle \mathbf{u}_i, \mathbf{x} \rangle|$ are distributed with a $\sqrt{\frac{2}{\pi}}$-bounded density. Thus in any interval of length ϵ, we have in expectation at most $\epsilon\sqrt{\frac{2}{\pi}}$ discontinuities. Theorem 2.A.8 together with the general recipe from Balcan et al. [2020a] implies $\frac{1}{2}$-dispersion. The task-averaged regret bound is now a simple application of Theorem 2.3.5. □

Our results are an improvement over prior work which have only considered iid and (single-task) online learning settings. Similar improvements can be obtained for auction design, as described below.

Posted price mechanisms with additive valuations There are m items and n bidders with valuations $v_j(b_i), j \in [n], i \in [2^m]$ for all 2^m bundles of items. We consider additive valuations which satisfy $v_j(b) = \sum_{i \in b} v_j(\{i\})$. The objective is to maximize the social welfare (sum of buyer valuations). If the item values for each buyer have κ-bounded distributions, then the corresponding social welfare is dispersed and our results apply.

Theorem 2.A.12. Consider instances of posted price mechanism design problems with additive buyers and κ-bounded marginals of item valuations. Then the asymptotic task-averaged regret for learning the price which maximizes the social welfare is $o_T(1) + 2V\sqrt{m} + \mathcal{O}(\sqrt{m})$.

Proof. As noted in Balcan et al. [2018b], the locations of discontinuities are along axis-parallel hyperplanes (buyer j will be willing to buy item i at a price p_i if and only if $v_j(\{i\}) \geqslant p_i$, each buyer-item pair in each instance corresponds to a hyperplane). Thus in any pair of points p, p' (corresponding to pricing) at distance ϵ, we have in expectation at most $\epsilon\kappa mn$ discontinuities along any axis-aligned path joining p, p', since discontinuities for an item can only occur along axis-aligned segment for the axis corresponding to the item. Theorem 2.A.8 now implies $\frac{1}{2}$-dispersion. The task-averaged regret bound is now a simple application of Theorem 2.3.5. □

2.A.6 Structural results for bandits

Tuning the step-size

Lemma 2.A.3. Let $\ell_1, \ldots, \ell_T : \mathbb{R}_{>0} \mapsto \mathbb{R}_{>0}$ be a sequence of functions of form $\ell_t(x) = \frac{B_t^2}{x} + G^2 x$ for adversarially chosen $B_t \in [0, D]$ and some $G > 0$. Then for any $\rho \geqslant 0$, the actions of EWOO [Hazan et al., 2007, Figure 4] with parameter $\frac{2\rho^2}{DG}$ run on the modified losses $\frac{B_t^2 + \rho^2 D^2}{x} +$

$G^2 x$ over the domain $\left[\frac{\rho D}{G}, \frac{D}{G}\sqrt{1+\rho^2}\right]$ achieves regret w.r.t. any $x > 0$ of

$$\sum_{t=1}^{T} \ell_t(x) - \ell_t(x) \leqslant \min\left\{\frac{\rho^2 D^2}{x}, \rho D G\right\} T + \frac{DG(1+\log(T+1))}{2\rho^2} \tag{2.146}$$

Proof. By Proposition 2.A.3 the modified functions are $\frac{2\rho^2}{DG}$-exp-concave, so applying Corollary 2.A.3 with B_t set to $\frac{B_t}{G}$, D to $\frac{D}{G}$, $\alpha_t = G^2$, and $\varepsilon = \frac{\rho D}{G}$ yields the result. □

Lemma 2.A.4. For $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_T \in \partial\mathcal{K}$ consider a sequence of functions of form

$$U_t(\mathbf{x}, \eta) = \frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x})}{\eta} + \eta G^2 m \tag{2.147}$$

where $\mathcal{B}$ is the Bregman divergence of a strictly convex d.g.f. $\psi : \mathcal{K}^\circ \mapsto \mathbb{R}$ and where $\mathbf{x}_1 = \arg\min_{\mathbf{x}\in\mathcal{K}} \psi(\mathbf{x})$ defines the projection $\mathbf{c}_\varepsilon(\mathbf{x}) = \mathbf{x}_1 + \frac{\mathbf{x}-\mathbf{x}_1}{1+\varepsilon}$ for some $\varepsilon > 0$. Suppose we play $\mathbf{x}_{t+1} \leftarrow \mathbf{c}_\varepsilon\left(\frac{1}{t}\sum_{s=1}^{t} \hat{\mathbf{x}}_s\right)$ and set η_t using the actions of EWOO [Hazan et al., 2007, Figure 4] with parameter $\frac{2\rho^2}{DG}$ for some $\rho, D_\varepsilon > 0$ s.t. $\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x}) \leqslant D_\varepsilon^2 \ \forall\ \mathbf{x} \in \mathcal{K}_\varepsilon$ on the functions $\frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x}_t)+\rho^2 D_\varepsilon^2}{\eta} + \eta G^2 m$ over the domain $\left[\frac{\rho D_\varepsilon}{G\sqrt{m}}, \frac{D_\varepsilon}{G}\sqrt{\frac{1+\rho^2}{m}}\right]$, with η_1 being at the midpoint of the domain. Then $U_t(\mathbf{x}_t, \eta_t) \leqslant D_\varepsilon G\sqrt{m}\left(\frac{1}{\rho} + \sqrt{1+\rho^2}\right)\ \forall\ t \in [T]$ and

$$\begin{aligned}\sum_{t=1}^{T} U_t(\mathbf{x}_t, \eta_t) \leqslant \min_{\eta>0, \mathbf{x}\in\mathcal{K}} \sum_{t=1}^{T} \frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x})}{\eta} + \eta G^2 m \\ + \min\left\{\frac{\rho^2 D_\varepsilon^2}{\eta}, \rho D_\varepsilon G\right\} T + \frac{D_\varepsilon G(1+\log(T+1))}{2\rho^2} + \frac{8 S_\varepsilon K^2 (1+\log T)}{\eta}\end{aligned} \tag{2.148}$$

for $K = \max_{\mathbf{x}\in\mathcal{K}} \|\mathbf{x}\|_2$ and $S_\varepsilon = \max_{\mathbf{x}\in\mathcal{K}_\varepsilon} \|\nabla^2\psi(\mathbf{x})\|_2$.

Proof. The first claim follows by directly substituting the worst-case values of η into $U_t(\mathbf{x}, \eta)$. For the second, apply Lemma 2.A.3 followed by Corollary 2.A.1:

$$\begin{aligned}&\sum_{t=1}^{T} U_t(\mathbf{x}_t, \eta_t) \\ &\quad= \sum_{t=1}^{T} \frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x}_t)}{\eta_t} + \eta_t G^2 m \\ &\quad\leqslant \min_{\eta>0} \min\left\{\frac{\rho^2 D_\varepsilon^2}{\eta}, \rho D_\varepsilon G\right\} T + \frac{D_\varepsilon G(1+\log(T+1))}{2\rho^2} + \sum_{t=1}^{T} \frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x})}{\eta} + \eta G^2 m \\ &\quad\leqslant \min_{\eta>0} \min\left\{\frac{\rho^2 D_\varepsilon^2}{\eta}, \rho D_\varepsilon G\right\} T + \frac{D_\varepsilon G(1+\log(T+1))}{2\rho^2} + \frac{8 S_\varepsilon K^2 (1+\log T)}{\eta} \\ &\qquad+ \min_{\mathbf{x}\in\mathcal{K}_\varepsilon} \sum_{t=1}^{T} \frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)||\mathbf{x})}{\eta} + \eta G^2 m\end{aligned} \tag{2.149}$$

Conclude by noting that the sum of Bregman divergence to $\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)$ is minimized on their convex hull, a subset of $\mathcal{K}_\varepsilon$. □

Main structural result

Theorem 2.A.13. Consider a family of strictly convex functions $\psi_\theta : \mathcal{K}^\circ \mapsto \mathbb{R}$ parameterized by θ lying in an interval $\Theta \subset \mathbb{R}$ of radius R_Θ that are all minimized at the same $\mathbf{x}_1 \in \mathcal{K}^\circ$, and for $\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_T \in \partial\mathcal{K}$ consider a sequence of functions of form $U_t(\mathbf{x}, \eta, \theta)$ (2.36), as well as the associated regularized upper bounds $U_t^{(\rho)}$ (2.37). Define the maximum divergence $D = \max_{\theta\in\Theta} D_\theta$, radius $K = \max_{\mathbf{x}\in\mathcal{K}} \|\mathbf{x}\|_2$, and L_η the Lipschitz constant w.r.t. $\theta \in \Theta$ of $\frac{\hat{V}_\theta^2}{\eta} + \eta g(\theta)m + f(\theta)m$. Then Algorithm 6 with $\Theta_k \subset \Theta$ the uniform discretization of Θ s.t. $\max_{\theta\in\Theta} \min_{\theta'\in\Theta_k} |\theta - \theta'| \leqslant \frac{R_\Theta}{k}$, $\rho \in (0,1)$, $\underline{\eta}(\theta) = \frac{\rho D_\theta}{\sqrt{g(\theta)m}}$, $\overline{\eta}(\theta) = D_\theta\sqrt{\frac{1+\rho^2}{g(\theta)m}}$, $\alpha(\theta) = \frac{2\rho^2}{D_\theta\sqrt{g(\theta)m}}$, and $\lambda = \left(M\left(\frac{1}{\rho} + \sqrt{1+\rho^2}\right) + Fm\right)^{-1}\sqrt{\frac{\log k}{2T}}$ leads to a sequence $(\mathbf{x}_t, \eta_t(\theta_t), \theta_t)$ s.t. $\mathbb{E}\sum_{t=1}^T U_t(\mathbf{x}_t, \eta_t(\theta_t), \theta_t)$ is bounded by

$$\mathbb{E}\min_{\theta\in\Theta,\eta>0} \frac{8SK^2(1+\log T)}{\eta} + \left(\frac{\hat{V}_\theta^2}{\eta} + \eta g(\theta)m + f(\theta)m + \frac{L_\eta R_\Theta}{k} + \min\left\{\frac{\rho^2 D^2}{\eta}, \rho M\right\}\right)T + \left(\frac{4M}{\rho} + Fm\right)\sqrt{T\log k} + \frac{M(1+\log(T+1))}{2\rho^2} \tag{2.150}$$

and $\sum_{t=1}^T U_t(\mathbf{x}_t, \eta_t(\theta_t), \theta_t)$ is bounded w.p. $\geqslant 1 - \delta 1_{k>1}$ by

$$\min_{\theta\in\Theta,\eta>0} \frac{8SK^2(1+\log T)}{\eta} + \left(\frac{\hat{V}_\theta^2}{\eta} + \eta g(\theta)m + f(\theta)m + \frac{L_\eta R_\Theta}{k} + \min\left\{\frac{\rho^2 D^2}{\eta}, \rho M\right\}\right)T + \left(\frac{4M}{\rho} + Fm\right)\left(\sqrt{T\log k} + 1_{k>1}\sqrt{\frac{T}{2}\log\frac{1}{\delta}}\right) + \frac{M(1+\log(T+1))}{2\rho^2} \tag{2.151}$$

Proof. Formally, we have that

$$\begin{aligned}
&\mathbb{E}\sum_{t=1}^{T} U_t(\mathbf{x}_t, \eta_t(\theta_t), \theta_t) \\
&= \mathbb{E}\sum_{t=1}^{T} \frac{B_{\theta_t}(\mathbf{c}_{\theta_t}(\hat{\mathbf{x}}_t)||\mathbf{x}_t)}{\eta_t(\theta_t)} + \eta_t(\theta_t)g(\theta)m + f(\theta)m \\
&\leqslant \left(M\left(\frac{1}{\rho} + \sqrt{2}\right) + Fm\right)\sqrt{2T\log k} + \mathbb{E}\min_{\theta\in\Theta_k}\sum_{t=1}^{T}\frac{B_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)||\mathbf{x}_t)}{\eta_t(\theta)} + \eta_t(\theta)g(\theta)m + f(\theta)m \\
&\leqslant \left(\frac{4M}{\rho} + Fm\right)\sqrt{T\log k} + \mathbb{E}\min_{\theta\in\Theta_k,\eta>0,\mathbf{x}\in\mathcal{K}}\sum_{t=1}^{T}\frac{\mathcal{B}_\theta(\mathbf{c}_\theta(\hat{\mathbf{x}}_t)||\mathbf{x})}{\eta} + \eta g(\theta)m + f(\theta)m \\
&\qquad + \min\left\{\frac{\rho^2 D_\theta^2}{\eta}, \rho D_\theta\sqrt{g(\theta)m}\right\}T + \frac{D_\theta\sqrt{g(\theta)m}(1+\log(T+1))}{2\rho^2} + \frac{8SK^2(1+\log T)}{\eta}
\end{aligned} \tag{2.152}$$

where the first inequality is the regret of multiplicative weights with step-size λ [Shalev-Shwartz, 2011, Corollary 2.14] and the second is by applying Lemma 2.A.4 for each θ. We then simplify and apply the definition of $\hat{V}_\theta^2$ via Claim B.2.2 and conclude by applying Lipschitzness w.r.t. θ:

$$\begin{aligned}
&\mathbb{E}\sum_{t=1}^{T} U_t(\mathbf{x}_t, \eta_t(\theta_t), \theta_t) \\
&\leqslant \left(\frac{4M}{\rho} + Fm\right)\sqrt{T\log k} + \mathbb{E}\min_{\theta\in\Theta_k,\eta>0}\frac{\hat{V}_\theta^2 T}{\eta} + \eta g(\theta)mT + f(\theta)mT \\
&\qquad + \min\left\{\frac{\rho^2 D^2}{\eta}, \rho M\right\}T + \frac{M(1+\log(T+1))}{2\rho^2} + \frac{8SK^2(1+\log T)}{\eta} \\
&\leqslant \mathbb{E}\min_{\theta\in\Theta,\eta>0}\frac{8SK^2(1+\log T)}{\eta} + \left(\frac{\hat{V}_\theta^2}{\eta} + \eta g(\theta)m + f(\theta)m + \frac{L_\eta R_\Theta}{k} + \min\left\{\frac{\rho^2 D^2}{\eta}, \rho M\right\}\right)T \\
&\qquad + \left(\frac{4M}{\rho} + Fm\right)\sqrt{T\log k} + \frac{M(1+\log(T+1))}{2\rho^2}
\end{aligned} \tag{2.153}$$

The w.h.p. guarantee follows by Cesa-Bianchi and Lugosi [2006, Lemma 4.1]. □

2.A.7 Implicit exploration

Properties of the Tsallis entropy

Lemma 2.A.5. For any $\varepsilon \in (0,1]$ and $\mathbf{x} \in \triangle$ s.t. $\mathbf{x}_{[a]} \geqslant \frac{\varepsilon}{d}\ \forall\ a \in [d]$ the β-Tsallis entropy $H_\beta(\mathbf{x}) = -\frac{1-\sum_{a=1}^{d}\mathbf{x}_{[a]}^{\beta}}{1-\beta}$ is $d\log\frac{d}{\varepsilon}$-Lipschitz w.r.t. $\beta \in [0,1]$.

Proof. Let $\log_\beta x = \frac{x^{1-\beta}-1}{1-\beta}$ be the β-logarithm function and note that by Yamano [2002, Equation 6] we have $\log_\beta x - \log x = (1-\beta)(\partial_b \log_\beta x + \log_\beta x \log x) \geqslant 0 \; \forall \; \beta \in [0,1]$. Then we have for $\beta \in [0,1)$ that

$$\begin{aligned}
|\partial_\beta H_\beta(\mathbf{x})| &= \left| \frac{-H_\beta(\mathbf{x}) - \sum_{a=1}^d \mathbf{x}_{[a]}^\beta \log \mathbf{x}_{[a]}}{1-\beta} \right| \\
&= \frac{1}{1-\beta} \left| \sum_{a=1}^d \mathbf{x}_{[a]}^\beta (\log_\beta \mathbf{x}_{[a]} - \log \mathbf{x}_{[a]}) \right| \\
&= \frac{1}{1-\beta} \sum_{a=1}^d \mathbf{x}_{[a]}^\beta (\log_\beta \mathbf{x}_{[a]} - \log \mathbf{x}_{[a]}) \\
&\leqslant \frac{1}{1-\beta} \left(\sum_{a=1}^d \mathbf{x}_{[a]} \right)^\beta \left(\sum_{a=1}^d (\log_\beta \mathbf{x}_{[a]} - \log \mathbf{x}_{[a]})^{\frac{1}{1-\beta}} \right)^{1-\beta} \\
&\leqslant \frac{1}{1-\beta} \sum_{a=1}^d \log_\beta \mathbf{x}_{[a]} - \log \mathbf{x}_{[a]} \leqslant \frac{d}{1-\beta} (\log_\beta \frac{d}{\varepsilon} - \log \frac{d}{\varepsilon}) \leqslant -d \log \frac{d}{\varepsilon}
\end{aligned} \tag{2.154}$$

where the fourth inequality follows by Hölder's inequality, the fifth by subadditivity of x^a for $a \in (0,1]$, the sixth by the fact that $\partial_x(\log_\beta x - \log x) = x^{-\beta} - 1/x \leqslant 0 \; \forall \; \beta, x \in [0,1)$, and the last line by substituting $\beta = 0$ since $\partial_\beta \left(\frac{\log_\beta x - \log x}{1-\beta} \right) = \frac{2(x-x^\beta)-(1-\beta)(x^\beta+x)\log x}{x^\beta(1-\beta)^3} \leqslant 0 \; \forall \; \beta \in [0,1), x \in (0, 1/d]$. For $\beta = 1$, applying L'Hôpital's rule yields

$$\lim_{\beta \to 1} \partial_\beta H_\beta(\mathbf{x}) = -\frac{1}{2} \lim_{\beta \to 1} \sum_{a=1}^d \mathbf{x}_{[a]}^\beta \log^2 \mathbf{x}_{[a]} (1 - (1-\beta) \log \mathbf{x}_{[a]}) = -\frac{1}{2} \sum_{a=1}^d \mathbf{x}_{[a]} \log^2 \mathbf{x}_{[a]} \tag{2.155}$$

which is bounded on $[-2d/e^2, 0]$. □

Lemma 2.A.6. Consider $\mathbf{x}_1, \dots, \mathbf{x}_T \in \triangle$ s.t. $\mathbf{x}_t(a_t) = 1$ for some $a_t \in [d]$, and let $\bar{\mathbf{x}} = \frac{1}{T} \sum_{t=1}^T \mathbf{x}_t$ be their average. For any $\varepsilon \in (0,1]$ and $\beta \in (0,1]$ we have that for every $t \in [T]$

$$H_\beta(\bar{\mathbf{x}}^{(\varepsilon)}) - H_\beta(\mathbf{x}_t^{(\varepsilon)}) \leqslant H_\beta(\bar{\mathbf{x}}) \tag{2.156}$$

where recall that $\mathbf{x}^{(\varepsilon)} = \mathbf{c}_{\frac{\varepsilon}{1-\varepsilon}}(\mathbf{x}) = \mathbf{1}_d/d + (1-\varepsilon)(\mathbf{x} - \mathbf{1}_d/d) = (1-\varepsilon)\mathbf{x} + \frac{\varepsilon}{d}\mathbf{1}_d$.

Proof. Assume w.l.o.g. that $\bar{\mathbf{x}}_{[1]} \leqslant \bar{\mathbf{x}}_{[2]} \leqslant \ldots \leqslant \bar{\mathbf{x}}_{[d]}$ and $a_t = 1$, so that $\mathbf{x}_t^{(\varepsilon)} = \mathbf{e}_1^{(\varepsilon)}$. We take the

derivative

$$\begin{aligned}
&\partial_\varepsilon H_\beta\left((1-\varepsilon)\bar{\mathbf{x}}+\frac{\varepsilon}{d}\mathbf{1}_d\right)-\partial_\varepsilon H_\beta\left(\mathbf{e}_1^{(\varepsilon)}\right)\\
&\quad=\frac{d}{1-\beta}\sum_{a=1}^{d-1}\left(\frac{1}{((1-\varepsilon)\bar{\mathbf{x}}_{[a]}+\varepsilon/d)^{1-\beta}}-\frac{1}{(\varepsilon/d)^{1-\beta}}\right)\\
&\quad+\frac{d}{1-\beta}\sum_{a=1}^{d-1}\left(\frac{1}{((1-\varepsilon)+\varepsilon/d)^{1-\beta}}-\frac{1}{((1-\varepsilon)\bar{\mathbf{x}}_{[d]}+\varepsilon/d)^{1-\beta}}\right)\\
&\quad+\frac{d^2}{1-\beta}\sum_{a=1}^{d-1}\bar{\mathbf{x}}_{[a]}\left(\frac{1}{((1-\varepsilon)\bar{\mathbf{x}}_{[d]}+\varepsilon/d)^{1-\beta}}-\frac{1}{((1-\varepsilon)\bar{\mathbf{x}}_{[a]}+\varepsilon/d)^{1-\beta}}\right)
\end{aligned}\tag{2.157}$$

By the assumption that $\bar{\mathbf{x}}_{[a]}$ is non-decreasing in a, each of the summands above become non-positive. So for $\varepsilon\in(0,1]$ the derivative is non-positive, and for $\varepsilon\to 0^+$ it goes to $-\infty$. Thus the l.h.s. of the bound is monotonically non-increasing in ε for all $\varepsilon\in[0,1]$. The result then follows from the fact that for $\varepsilon=0$ we have $H_\beta\left((1-\varepsilon)\bar{\mathbf{x}}+\frac{\varepsilon}{d}\mathbf{1}_d\right)-H_\beta\left(\mathbf{e}_1^{(\varepsilon)}\right)=H_\beta(\bar{\mathbf{x}})$. □

Implicit exploration bounds

Lemma 2.A.7. Suppose we play $\texttt{OMD}_{\beta,\eta}$ with regularizer ψ_β the negative Tsallis entropy and initialization $\mathbf{x}_1\in\triangle$ on the sequence of linear loss functions $\ell_1,\dots,\ell_T\in[0,1]^d$. Then for any $\mathbf{x}\in\triangle$ we have

$$\sum_{t=1}^{T}\langle\boldsymbol{\ell}_t,\mathbf{x}_t-\mathbf{x}\rangle\leqslant\frac{\mathcal{B}_\beta(\mathbf{x}||\mathbf{x}_1)}{\eta}+\frac{\eta}{\beta}\sum_{a=1}^{d}\mathbf{x}_{t[a]}^{2-\beta}\ell_t^2(a)\tag{2.158}$$

Proof. Note that the following proof follows parts of the course notes by Luo [2017], which we reproduce for completeness. The OMD update at each step t involves the following two steps: set $\mathbf{y}_{t+1}\in\triangle$ s.t. $\nabla\psi_\beta(\mathbf{y}_{t+1})=\nabla\psi_\beta(\mathbf{x}_t)-\eta\ell_t$ and then set $\mathbf{x}_{t+1}=\arg\min_{\mathbf{x}\in\triangle}\mathcal{B}_\beta(\mathbf{x},\mathbf{y}_{t+1})$ [Hazan, 2015, Algorithm 14]. Note that by Hazan [2015, Equation 5.3] and nonnegativity of the Bregman divergence we have

$$\sum_{t=1}^{T}\langle\boldsymbol{\ell}_t,\mathbf{x}_t-\mathbf{x}\rangle\leqslant\frac{\mathcal{B}_\beta(\mathbf{x}||\mathbf{x}_1)}{\eta}+\frac{1}{\eta}\sum_{t=1}^{T}\mathcal{B}_\beta(\mathbf{x}_t||\mathbf{y}_{t+1})\tag{2.159}$$

To bound the second term, note that when ψ_β is the negative Tsallis entropy we have

$$\begin{aligned}
&\mathcal{B}_\beta(\mathbf{x}_t||\mathbf{y}_{t+1})\\
&\quad=\frac{1}{1-\beta}\sum_{a=1}^{d}\left(\mathbf{y}_{t+1[a]}^{\beta}-\mathbf{x}_{t[a]}^{\beta}+\frac{\beta}{\mathbf{y}_{t+1[a]}^{1-\beta}}(\mathbf{x}_{t[a]}-\mathbf{y}_{t+1[a]}\right)\\
&\quad=\frac{1}{1-\beta}\sum_{a=1}^{d}\left((1-\beta)\mathbf{y}_{t+1[a]}^{\beta}-\mathbf{x}_{t[a]}^{\beta}+\beta\left(\frac{1}{\mathbf{x}_{t[a]}^{1-\beta}}+\frac{1-\beta}{\beta}\eta\ell_t(a)\right)\mathbf{x}_{t[a]}\right)\\
&\quad=\sum_{a=1}^{d}\left(\mathbf{y}_{t+1[a]}^{\beta}-\mathbf{x}_{t[a]}^{\beta}+\eta\mathbf{x}_{t[a]}\ell_t(a)\right)
\end{aligned}\tag{2.160}$$

Plugging the following result, which follows from $(1+x)^\alpha \leqslant 1+\alpha x+\alpha(\alpha-1)x^2 \; \forall \, x \geqslant 0, \alpha < 0$, into the above yields the desired bound.

$$\begin{aligned}
\mathbf{y}_{t+1[a]}^{\beta} = \mathbf{x}_{t[a]}^{\beta} \left(\frac{\mathbf{y}_{t+1[a]}^{\beta-1}}{\mathbf{x}_{t[a]}^{\beta-1}} \right)^{\frac{\beta}{\beta-1}} &= \mathbf{x}_{t[a]}^{\beta} \left(1 + \frac{1-\beta}{\beta} \eta \mathbf{x}_{t[a]}^{1-\beta} \ell_t(a) \right)^{\frac{\beta}{\beta-1}} \\
&\leqslant \mathbf{x}_{t[a]}^{\beta} \left(1 - \eta \mathbf{x}_{t[a]}^{1-\beta} \ell_t(a) + \frac{\eta^2}{\beta} \mathbf{x}_{t[a]}^{2-2\beta} \ell_t(a)^2 \right) \\
&= \mathbf{x}_{t[a]}^{\beta} - \eta \mathbf{x}_{t[a]} \ell_t(a) + \frac{\eta^2}{\beta} \mathbf{x}_{t[a]}^{2-\beta} \ell_t(a)^2
\end{aligned} \tag{2.161}$$

□

Theorem 2.A.14. In Algorithm 6, let $\texttt{OMD}_{\eta,\beta}$ be online mirror descent with the Tsallis entropy regularizer ψ_β over γ-offset loss estimators, Θ_k is a subset of $[\underline{\beta}, \overline{\beta}] \subset [\frac{1}{\log d}, 1]$, and

$$U_t(\mathbf{x}, \eta, \beta) = \frac{\mathcal{B}_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x})}{\eta} + \frac{\eta d^\beta m}{\beta} \tag{2.162}$$

where $\hat{\mathbf{x}}_t^{(\varepsilon)} = (1-\varepsilon)\hat{\mathbf{x}}_t + \varepsilon \mathbf{1}_d/d$. Note that $U_t^{(\rho)}(\mathbf{x}, \eta, \beta) = U_t(\mathbf{x}, \eta, \beta) + \frac{\rho^2(d^{1-\beta}-1)}{\eta(1-\beta)}$. Then there exists settings of $\underline{\eta}, \overline{\eta}, \alpha, \lambda$ s.t. for all $\varepsilon, \rho, \gamma \in (0,1)$ we have w.p. $\geqslant 1-\delta$ that

$$\begin{aligned}
&\sum_{t=1}^{T} \sum_{i=1}^{m} \ell_{t,i}(a_{t,i}) - \ell_{t,i}(\mathring{a}_t) \\
&\leqslant (\varepsilon + \gamma d) mT + \frac{2 + \sqrt{\frac{d \log d}{em}}}{\gamma} \log \frac{5}{\delta} + \frac{8d\sqrt{m}}{\rho} \left(1_{k>1} \sqrt{T \log \frac{5k}{\delta}} + \frac{1 + \log(T+1)}{16\rho} \right) \\
&\quad + \min_{\beta \in [\underline{\beta}, \overline{\beta}], \eta > 0} \frac{8 \left(\frac{d}{\varepsilon}\right)^{2-\underline{\beta}} (1 + \log T)}{\eta} + \left(\frac{\hat{H}_\beta}{\eta} + \frac{\eta d^\beta m}{\beta} + \frac{L_\eta(\overline{\beta} - \underline{\beta})}{2k} + d \min \left\{ \frac{\rho^2}{2\eta}, \rho\sqrt{m} \right\} \right) T
\end{aligned} \tag{2.163}$$

for $L_\eta = \left(\frac{\log \frac{d}{\varepsilon}}{\eta} + \eta m \log^2 d \right) d$.

Proof. In this setting we have $g(\beta) = d^\beta/\beta$, $f(\beta) = 0$, $D_\beta^2 = \frac{d^{1-\beta}-1}{1-\beta}$, $D \leqslant \sqrt{d/2}$, $M = d\sqrt{m}$,

$F = 0$, $S = (d/\varepsilon)^{2-\underline{\beta}}$, and $K = 1$. We have that

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(a_{t,i}) - \ell_{t,i}(\mathring{a}_t) \\
&\quad= \sum_{t=1}^{T}\sum_{i=1}^{m} \langle \hat{\boldsymbol{\ell}}_{t,i}, \mathbf{x}_{t,i}\rangle - \ell_{t,i}(\mathring{a}_t) + \gamma \sum_{a=1}^{d} \hat{\ell}_{t,i}(a) \\
&\quad\leqslant \sum_{t=1}^{T} \frac{\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}_{t,1})}{\eta_t} + \sum_{i=1}^{m} \langle \hat{\boldsymbol{\ell}}_{t,i}, \hat{\mathbf{x}}_t^{(\varepsilon)}\rangle - \ell_{t,i}(\mathring{a}_t) + \frac{\eta_t}{\beta_t}\sum_{a=1}^{d} \mathbf{x}_{t,i[a]}^{2-\beta_t}\hat{\ell}_{t,i}^2(a) + \gamma\sum_{a=1}^{d}\hat{\ell}_{t,i}(a) \qquad (2.164) \\
&\quad\leqslant \varepsilon m T + \sum_{t=1}^{T} \frac{\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}_{t,1})}{\eta_t} + \sum_{i=1}^{m} \langle \hat{\boldsymbol{\ell}}_{t,i}, \hat{\mathbf{x}}_t^{(\varepsilon)}\rangle - \langle \boldsymbol{\ell}_{t,i}, \mathring{\mathbf{x}}_t^{(\varepsilon)}\rangle \\
&\qquad + \sum_{t=1}^{T}\frac{\eta_t}{\beta_t}\sum_{i=1}^{m}\sum_{a=1}^{d} \mathbf{x}_{t,i[a]}^{1-\beta_t}\hat{\ell}_{t,i}(a) + \gamma\sum_{a=1}^{d}\hat{\ell}_{t,i}(a)
\end{aligned}$$

where the equality follows similarly to Luo [2017] since $\langle \hat{\boldsymbol{\ell}}_{t,i}, \mathbf{x}_{t,i}\rangle = \ell_{t,i}(a_{t,i}) - \gamma\sum_{a=1}^{d}\hat{\ell}_{t,i}(a)$, the first inequality follows by Lemma 2.A.7 and the second by Hölder's inequality and the definitions of $\hat{\ell}_{t,i}$ and $\hat{\mathbf{x}}_{t,i}^{(\varepsilon)}$. We next apply the optimality of $\hat{a}_t$ for $\sum_{i=1}^{m}\hat{\ell}_{t,i}$ to get

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(a_{t,i}) - \ell_{t,i}(\mathring{a}_t) \\
&\quad\leqslant \varepsilon m T + \sum_{t=1}^{T} \frac{\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}_{t,1})}{\eta_t} + (1-\varepsilon)\sum_{i=1}^{m}\hat{\ell}_{t,i}(\mathring{a}_t) - \ell_{t,i}(\mathring{a}_t) + \frac{\varepsilon}{d}\sum_{a=1}^{d}\hat{\ell}_{t,i}(a) - \ell_{t,i}(a) \\
&\qquad + \sum_{t=1}^{T}\frac{\eta_t}{\beta_t}\sum_{i=1}^{m}\sum_{a=1}^{d} \mathbf{x}_{t,i[a]}^{1-\beta_t}\hat{\ell}_{t,i}(a) + \gamma\sum_{a=1}^{d}\hat{\ell}_{t,i}(a) \\
&\quad\leqslant \varepsilon m T + \frac{1 + \frac{\varepsilon}{d} + \frac{\overline{\eta}}{\underline{\beta}} + \gamma}{2\gamma}\log\frac{5}{\delta} + \sum_{t=1}^{T}\frac{\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}_{t,1})}{\eta_t} \\
&\qquad + \sum_{t=1}^{T}\frac{\eta_t}{\beta_t}\sum_{i=1}^{m}\sum_{a=1}^{d} \mathbf{x}_{t,i[a]}^{1-\beta_t}\ell_{t,i}(a) + \gamma\sum_{a=1}^{d}\ell_{t,i}(a) \\
&\quad\leqslant \varepsilon m T + \frac{2 + \sqrt{\frac{d\log d}{em}}}{\gamma}\log\frac{5}{\delta} + \gamma d m T + \sum_{t=1}^{T}\frac{\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}_{t,1})}{\eta_t} + \frac{\eta_t d^{\beta_t} m}{\beta_t}
\end{aligned} \qquad (2.165)$$

where the the second inequality follows by Neu [2015, Lemma 1] applied to each of the last four terms and the fifth by the definition of $\ell_{t,i}$ and using $\max_{\beta\in[\frac{1}{\log d},1]}\overline{\eta}(\beta) \leqslant \sqrt{\frac{d}{em\log d}}$. Substituting into Theorem 2.A.13 and simplifying yields the result except with $\hat{V}_\beta^2 = \frac{1}{T}\sum_{t=1}^{T}\psi_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}) - \psi_\beta(\hat{\bar{\mathbf{x}}}^{(\varepsilon)})$ in place of $\hat{H}_\beta$, but the former is bounded by the latter by Lemma 2.A.6. □

Corollary 2.A.10. Let $\underline{\beta} = \overline{\beta} = 1$. Then w.h.p. we can ensure task-averaged regret at most

$$2\sqrt{\hat{H}_1 dm} + \tilde{\mathcal{O}}\left(\frac{d\sqrt{m} + d^{\frac{2}{3}}m^{\frac{2}{3}}}{\sqrt[3]{T}}\right) \tag{2.166}$$

so long as $mT \geqslant d^2$ or alternatively ensure

$$\min\left\{2\sqrt{\hat{H}_1 dm} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{3}{4}}m^{\frac{3}{4}} + d\sqrt{m}}{\sqrt[4]{T}}\right), 2\sqrt{dm\log d} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{3}{2}}\sqrt{m}}{\sqrt{T}}\right)\right\} \tag{2.167}$$

so long as $mT \geqslant d$.

Proof. Applying Theorem 2.A.14, simplifying, and dividing by T yields task-averaged regret at most

$$\begin{aligned}(\varepsilon + \gamma d)m + \frac{2 + \sqrt{\frac{d\log d}{em}}}{\gamma T}\log\frac{5}{\delta} + \left(\frac{1 + \log(T+1)}{2\rho^2 T} + \min\left\{\frac{\rho^2}{\eta\sqrt{m}}, \rho\right\}\right)d\sqrt{m} \\ + \min_{\eta > 0}\frac{8d(1+\log T)}{\varepsilon\eta T} + \left(\frac{\hat{H}_1}{\eta} + \eta dm\right)\end{aligned} \tag{2.168}$$

Set $\gamma = \frac{1}{\sqrt{dmT}}$. Then set $\varepsilon = \sqrt[3]{\frac{d^2}{mT}}$ and $\rho = \frac{1}{\sqrt[3]{T}}$, and use $\eta = \sqrt{\frac{\hat{H}_1}{dm}} + \frac{1}{\sqrt[3]{dmT}}$ to get the first result. Otherwise, set $\varepsilon = \sqrt{\frac{d}{mT}}$ and $\rho = \frac{1}{\sqrt[4]{T}}$, and use the better of $\eta = \sqrt{\frac{\hat{H}_1}{dm}} + \frac{1}{\sqrt[4]{dmT}}$ and $\eta = \sqrt{\frac{\log d}{dm}}$ to get the second. □

Corollary 2.A.11. Let $\underline{\beta} = \frac{1}{2}$ and $\overline{\beta} = 1$ and assume $mT \geqslant d^{\frac{5}{2}}$. Then w.h.p. we can ensure task-averaged regret at most

$$\min_{\beta \in [\frac{1}{2}, 1]} 2\sqrt{\hat{H}_\beta d^\beta m/\beta} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{5}{7}}m^{\frac{5}{7}}}{T^{\frac{2}{7}}} + \frac{d\sqrt{m}}{\sqrt[4]{T}}\right) \tag{2.169}$$

using $k = \left\lceil \sqrt[4]{d}\sqrt{T} \right\rceil$.

Proof. Applying Theorem 2.A.14, simplifying, and dividing by T yields task-averaged regret at most

$$\begin{aligned}(\varepsilon + \gamma d)m + \frac{2 + \sqrt{\frac{d\log d}{em}}}{\gamma T}\log\frac{5}{\delta} + \frac{8d\sqrt{m}}{\rho}\left(\sqrt{\frac{\log\frac{5k}{\delta}}{T}} + \frac{1 + \log(T+1)}{16\rho T}\right) \\ + \min_{\beta \in [\underline{\beta}, \overline{\beta}], \eta > 0}\frac{8d^{\frac{3}{2}}(1+\log T)}{\varepsilon^{\frac{3}{2}}\eta T} + \left(\frac{\hat{H}_\beta}{\eta} + \frac{\eta d^\beta m}{\beta} + \frac{d}{4k}\left(\frac{\log\frac{d}{\varepsilon}}{\eta} + \eta m\log^2 d\right) + \rho d\sqrt{m}\right)\end{aligned} \tag{2.170}$$

Set $\gamma = \frac{1}{\sqrt{dmT}}$, $\varepsilon = \frac{d^{\frac{5}{7}}}{(mT)^{\frac{2}{7}}}$, $\rho = \frac{1}{\sqrt[4]{T}}$, and use $\eta = \sqrt{\frac{\beta\hat{H}_\beta}{md^\beta}} + \frac{1}{(dmT)^{\frac{2}{7}}}$ to get the result. □

Corollary 2.A.12. Let $\underline{\beta} = \frac{1}{\log d}$ and $\overline{\beta} = 1$ and assume $mT \geqslant d^3$. Then w.h.p. we can ensure task-averaged regret at most

$$\min_{\beta \in (0,1]} 2\sqrt{\hat{H}_\beta d^\beta m/\beta} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{3}{4}} m^{\frac{3}{4}} + d\sqrt{m}}{\sqrt[4]{T}}\right) \tag{2.171}$$

using $k = \left\lceil \sqrt[4]{d}\sqrt{T} \right\rceil$.

Proof. Applying Theorem 2.A.14, dividing by T, and simplifying yields

$$\begin{aligned}(\varepsilon + \gamma d)m + \frac{2 + \sqrt{\frac{d \log d}{em}}}{\gamma T} \log \frac{5}{\delta} + \frac{8d\sqrt{m}}{\rho}\left(\sqrt{\frac{\log \frac{5k}{\delta}}{T}} + \frac{1 + \log(T+1)}{16\rho T}\right) \\ + \min_{\beta \in [\underline{\beta}, \overline{\beta}], \eta > 0} \frac{8d^2(1 + \log T)}{\varepsilon^2 \eta T} + \left(\frac{\hat{H}_\beta}{\eta} + \frac{\eta d^\beta m}{\beta} + \frac{d}{2k}\left(\frac{\log \frac{d}{\varepsilon}}{\eta} + \eta \log^2 d\right) + \rho d\sqrt{m}\right)\end{aligned} \tag{2.172}$$

Note that $\hat{H}_\beta$ and $\frac{d^\beta}{\beta}$ are both decreasing on $\beta < \frac{1}{\log d}$, so β in the chosen interval is optimal over all $\beta \in (0, 1]$. Set $\gamma = \frac{1}{\sqrt{dmT}}$, $\varepsilon = \frac{d^{\frac{3}{4}}}{\sqrt[4]{mT}}$, $\rho = \frac{1}{\sqrt[4]{T}}$, and use $\eta = \sqrt{\frac{\beta \hat{H}_\beta}{md^\beta}} + \frac{1}{\sqrt[4]{dmT}}$ to get the result. □

2.A.8 Guaranteed exploration

Best-arm identification

Lemma 2.A.8. Suppose for $\varepsilon > 0$ we run OMD on task $t \in [T]$ with initialization $\mathbf{x}_{t,1} \in \triangle^{(\varepsilon)}$, regularizer $\psi_{\beta_t} + I_{\triangle^{(\varepsilon)}}$ for some $\beta_t \in (0, 1]$, and unbiased loss estimators ($\gamma = 0$). If Assumption 2.4.1 holds and $m > \frac{28d \log d}{3\varepsilon\Delta^2}$ then $\hat{\mathbf{x}}_t = \mathring{\mathbf{x}}_t$ w.p. $\geqslant 1 - d\kappa$, where $\kappa = \exp\left(-\frac{3\varepsilon\Delta^2 m}{28d}\right)$.

Proof. We extend the proof by Abbasi-Yadkori et al. [2018, Appendices B and F] to arbitrary lower bounds ε/d on the probability. First, since $0 \leqslant \hat{\ell}_{t,i}(a) \leqslant \frac{d}{\varepsilon}\ell_{t,i}(a)$ we have that

$$-\frac{d}{\varepsilon} \leqslant -1 \leqslant -\ell_{t,i}(a) \leqslant \hat{\ell}_{t,i}(a) - \ell_{t,i}(a) \leqslant \left(\frac{d}{\varepsilon} - 1\right)\ell_{t,i}(a) \leqslant \frac{d}{\varepsilon} \tag{2.173}$$

and so $|\hat{\ell}_{t,i}(a) - \ell_{t,i}(a)| \leqslant \frac{d}{\varepsilon}$. Therefore since the variance of the estimated losses is a scaled Bernoulli we have that

$$\mathrm{Var}(\hat{\ell}_{t,i}(a) - \ell_{t,i}(a)) = \mathrm{Var}(\hat{\ell}_{t,i}(a)) = \mathbf{x}_{t,i[a]}(1 - \mathbf{x}_{t,i[a]})\left(\frac{\ell_{t,i}(a)}{\mathbf{x}_{t,i[a]}}\right)^2 \leqslant \frac{\ell_{t,i}^2(a)}{\mathbf{x}_{t,i[a]}} \leqslant \frac{d}{\varepsilon} \tag{2.174}$$

We can thus apply a martingale concentration inequality of Fan et al. [2012, Corollary 2.1] to the

martingale difference sequence (MDS) $\frac{\varepsilon}{d}(\hat{\ell}_{t,i}(a) - \ell_{t,i}(a)) \in \left[-\frac{\varepsilon}{d}, 1\right]$ to obtain

$$\begin{aligned}
\Pr\left(\sum_{i=1}^{m} \hat{\ell}_{t,i}(a) - \ell_{t,i}(a) \geqslant \frac{m\Delta_a}{2}\right) &= \Pr\left(\frac{\varepsilon}{d}\sum_{i=1}^{m} \hat{\ell}_{t,i}(a) - \ell_{t,i}(a) \geqslant \frac{\varepsilon m\Delta_a}{2d}\right) \\
&\leqslant \Pr\left(\max_{j\in[m]} \frac{\varepsilon}{d}\sum_{i=j}^{m} \hat{\ell}_{t,i}(a) - \ell_{t,i}(a) \geqslant \frac{\varepsilon m\Delta_a}{2d}\right) \\
&\leqslant \exp\left(-\frac{2\left(\frac{\varepsilon m\Delta_a}{2d}\right)^2}{\min\left\{m(1+\varepsilon/d)^2, 4(\varepsilon m/d + \frac{\varepsilon m\Delta_a}{6})\right\}}\right) \\
&\leqslant \exp\left(-\frac{2\left(\frac{\varepsilon m\Delta_a}{2d}\right)^2}{4(\varepsilon m/d + \frac{\varepsilon m\Delta_a}{6})}\right) \\
&= \exp\left(-\frac{3\varepsilon m\Delta_a^2}{4d(6+\Delta_a)}\right) \\
&\leqslant \exp\left(-\frac{3\varepsilon m\Delta_a^2}{28d}\right)
\end{aligned} \tag{2.175}$$

where $\Delta_a = \frac{1}{m}\left|\sum_{i=1}^{m} \ell_{t,i}(a) - \min_{a'\neq a}\sum_{i=1}^{m} \ell_{t,i}(a')\right|$ is the per-arm loss gap in the last step we apply $\Delta_a \leqslant 1$. For the symmetric MDS $-\frac{\varepsilon}{d} \leqslant \ell_{t,i}(a) - \hat{\ell}_{t,i}(a) \leqslant 1$ we have

$$\begin{aligned}
\Pr\left(\sum_{i=1}^{m} \hat{\ell}_{t,i}(a) - \ell_{t,i}(a) \leqslant -\frac{m\Delta_a}{2}\right) &= \Pr\left(\sum_{i=1}^{m} \ell_{t,i}(a) - \hat{\ell}_{t,i}(a) \geqslant \frac{m\Delta_a}{2}\right) \\
&\leqslant \exp\left(-\frac{2\left(\frac{m\Delta_a}{2}\right)^2}{4\left(\frac{dm}{\varepsilon} + \frac{m\Delta_a}{6}\right)}\right) \\
&\leqslant \exp\left(-\frac{3\varepsilon m\Delta_a^2/d}{4(6+\varepsilon\Delta_a/d)}\right) \\
&\leqslant \exp\left(-\frac{3\varepsilon m\Delta_a^2}{28d}\right)
\end{aligned} \tag{2.176}$$

We can then conclude that

$$\begin{aligned}
&\Pr\left(\hat{\mathbf{x}}_t \neq \mathring{\mathbf{x}}_t\right) \\
&\quad\leqslant \Pr\left(\exists\, a \neq \mathring{a}_t : \sum_{i=1}^{m} \hat{\ell}_{t,i}(a) \leqslant \sum_{i=1}^{m} \ell_{t,i}(\mathring{a}_t)\right) \\
&\quad\leqslant \Pr\left(\sum_{i=1}^{m} \hat{\ell}_{t,i}(\mathring{a}_t) \geqslant \sum_{i=1}^{m} \ell_{t,i}(\mathring{a}_t) + \frac{m\Delta_{\mathring{a}_t}}{2} \;\vee\; \exists\, a \neq \mathring{a}_t : \sum_{i=1}^{m} \hat{\ell}_{t,i}(a) \leqslant \sum_{i=1}^{m} \ell_{t,i}(a) - \frac{m\Delta_a}{2}\right) \\
&\quad\leqslant \Pr\left(\sum_{i=1}^{m} \hat{\ell}_{t,i}(\mathring{a}_t) \geqslant \sum_{i=1}^{m} \ell_{t,i}(\mathring{a}_t) + \frac{m\Delta_{\mathring{a}_t}}{2}\right) + \sum_{a \neq \mathring{a}_t} \Pr\left(\sum_{i=1}^{m} \hat{\ell}_{t,i}(a) \leqslant \sum_{i=1}^{m} \ell_{t,i}(a) - \frac{m\Delta_a}{2}\right) \\
&\quad\leqslant \exp\left(-\frac{3\varepsilon m \Delta_{\mathring{a}_t}^2}{28d}\right) + \sum_{a \neq \mathring{a}_t} \exp\left(-\frac{3\varepsilon m \Delta_a^2}{28d}\right) \\
&\quad\leqslant d \exp\left(-\frac{3\varepsilon m \Delta^2}{28d}\right)
\end{aligned} \tag{2.177}$$

where the second-to-last line follows by substituting the bounds (2.175) and (2.176) into the left and right terms, respectively. □

Lemma 2.A.9. Suppose on each task $t \in [T]$ we run OMD as in Lemma 2.A.8. Then for any $\beta \in (0, 1]$ we have $\frac{1}{T}\mathbb{E}\sum_{t=1}^{T} \psi_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}) - \psi_\beta(\hat{\bar{\mathbf{x}}}^{(\varepsilon)}) \leqslant -\psi_\beta(\mathring{\bar{\mathbf{x}}}) + \frac{3d\kappa\beta}{1-\beta}\left(\left(\frac{d}{\varepsilon}\right)^{1-\beta} - 1\right)$.

Proof. We consider the expected divergence of the best initialization under the worst-case distribution of best arm estimation, which satisfies Lemma 2.A.8 and (2.177). We have by Claim B.2.2 and the mean-as-minimizer property of Bregman divergences that

$$\begin{aligned}
\frac{1}{T}\mathbb{E}\sum_{t=1}^{T} \psi_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}) - \psi_\beta(\hat{\bar{\mathbf{x}}}^{(\varepsilon)}) &= \mathbb{E} \min_{\mathbf{x} \in \triangle^{(\varepsilon)}} \frac{1}{T}\sum_{t=1}^{T} \mathcal{B}_\beta\left(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}\right) \\
&\leqslant \min_{\mathbf{x} \in \triangle^{(\varepsilon)}} \mathbb{E}\frac{1}{T}\sum_{t=1}^{T} \mathcal{B}_\beta\left(\hat{\mathbf{x}}_t^{(\varepsilon)} || \mathbf{x}\right) \\
&= \min_{\mathbf{x} \in \triangle^{(\varepsilon)}} \frac{1}{T}\sum_{t=1}^{T}\sum_{a=1}^{d} \mathbb{P}(a = \hat{a}_t)\mathcal{B}_\beta\left(\mathbf{e}_a^{(\varepsilon)} || \mathbf{x}\right) \\
&\leqslant \max_{\substack{\mathbf{p}_t \in \triangle, \forall t \in [T] \\ \mathbf{p}_{t[a]} \leqslant 2\kappa, \forall t \in [T], a \neq \mathring{a}_t \\ 1 - d\kappa \leqslant \mathbf{p}_{t[a]}, \forall t \in [T], a = \mathring{a}_t}} \min_{\mathbf{x} \in \triangle^{(\varepsilon)}} \frac{1}{T}\sum_{t=1}^{T}\sum_{a=1}^{d} \mathbf{p}_{t[a]}\mathcal{B}_\beta\left(\mathbf{e}_a^{(\varepsilon)} || \mathbf{x}\right)
\end{aligned} \tag{2.178}$$

To simplify the last expression, we define $\bar{\mathbf{p}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t$ and again apply the (weighted) mean-

as-minimizer property, followed by Claim B.2.2:

$$\begin{aligned}\min_{\mathbf{x}\in\triangle^{(\varepsilon)}} \frac{1}{T}\sum_{t=1}^{T}\sum_{a=1}^{d}\mathbf{p}_{t[a]}\mathcal{B}_\beta\left(\mathbf{e}_a^{(\varepsilon)}||\mathbf{x}\right) &= \min_{\mathbf{x}\in\triangle^{(\varepsilon)}} \sum_{a=1}^{d}\bar{\mathbf{p}}_{[a]}\mathcal{B}_\beta\left(\mathbf{e}_a^{(\varepsilon)}||\mathbf{x}\right) = \sum_{a=1}^{d}\mathcal{B}_\beta\left(\mathbf{e}_a^{(\varepsilon)}||\bar{\mathbf{p}}^{(\varepsilon)}\right)\\ &= \psi_\beta(\mathbf{e}_1^{(\varepsilon)}) - \psi_\beta(\bar{\mathbf{p}}^{(\varepsilon)})\end{aligned} \tag{2.179}$$

By substituting into the previous inequality, we can bound the expected divergence for the worst-case $\mathbf{p}_t$ as follows:

$$\begin{aligned}\frac{1}{T}\mathbb{E}\sum_{t=1}^{T}\psi_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}) - \psi_\beta(\hat{\bar{\mathbf{x}}}^{(\varepsilon)}) &\leqslant \psi_\beta\left(\mathbf{e}_1^{(\varepsilon)}\right) + \max_{\substack{\mathbf{p}_t\in\triangle,\forall t\in[T]\\ \mathbf{p}_{t[a]}\leqslant 2\kappa,\forall t\in[T],a\neq\mathring{a}_t\\ 1-d\kappa\leqslant\mathbf{p}_{t[a]},\forall t\in[T],a=\mathring{a}_t}} -\psi_\beta(\bar{\mathbf{p}}^{(\varepsilon)})\\ &\leqslant \psi_\beta\left(\mathbf{e}_1^{(\varepsilon)}\right) + \max_{\substack{\sum_{t=1}^{T}\sum_{a=1}^{d}\mathbf{p}_{t[a]}=T\\ \sum_{t=1}^{T}\mathbf{p}_{t[a]}\geqslant(1-d\kappa)\mathring{\bar{\mathbf{x}}}_{[a]}T,\forall a\\ \sum_{t=1}^{T}\mathbf{p}_{t[a]}\leqslant(2\kappa(1-\mathring{\bar{\mathbf{x}}}_{[a]})T+\mathring{\bar{\mathbf{x}}}_{[a]}T),\forall a}} -\psi_\beta(\bar{\mathbf{p}}^{(\varepsilon)})\\ &= \psi_\beta\left(\mathbf{e}_1^{(\varepsilon)}\right) - \min_{\substack{\bar{\mathbf{p}}\in\triangle\\ \bar{\mathbf{p}}_{[a]}\geqslant(1-d\kappa)\mathring{\bar{\mathbf{x}}}_{[a]},\forall a\\ \bar{\mathbf{p}}_{[a]}\leqslant 2\kappa+(1-2\kappa)\mathring{\bar{\mathbf{x}}}_{[a]},\forall a}} \psi_\beta(\bar{\mathbf{p}}^{(\varepsilon)})\end{aligned} \tag{2.180}$$

We use the shorthand $h(\mathbf{x}) = \psi_\beta\left((1-\varepsilon)\mathbf{x} + \frac{\varepsilon}{d}\mathbf{1}_d\right)$. We have

$$\begin{aligned}-\partial_{\mathbf{x}_{[a]}}\left(\psi_\beta(\mathbf{x})\right) &= \partial_{\mathbf{x}_{[a]}}\left(\frac{1}{(1-\beta)}\left(\sum_{b=1}^{d}\mathbf{x}_{[b]}^{\beta} - 1\right)\right)\\ &= \partial_{\mathbf{x}_{[a]}}\left(\frac{1}{(1-\beta)}\left(\sum_{b=1}^{d}\mathbf{x}_{[b]}^{\beta} + \beta d^{1-\beta}(1-\sum_{b=1}^{d}\mathbf{x}_{[b]}) - 1\right)\right)\\ &= \frac{\beta}{1-\beta}\cdot\left(\mathbf{x}_{[a]}^{\beta-1} - d^{1-\beta}\right)\end{aligned} \tag{2.181}$$

and therefore

$$\begin{aligned}\|\nabla h(\mathbf{x})\|_\infty &= \max_{a=1,\ldots,d}\left|\partial_{\mathbf{x}_{[a]}}\psi_\beta\left((1-\varepsilon)\mathbf{x} + \frac{\varepsilon}{d}\mathbf{1}_d\right)\right|\\ &\leqslant \frac{\beta}{1-\beta}\max_{a=1,\ldots,d}\left|((1-\varepsilon)\mathbf{x}_{[a]} + \varepsilon/d)^{\beta-1} - d^{1-\beta}\right|\\ &\leqslant \frac{\beta}{1-\beta}\left(\left(\frac{d}{\varepsilon}\right)^{1-\beta} - 1\right) = \beta\log_\beta\left(\frac{d}{\varepsilon}\right)\end{aligned} \tag{2.182}$$

Finally, by convexity of h we have

$$\begin{aligned}\min_{\substack{\bar{\mathbf{p}}\in\triangle\\ \bar{\mathbf{p}}_{[a]}\geqslant(1-d\kappa)\mathring{\bar{\mathbf{x}}}_{[a]},\forall a\\ \bar{\mathbf{p}}_{[a]}\leqslant 2\kappa+(1-2\kappa)\mathring{\bar{\mathbf{x}}}_{[a]},\forall a}} h(\bar{\mathbf{p}}) &\geqslant h(\mathring{\bar{\mathbf{x}}}) - \|\nabla h(\mathring{\bar{\mathbf{x}}})\|_\infty \max_{\substack{\bar{\mathbf{p}}\in\triangle\\ \bar{\mathbf{p}}_{[a]}\geqslant(1-d\kappa)\mathring{\bar{\mathbf{x}}}_{[a]},\forall a\\ \bar{\mathbf{p}}_{[a]}\leqslant 2\kappa+(1-2\kappa)\mathring{\bar{\mathbf{x}}}_{[a]},\forall a}} \|\bar{\mathbf{p}} - \mathring{\bar{\mathbf{x}}}\|_1\\ &\geqslant h(\mathring{\bar{\mathbf{x}}}) - 3d\kappa\|\nabla h(\mathring{\bar{\mathbf{x}}})\|_\infty\\ &\geqslant h(\mathring{\bar{\mathbf{x}}}) - 3d\kappa\beta\log_\beta\left(\frac{d}{\varepsilon}\right)\end{aligned} \tag{2.183}$$

so we can substitute into (2.180) to get

$$\frac{1}{T}\mathbb{E}\sum_{t=1}^{T}\psi_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}) - \psi_\beta(\hat{\mathring{\mathbf{x}}}^{(\varepsilon)}) \leqslant -\psi_\beta(\mathring{\mathbf{x}}^{(\varepsilon)}) + \frac{3d\kappa\beta}{1-\beta}\left(\left(\frac{d}{\varepsilon}\right)^{1-\beta} - 1\right) \tag{2.184}$$

Applying Lemma 2.A.6 completes the proof. □

Guaranteed exploration bounds

Lemma 2.A.10. Suppose we play $\texttt{OMD}_{\beta,\eta}$ with initialization $\mathbf{x}_1 \in \triangle^{(\varepsilon)}$, regularizer $\psi_\beta + I_{\triangle^{(\varepsilon)}}$ for some $\beta \in (0,1]$, and unbiased loss estimators ($\gamma = 0$) on the sequence of loss functions $\ell_1, \dots, \ell_T \in [0,1]^d$. Then for any $\mathring{a} \in [d]$ we have expected regret

$$\mathbb{E}\sum_{t=1}^{T}\ell_t(a_t) - \ell_t(\mathring{a}) \leqslant \frac{\mathbb{E}\mathcal{B}_\beta(\hat{\mathbf{x}}^{(\varepsilon)}||\mathbf{x}_1)}{\eta} + \frac{\eta d^\beta m}{\beta} + \varepsilon m \tag{2.185}$$

for $\hat{\mathbf{x}}$ the estimated optimum of the loss estimators $\hat{\ell}_1, \dots, \hat{\ell}_T$.

Proof.

$$\begin{aligned}
\mathbb{E}\sum_{t=1}^{T}\ell_t(a_t) - \ell_t(\mathring{a}) &= \mathbb{E}\sum_{t=1}^{T}\ell_t(a_t) - \langle\boldsymbol{\ell}_t, \mathring{\mathbf{x}}\rangle \\
&\leqslant \mathbb{E}\sum_{t=1}^{T}\ell_t(a_t) - \langle\boldsymbol{\ell}_t, \mathring{\mathbf{x}}^{(\varepsilon)}\rangle + \varepsilon m \\
&= \mathbb{E}\sum_{t=1}^{m}\hat{\ell}_t(a_t) - \langle\hat{\boldsymbol{\ell}}_t, \mathring{\mathbf{x}}^{(\varepsilon)}\rangle + \varepsilon m \\
&\leqslant \mathbb{E}\sum_{t=1}^{m}\hat{\ell}_t(a_t) - \langle\hat{\boldsymbol{\ell}}_t, \hat{\mathbf{x}}^{(\varepsilon)}\rangle + \varepsilon m \\
&\leqslant \mathbb{E}\left(\frac{\mathcal{B}_\beta(\hat{\mathbf{x}}^{(\varepsilon)}||\mathbf{x}_1)}{\eta} + \frac{\eta}{\beta}\sum_{t=1}^{T}\sum_{a=1}^{d}\hat{\ell}_t^2(a)\mathbf{x}_{t[a]}^{2-\beta}\right) + \varepsilon m \\
&\leqslant \frac{\mathbb{E}\mathcal{B}_\beta(\hat{\mathbf{x}}^{(\varepsilon)}||\mathbf{x}_1)}{\eta} + \frac{\eta d^\beta m}{\beta} + \varepsilon m
\end{aligned} \tag{2.186}$$

where the second inequality follows by optimality of $\hat{\mathbf{x}}$ for the estimated losses $\hat{\ell}_t$, the third by Lemma 2.A.7 constrained to $\triangle^{(\varepsilon)}$, and the fourth similarly to Theorem 2.A.14 (note both are also effectively shown in Luo [2017]). □

Theorem 2.A.15. In Algorithm 6, let $\texttt{OMD}_{\eta,\beta}$ be online mirror descent with the regularizer $\psi_\beta + I_{\triangle^{(\varepsilon)}}$ over unbiased ($\gamma = 0$) loss estimators, Θ_k is a subset of $[\underline{\beta}, \overline{\beta}] \subset [\frac{1}{\log d}, 1]$, and

$$U_t(\mathbf{x}, \eta, \beta) = \frac{\mathcal{B}_\beta(\hat{\mathbf{x}}_t^{(\varepsilon)}||\mathbf{x})}{\eta} + \frac{\eta d^\beta m}{\beta} \tag{2.187}$$

where $\hat{\mathbf{x}}_t^{(\varepsilon)} = (1-\varepsilon)\hat{\mathbf{x}}_t + \varepsilon \mathbf{1}_d/d$. Note that $U_t^{(\rho)}(\mathbf{x}, \eta, \beta) = U_t(\mathbf{x}, \eta, \beta) + \frac{\rho^2(d^{1-\beta}-1)}{\eta(1-\beta)}$. Then under Assumption 2.4.1 there exists settings of $\underline{\eta}, \overline{\eta}, \alpha, \lambda$ s.t. for all $\varepsilon, \rho \in (0,1)$ we have that

$$\begin{aligned}
&\mathbb{E}\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(a_{t,i}) - \ell_{t,i}(\mathring{a}_t) \\
&\leqslant \varepsilon m + \frac{8d\sqrt{m}}{\rho}\left(1_{k>1}\sqrt{\frac{\log k}{T}} + \frac{1+\log(T+1)}{16\rho T}\right) \\
&\quad + \min_{\beta\in[\underline{\beta},\overline{\beta}],\eta>0} \frac{8\left(\frac{d}{\varepsilon}\right)^{2-\underline{\beta}}(1+\log T)}{\eta T} + \frac{h_\beta(\Delta)}{\eta} + \frac{\eta d^\beta m}{\beta} + \frac{L_\eta(\overline{\beta}-\underline{\beta})}{2k} + d\min\left\{\frac{\rho^2}{2\eta}, \rho\sqrt{m}\right\}
\end{aligned} \tag{2.188}$$

for $L_\eta = \left(\frac{\log\frac{d}{\varepsilon}}{\eta} + \eta m \log^2 d\right) d$, $h_\beta(\Delta) = (H_\beta + \frac{56}{dm})\iota_\Delta + \frac{d^{1-\beta}-1}{1-\beta}(1-\iota_\Delta)$, and $\iota_\Delta = 1_{m \geqslant \frac{75d}{\varepsilon\Delta^2}\log\frac{d}{\varepsilon\Delta^2}}$.

Proof. By Lemma 2.A.10 we have

$$\mathbb{E}\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(a_{t,i}) - \ell_{t,i}(\mathring{a}_t) \leqslant \varepsilon m T + \mathbb{E}\sum_{t=1}^{T}\frac{\mathcal{B}_{\beta_t}(\hat{\mathbf{x}}_t^{(\varepsilon)}||\mathbf{x}_{t,1})}{\eta_t} + \frac{\eta_t d^{\beta_t} m}{\beta_t} \tag{2.189}$$

Since we have the same environment-dependent quantities as in Theorem 2.A.14, we can substitute the above bound into Theorem 2.A.13 and then apply the Lemma 2.A.9 bound

$$\begin{aligned}
\mathbb{E}\hat{V}_\beta^2 \leqslant H_\beta + \frac{3d\kappa\beta}{1-\beta}\left(\left(\frac{d}{\varepsilon}\right)^{1-\beta} - 1\right) &\leqslant H_\beta + \frac{3d^2}{\varepsilon}\exp\left(-\frac{3\varepsilon\Delta^2 m}{28d}\right) \\
&= H_\beta + \frac{3\varepsilon\Delta^2}{d^2}\exp\left(4\log\frac{d}{\varepsilon\Delta^2} - \frac{3\varepsilon\Delta^2 m}{28d}\right) \\
&\leqslant H_\beta + \frac{3\varepsilon\Delta^2/d^2}{\frac{3\varepsilon\Delta^2 m}{28d} - 4\log\frac{d}{\varepsilon\Delta^2}} \\
&\leqslant H_\beta + \frac{56}{dm}
\end{aligned} \tag{2.190}$$

where the last line follows by assuming $m \geqslant \frac{75d}{\varepsilon\Delta^2}\log\frac{d}{\varepsilon\Delta^2}$. If this condition does not hold, then we apply the default bound of $\mathbb{E}\hat{V}_\beta^2 \leqslant = \frac{1}{T}\sum_{t=1}^{T}\psi_\beta(\hat{\mathbf{x}}_t) - \psi_\beta(\hat{\bar{\mathbf{x}}}) \leqslant \frac{d^{1-\beta}-1}{1-\beta}$. □

Corollary 2.A.13. Let $\underline{\beta} = \overline{\beta} = 1$. Then for known Δ and assuming $m \geqslant \frac{75d}{\Delta^2}\log\frac{d}{\Delta^2}$ we can ensure expected task-averaged regret at most

$$2\sqrt{H_1 dm + 56} + \frac{75d}{\Delta^2}W\left(\frac{m}{75}\right) + \tilde{\mathcal{O}}\left(\frac{d^{\frac{3}{2}}m^{\frac{3}{4}}}{\sqrt{T}} + \frac{d\Delta^2 m^2}{T}\right) \tag{2.191}$$

where W is the Lambert W-function, while for unknown Δ we can ensure expected task-averaged regret at most

$$2\sqrt{H_1 dm + 56} + \frac{3}{\Delta}\sqrt[3]{50dm\log d\log\frac{d^2m^2}{150\Delta^6\log d}} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{3}{2}}m^{\frac{3}{4}}}{\sqrt{T}} + \frac{d^{\frac{4}{3}}m^{\frac{5}{3}}}{T}\right) \tag{2.192}$$

so long as $m^2 \geqslant 150d \log d$.

Proof. Applying Theorem 2.A.15 and simplifying yields

$$\varepsilon m + \frac{8d\sqrt{m}(1+\log(T+1))}{16\rho^2 T} + \min_{\eta>0} \frac{8d(1+\log T)}{\varepsilon\eta T} + \frac{h_1(\Delta)}{\eta} + \eta dm + \frac{d\rho^2}{2\eta} \tag{2.193}$$

Then substitute $\eta = \sqrt{\frac{h_1(\Delta)}{dm}}$ and set $\rho = \sqrt[4]{\frac{1}{dT\sqrt{m}}}$ and $\varepsilon = \frac{75d}{\Delta^2 m} W(\frac{m}{75})$ (for known Δ) or $\varepsilon = \sqrt[3]{\frac{150d\log d}{m^2}}$ (otherwise). □

Corollary 2.A.14. Let $\underline{\beta} = \frac{1}{2}$ and $\overline{\beta} = 1$. Then for known Δ and assuming $m \geqslant \frac{75d}{\Delta^2} \log \frac{d}{\Delta^2}$ we can ensure task-averaged regret at most

$$\min_{\beta\in[\frac{1}{2},1]} 2\sqrt{(H_\beta m + 56/d)d^\beta/\beta} + \frac{75d}{\Delta^2} W\left(\frac{m}{75}\right) + \tilde{\mathcal{O}}\left(\frac{d^{\frac{4}{3}} m^{\frac{2}{3}}}{\sqrt[3]{T}} + \frac{d^{\frac{5}{3}} m^{\frac{5}{6}}}{T^{\frac{2}{3}}} + \frac{d\Delta^3 m^{\frac{5}{2}}}{T}\right) \tag{2.194}$$

using $k = \lceil \sqrt[3]{d^2 mT} \rceil$, while for unknown Δ we can ensure expected task-averaged regret at most

$$\min_{\beta\in[\frac{1}{2},1]} 2\sqrt{(H_\beta m + 56/d)d^\beta/\beta} + \frac{3}{\Delta}\sqrt[3]{50d^2 m \log\frac{dm^2}{150\Delta^6}} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{4}{3}} m^{\frac{2}{3}}}{\sqrt[3]{T}} + \frac{d^{\frac{5}{3}} m^{\frac{5}{6}}}{T^{\frac{2}{3}}} + \frac{d^{\frac{3}{2}} m^2}{T}\right) \tag{2.195}$$

so long as $m \geqslant 5d\sqrt{6}$.

Proof. Applying Theorem 2.A.15 and simplifying yields

$$\begin{aligned}\varepsilon m + \frac{8d\sqrt{m}}{\rho}\left(\sqrt{\frac{\log k}{T}} + \frac{1+\log(T+1)}{16\rho T}\right) \\ + \min_{\beta\in[\underline{\beta},\overline{\beta}],\eta>0} \frac{8d^{\frac{3}{2}}(1+\log T)}{\varepsilon^{\frac{3}{2}}\eta T} + \frac{h_\beta(\Delta)}{\eta} + \frac{\eta d^\beta m}{\beta} + \frac{d}{4k}\left(\frac{\log\frac{d}{\varepsilon}}{\eta} + \eta m \log^2 d\right) + \frac{d\rho^2}{2\eta}\end{aligned} \tag{2.196}$$

Then substitute $\eta = \sqrt{\frac{h_\beta(\Delta)}{d^\beta m/\beta}}$ and set $\rho = \sqrt[3]{\frac{1}{d\sqrt{mT}}}$ and $\varepsilon = \frac{75d}{\Delta^2 m} W(\frac{m}{75})$ (for known Δ) or $\varepsilon = \sqrt[3]{\frac{150d^2}{m^2}}$ (otherwise). □

Corollary 2.A.15. Let $\underline{\beta} = \frac{1}{\log d}$ and $\overline{\beta} = 1$. Then for known Δ and assuming $m \geqslant \frac{75d}{\Delta^2} \log \frac{d}{\Delta^2}$ we can ensure task-averaged regret at most

$$\min_{\beta\in(0,1]} 2\sqrt{(H_\beta m + 56/d)d^\beta/\beta} + \frac{75d}{\Delta^2} W\left(\frac{m}{75}\right) + \tilde{\mathcal{O}}\left(\frac{d^{\frac{4}{3}} m^{\frac{2}{3}}}{\sqrt[3]{T}} + \frac{d^{\frac{5}{3}} m^{\frac{5}{6}}}{T^{\frac{2}{3}}} + \frac{d\Delta^4 m^3}{T}\right) \tag{2.197}$$

using $k = \lceil \sqrt[3]{d^2 mT} \rceil$, while for unknown Δ we can ensure expected task-averaged regret at most

$$\min_{\beta\in(0,1]} 2\sqrt{(H_\beta m + 56/d)d^\beta/\beta} + \frac{3}{\Delta}\sqrt[3]{50d^2 m \log\frac{dm^2}{150\Delta^6}} + \tilde{\mathcal{O}}\left(\frac{d^{\frac{4}{3}} m^{\frac{2}{3}}}{\sqrt[3]{T}} + \frac{d^{\frac{5}{3}} m^{\frac{5}{6}}}{T^{\frac{2}{3}}} + \frac{d^{\frac{5}{3}} m^{\frac{7}{3}}}{T}\right) \tag{2.198}$$

so long as $m \geqslant 5d\sqrt{6}$.

Proof. Applying Theorem 2.A.15 and simplifying yields

$$\begin{aligned}&\varepsilon m + \frac{8d\sqrt{m}}{\rho}\left(\sqrt{\frac{\log k}{T}} + \frac{1+\log(T+1)}{16\rho T}\right)\\&\quad + \min_{\beta\in[\underline{\beta},\overline{\beta}],\eta>0} \frac{8d^2(1+\log T)}{\varepsilon^2\eta T} + \frac{h_\beta(\Delta)}{\eta} + \frac{\eta d^\beta m}{\beta} + \frac{d}{2k}\left(\frac{\log\frac{d}{\varepsilon}}{\eta} + \eta m\log^2 d\right) + \frac{d\rho^2}{2\eta}\end{aligned} \tag{2.199}$$

Then substitute $\eta = \sqrt{\frac{h_\beta(\Delta)}{d^\beta m/\beta}}$ and set $\rho = \sqrt[3]{\frac{1}{d\sqrt{mT}}}$ and $\varepsilon = \frac{75d}{\Delta^2 m}W(\frac{m}{75})$ (for known Δ) or $\varepsilon = \sqrt[3]{\frac{150d^2}{m^2}}$ (otherwise). □

Corollary 2.A.16. Let $\underline{\beta} = \frac{1}{\log d}$ and $\overline{\beta} = 1$. Then for unknown Δ and assuming $m \geqslant \max\{d^{\frac{3}{4}}, 56\}$ we can ensure task-averaged regret at most

$$\begin{aligned}&\min_{\beta\in(0,1]} \min\left\{8\sqrt{dm}, 2\sqrt{\left(H_\beta m + \frac{56}{d}\right)\frac{d^\beta}{\beta}} + \frac{21d^{\frac{3}{4}}\sqrt[3]{m}}{\Delta}\sqrt{3\log\frac{dm}{\Delta^2}}\right\}\\&\quad + \tilde{\mathcal{O}}\left(\frac{d^{\frac{4}{3}}m^{\frac{2}{3}}}{\sqrt[3]{T}} + \frac{d^{\frac{5}{3}}m^{\frac{5}{6}}}{T^{\frac{2}{3}}} + \frac{d^2m^{\frac{7}{3}}}{T}\right)\end{aligned} \tag{2.200}$$

using $k = \lceil\sqrt[3]{d^2mT}\rceil$.

Proof. Applying Theorem 2.A.15 and simplifying yields

$$\begin{aligned}&\varepsilon m + \frac{8d\sqrt{m}}{\rho}\left(\sqrt{\frac{\log k}{T}} + \frac{1+\log(T+1)}{16\rho T}\right)\\&\quad + \min_{\beta\in[\underline{\beta},\overline{\beta}],\eta>0} \frac{8d^2(1+\log T)}{\varepsilon^2\eta T} + \frac{h_\beta(\Delta)}{\eta} + \frac{\eta d^\beta m}{\beta} + \frac{d}{2k}\left(\frac{\log\frac{d}{\varepsilon}}{\eta} + \eta m\log^2 d\right) + \frac{d\rho^2}{2\eta}\end{aligned} \tag{2.201}$$

Then substitute $\eta = \sqrt{\frac{h_\beta(\Delta)}{d^\beta m/\beta}}$ and set $\rho = \sqrt[3]{\frac{1}{d\sqrt{mT}}}$ and $\varepsilon = \frac{\sqrt{d}}{\sqrt[3]{m^2}}$. □

2.A.9 Robustness to outliers

Proposition 2.A.6. Suppose there exists a constant $p \in [0,1]$ and a subset $S \subset [T]$ of size s such that $\mathring{a}_t \in S$ for all but $\mathcal{O}(T^p)$ MAB tasks $t \in [T]$. Then if $\beta \in [\frac{1}{\log d}, \frac{1}{2}]$ we have $H_\beta = \mathcal{O}(s + \frac{d^{1-\beta}}{T^{\beta(1-p)}})$.

Proof. Define the vector $\mathbf{e}_S \in [0,1]^d$ s.t. $\mathbf{e}_{S[a]} = 1_{a\in S}$. Then by Claim B.2.2 and the mean-as-

minimizer property of Bregman divergences we have

$$
\begin{aligned}
H_\beta &= -\psi_\beta(\mathring{\tilde{\mathbf{x}}}) \\
&= \frac{1}{T}\sum_{t=1}^{T}\psi_\beta(\mathring{\mathbf{x}}_t) - \psi_\beta(\mathring{\tilde{\mathbf{x}}}) \\
&= \frac{1}{T}\sum_{t=1}^{T}\mathcal{B}_\beta(\mathring{\mathbf{x}}_t||\mathring{\tilde{\mathbf{x}}}) \\
&= \min_{\mathbf{x}\in\triangle_d}\frac{1}{T}\sum_{t=1}^{T}\mathcal{B}_\beta(\mathring{\mathbf{x}}_t||\mathring{\tilde{\mathbf{x}}}) \\
&\leqslant \min_{\delta\in(0,1)}\frac{1}{T}\sum_{t=1}^{T}\mathcal{B}_\beta\left(\mathring{\mathbf{x}}_t\middle\|\frac{1-\delta}{s}\mathbf{e}_S + \frac{\delta}{d}\mathbf{1}_d\right) \\
&= \min_{\delta\in(0,1)}\frac{1}{T}\sum_{t=1}^{T}\frac{1}{1-\beta}\sum_{a=1}^{d}\left(\frac{1-\delta}{s}1_{a\in S} + \frac{\delta}{d}\right)^\beta - \mathring{\mathbf{x}}_{t[a]}^\beta + \frac{\beta(\mathring{\mathbf{x}}_{t[a]} - \frac{1-\delta}{s}1_{a\in S} - \frac{\delta}{d})}{(\frac{1-\delta}{s}1_{s\in S} + \frac{\delta}{d})^\beta} \\
&= \min_{\delta\in(0,1)}\frac{1}{T}\sum_{t=1}^{T}\sum_{a=1}^{d}\left(\frac{1-\delta}{s}1_{a\in S} + \frac{\delta}{d}\right)^\beta - \frac{\mathring{\mathbf{x}}_{t[a]}^\beta}{1-\beta} + \frac{\beta\mathring{\mathbf{x}}_{t[a]}^\beta}{(1-\beta)(\frac{1-\delta}{s}1_{a\in S} + \frac{\delta}{d})^{1-\beta}} \\
&\leqslant \min_{\delta\in(0,1)} s^{1-\beta} + \delta^\beta d^{1-\beta} + \frac{\beta}{(1-\beta)T}\sum_{t=1}^{T}\sum_{a=1}^{d}\frac{1_{a=\mathring{a}_t}}{(1-\beta)(\frac{1-\delta}{s}1_{a\in S} + \frac{\delta}{d})^{1-\beta}} \\
&\leqslant \min_{\delta\in(0,1)}\frac{s^{1-\beta}}{1-\beta} + \delta^\beta d^{1-\beta} + \mathcal{O}\left(\frac{\beta(\frac{d}{\delta})^{1-\beta}}{(1-\beta)T^{1-p}}\right) \\
&= \mathcal{O}\left(s + \frac{d^{1-\beta}}{T^{\beta(1-p)}}\right)
\end{aligned}
\tag{2.202}
$$

where the last line follows by considering $\delta = 1/T^{1-p}$. □

2.A.10 Online learning with self-concordant barrier regularizers

General results

Lemma 2.A.11. Let $\mathcal{K} \subset \mathbb{R}^d$ be a convex set and $\psi : \mathcal{K}^\circ \mapsto \mathbb{R}^d$ be a self-concordant barrier. Suppose $\ell_1, \ldots, \ell_T$ are a sequence of loss functions satisfying $|\langle \boldsymbol{\ell}_t, \mathbf{x}\rangle| \leqslant 1 \; \forall \, \mathbf{x} \in \mathcal{K}$. Then if we run OMD with step-size $\eta > 0$ as in Abernethy et al. [2008b, Algorithm 1] on the sequence of estimators $\hat{\ell}_t$ our estimated regret w.r.t. any $\mathbf{x} \in \mathcal{K}_\varepsilon$ for $\varepsilon > 0$ will satisfy

$$
\sum_{t=1}^{T}\langle \hat{\boldsymbol{\ell}}_t, \mathbf{x}_t - \mathbf{x}\rangle \leqslant \frac{\mathcal{B}(\mathbf{x}||\mathbf{x}_1)}{\eta} + 32d^2\eta T \tag{2.203}
$$

Proof. The result follows from Abernethy et al. [2008b] by stopping the derivation on the second inequality below Equation 10. □

Definition 2.A.2. For any convex set $\mathcal{K}$ and any point $\mathbf{y} \in \mathcal{K}$, $\pi_{\mathbf{y}}(\mathbf{x}) = \inf_{t \geqslant 0, \mathbf{y} + \frac{\mathbf{x}-\mathbf{y}}{t} \in \mathcal{K}} t$ is the **Minkowski function with pole** $\mathbf{y}$.

Lemma 2.A.12. For any $\mathbf{x} \in \mathcal{K} \subset \mathbb{R}^d$ and $\psi : \mathcal{K}^\circ \mapsto \mathbb{R}$ a ν-self-concordant regularizer with minimum $\mathbf{x}_1 \in \mathcal{K}^\circ$, the quantity $\psi(\mathbf{c}_\varepsilon(\mathbf{x}))$ is $\nu\sqrt{2}$-Lipschitz w.r.t. $\varepsilon \in [0, 1]$.

Proof. Consider any $\varepsilon, \varepsilon' \in [0, 1]$ s.t. $\varepsilon' - \varepsilon \in (0, \frac{1}{2}]$ Note that for $t = \frac{\varepsilon' - \varepsilon}{1+\varepsilon}$ we have

$$\mathbf{c}_{\varepsilon'}(\mathbf{x}) + \frac{\mathbf{c}_{\varepsilon'}(\mathbf{x}) - \mathbf{c}_\varepsilon(\mathbf{x})}{t} = \mathbf{x}_1 + \frac{\mathbf{x} - \mathbf{x}_1}{1 + \varepsilon'} + \frac{\mathbf{x}_1 + \frac{\mathbf{x}-\mathbf{x}_1}{1+\varepsilon} - \mathbf{x}_1 - \frac{\mathbf{x}-\mathbf{x}_1}{1+\varepsilon'}}{t} = \mathbf{x} \in \mathcal{K} \tag{2.204}$$

so $\pi_{\mathbf{c}_{\varepsilon'}(\mathbf{x})}(\mathbf{c}_\varepsilon(\mathbf{x})) \leqslant \frac{\varepsilon' - \varepsilon}{1+\varepsilon} \leqslant \varepsilon' - \varepsilon$. Therefore by Nesterov and Nemirovskii [1994, Proposition 2.3.2] we have

$$\psi(\mathbf{c}_\varepsilon(\mathbf{x})) - \psi(\mathbf{c}_{\varepsilon'}(\mathbf{x})) \leqslant \nu \log\left(\frac{1}{1 - \pi_{\mathbf{c}_{\varepsilon'}(\mathbf{x})}(\mathbf{c}_\varepsilon(\mathbf{x}))}\right) \leqslant \nu \log\left(\frac{1}{1 + \varepsilon - \varepsilon'}\right) \leqslant \nu(\varepsilon' - \varepsilon)\sqrt{2} \tag{2.205}$$

where for the last inequality we used $-\log(1-x) \leqslant x\sqrt{2}$ for $x \in [0, \frac{1}{2}]$. The case of $\varepsilon' - \varepsilon \in (0, 1]$ follows by considering $\varepsilon'' = \frac{\varepsilon' + \varepsilon}{2}$ and applying the above twice. □

Theorem 2.A.16. In Algorithm 6, let $\texttt{OMD}_{\eta,\varepsilon}$ be online mirror descent over loss estimators specified in Abernethy et al. [2008b] with a ν-self-concordant barrier regularizer $\psi : \mathcal{K}^\circ \mapsto \mathbb{R}$ that satisfies $\nu \geqslant 1$ and $\|\nabla^2\psi(\mathbf{x}_1)\|_\infty = S_1 \geqslant 1$. Let Θ_k be a subset of $[\frac{1}{m}, 1]$ and

$$U_t(\mathbf{x}, \eta, \varepsilon) = \frac{\mathcal{B}(\mathbf{c}_\varepsilon(\hat{\mathbf{x}}) || \mathbf{x})}{\eta} + 32\eta d^2 + \varepsilon m \tag{2.206}$$

Note that $U_t^{(\rho)}(\mathbf{x}, \eta, \varepsilon) = U_t(\mathbf{x}, \eta, \varepsilon) + \frac{9\nu^{\frac{3}{2}}\rho^2 K m \sqrt{S_1}}{\eta}$. Then there exists settings of $\underline{\eta}, \overline{\eta}, \alpha, \lambda$ s.t. for all $\varepsilon, \rho \in (0, 1)$ we have expected task averaged regret at most

$$\begin{aligned}
&\mathbb{E} \min_{\varepsilon \in [\frac{1}{m}, 1], \eta > 0} \frac{512\nu^2 K^2 S_1 m^2 (1 + \log T)}{\eta} + \left(\frac{\hat{V}_\varepsilon^2}{\eta} + 32\eta d^2 m + \varepsilon m + \frac{\nu\sqrt{2}/\eta + m}{k}\right) T \\
&\quad + 3\nu^{\frac{3}{4}} m \min\left\{\frac{3\rho^2 \nu^{\frac{3}{4}} K \sqrt{S_1}}{\eta}, 4d\rho\sqrt{2K\sqrt{S_1}}\right\} T \\
&\quad + \frac{7dm}{\rho}\sqrt{2K\sqrt{\nu^3 S_1}}\left(7\sqrt{T \log k} + \frac{1 + \log(T+1)}{\rho}\right)
\end{aligned} \tag{2.207}$$

Proof. Let $\underline{\varepsilon} = \frac{1}{m}$. For any $\varepsilon \in [\underline{\varepsilon}, 1]$ and $\mathbf{x} \in \mathcal{K}$ we have $\pi_{\mathbf{x}_1}(\mathbf{c}_\varepsilon(\mathbf{x})) \leqslant \frac{1}{1+\varepsilon}$, so by Nesterov and Nemirovskii [1994, Proposition 2.3.2] we have

$$\|\nabla^2\psi(\mathbf{c}_\varepsilon(\mathbf{x}))\|_\infty \leqslant \left(\frac{1 + 3\nu}{1 - \pi_{\mathbf{x}_1}(\mathbf{c}_\varepsilon(\mathbf{x}))}\right)^2 \|\nabla^2\psi(\mathbf{x}_1)\|_\infty \leqslant \frac{64\nu^2 S_1}{\varepsilon^2} \tag{2.208}$$

Thus $S = \max_{\mathbf{x},\mathbf{y}\in\mathcal{K},\varepsilon\in[\underline{\varepsilon},1]} \|\nabla^2\psi(\mathbf{c}_\varepsilon(\mathbf{x}))\|_\infty = \frac{64\nu^2 S_1}{\underline{\varepsilon}^2}$ and also

$$\begin{aligned}
D_\varepsilon^2 &= \max_{\mathbf{x},\mathbf{y}\in\mathcal{K}} \mathcal{B}(\mathbf{c}_\varepsilon(\mathbf{x})||\mathbf{c}_\varepsilon(\mathbf{y})) \\
&= \max_{\mathbf{x},\mathbf{y}\in\mathcal{K}} \psi(\mathbf{c}_\varepsilon(\mathbf{x})) - \psi(\mathbf{c}_\varepsilon(\mathbf{y})) - \langle\nabla\psi(\mathbf{c}_\varepsilon(\mathbf{y})), \mathbf{x}-\mathbf{y}\rangle \\
&\leqslant \max_{\mathbf{x},\mathbf{y}\in\mathcal{K}} \nu\log\left(\frac{1}{1-\pi_{\mathbf{x}_1}(\mathbf{c}_\varepsilon(\mathbf{x}))}\right) + \sqrt{\nu\|\nabla^2\psi(\mathbf{c}_\varepsilon(\mathbf{y}))\|_2}\|\mathbf{x}-\mathbf{y}\|_2 \\
&\leqslant \nu\log\frac{2}{\varepsilon} + \frac{8\nu^{\frac{3}{2}}K\sqrt{S_1}}{\varepsilon} \\
&\leqslant \frac{9\nu^{\frac{3}{2}}K\sqrt{S_1}}{\varepsilon}
\end{aligned} \tag{2.209}$$

where the first inequality follows by Nesterov and Nemirovskii [1994, Proposition 2.3.2] and the definition of a self-concordant barrier [Abernethy et al., 2008b, Definition 5]. In addition, we have $g(\varepsilon) = 32d^2$, $f(\varepsilon) = \varepsilon$, $M = 12d\sqrt{2Km/\underline{\varepsilon}}\sqrt[4]{\nu^3 S_1}$, and $F = 1$. We have

$$\begin{aligned}
\mathbb{E}\sum_{t=1}^{T}\sum_{i=1}^{m}\langle\boldsymbol{\ell}_{t,i}, \mathbf{x}_{t,i} - \mathring{\mathbf{x}}_t\rangle &\leqslant \mathbb{E}\sum_{t=1}^{T}\varepsilon_t m + \sum_{i=1}^{m}\langle\boldsymbol{\ell}_{t,i}, \mathbf{x}_{t,i} - \mathbf{c}_{\varepsilon_t}(\mathring{\mathbf{x}}_t)\rangle \\
&\leqslant \mathbb{E}\sum_{t=1}^{T}\varepsilon_t m + \sum_{i=1}^{m}\langle\hat{\boldsymbol{\ell}}_{t,i}, \mathbf{x}_{t,i} - \mathbf{c}_{\varepsilon_t}(\mathring{\mathbf{x}}_t)\rangle \\
&\leqslant \mathbb{E}\sum_{t=1}^{T}\varepsilon_t m + \sum_{i=1}^{m}\langle\hat{\boldsymbol{\ell}}_{t,i}, \mathbf{x}_{t,i} - \mathbf{c}_{\varepsilon_t}(\hat{\mathbf{x}}_t)\rangle \\
&\leqslant \sum_{t=1}^{T}\frac{\mathbb{E}\mathcal{B}(\mathbf{c}_{\varepsilon_t}(\hat{\mathbf{x}}_t||\mathbf{x}_{t,1})}{\eta_t} + (32\eta_t d^2 + \varepsilon_t)m
\end{aligned} \tag{2.210}$$

where the first inequality follows by Abernethy et al. [2008b, Lemma 8], the second by Abernethy et al. [2008b, Lemma 3], the third by optimality of $\hat{\mathbf{x}}_t$, and the fourth by Lemma 2.A.11. Substituting into Theorem 2.A.13 and simplifying yields the result. □

Specialization to the unit sphere

Corollary 2.A.17. Let $\mathcal{K}$ be the unit sphere with the self-concordant barrier $\psi(\mathbf{x}) = -\log(1 - \|\mathbf{x}\|_2^2)$. Then Algorithm 6 attains expected task-averaged regret bounded by

$$\tilde{\mathcal{O}}\left(\frac{dm^{\frac{3}{2}}}{T^{\frac{3}{4}}} + \frac{dm}{\sqrt[4]{T}}\right) + \min_{\varepsilon\in[\frac{1}{m},1]} 4d\sqrt{2m\log\left(1 + \frac{1-\mathbb{E}\|\hat{\mathbf{x}}\|_2^2}{2\varepsilon+\varepsilon^2}\right)} + \varepsilon m \tag{2.211}$$

using $k = \lceil\sqrt{T}\rceil$.

Proof. Using the fact the $\nu = 1$ and $K = S_1 = 2$, we apply Theorem 2.A.16 and simplify to obtain

$$\mathbb{E}\min_{\varepsilon\in[\frac{1}{m},1],\eta>0}\frac{\hat{V}_\varepsilon^2}{\eta} + 32\eta d^2 m + \varepsilon m + \tilde{\mathcal{O}}\left(\frac{m^2}{\eta T} + \frac{1}{\eta k} + \frac{m}{k} + m\min\left\{\frac{\rho^2}{\eta}, d\rho\right\} + \frac{dm}{\rho\sqrt{T}} + \frac{dm}{\rho^2 T}\right) \tag{2.212}$$

Then substitute $\eta = \frac{\hat{V}_\varepsilon}{4\sqrt{2dm}} + \frac{\sqrt{m}}{d\sqrt[4]{T}}$, set $\rho = \frac{1}{\sqrt[4]{T}}$, and note that

$$\begin{aligned}\mathbb{E}\hat{V}_\varepsilon = \mathbb{E}\sqrt{\log\left(\frac{1-\|\mathbf{c}_\varepsilon(\hat{\tilde{\mathbf{x}}})\|_2^2}{\sqrt[T]{\prod_{t=1}^T 1-\|\mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)\|_2^2}}\right)} &= \mathbb{E}\sqrt{\log\left(\frac{1-(1+\varepsilon)^{-2}\|\hat{\tilde{\mathbf{x}}}\|_2^2}{1-(1+\varepsilon)^{-2}}\right)} \\ &\leqslant \sqrt{\log\left(1+\frac{1-\mathbb{E}\|\hat{\tilde{\mathbf{x}}}\|_2^2}{2\varepsilon+\varepsilon^2}\right)}\end{aligned} \tag{2.213}$$

where we use the fact that $\|\hat{\mathbf{x}}_t\|_2 = 1$ and the inequality is Jensen's. □

Specialization to polytopes, specifically the bandit online shortest-path problem

Corollary 2.A.18. Let $\mathcal{K} = \{\mathbf{x} \in [0,1]^{|E|} : \langle \mathbf{a}, \mathbf{x}\rangle \leqslant b \ \forall\ (\mathbf{a}, b) \in \mathcal{C}\}$ be the set of flows from u to v on a graph $G(V, E)$, where $\mathcal{C} \subset \mathbb{R}^{|E|} \times \mathbb{R}$ is a set of $\mathcal{O}(|E|)$ linear constraints. Suppose we see T instances of the bandit online shortest path problem with m timesteps each. Then sampling from probability distributions over paths from u to v returned by running Algorithm 6 with regularizer $\psi(\mathbf{x}) = -\sum_{\mathbf{a},b\in\mathcal{C}} \log(b - \langle \mathbf{a}, \mathbf{x}\rangle)$ attains the following expected average regret across instances

$$\tilde{\mathcal{O}}\left(\frac{|E|^4 m^{\frac{3}{2}}}{T^{\frac{3}{4}}} + \frac{|E|^{\frac{5}{2}} m^{\frac{5}{6}}}{\sqrt[4]{T}}\right) + \min_{\varepsilon\in[\frac{1}{m},1]} 4|E|\mathbb{E}\sqrt{2m\sum_{\mathbf{a},b\in\mathcal{C}} \log\left(\frac{\frac{1}{T}\sum_{t=1}^T b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)\rangle}{\sqrt[T]{\prod_{t=1}^T b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)\rangle}}\right)} + \varepsilon m \tag{2.214}$$

using $k = \lceil\sqrt{T}\rceil$.

Proof. Using the fact that $d = |E|$, $\nu = \mathcal{O}(|E|)$, $K = \sqrt{|E|}$, and $S_1 \leqslant \sum_{\mathbf{a},b\in\mathcal{C}} \frac{\|\mathbf{a}\mathbf{a}^\top\|_\infty}{(\langle \mathbf{a}, \mathbf{1}_{|E|}/|E|\rangle - \mathbf{b})^2} = \mathcal{O}(|E|^3)$, we apply Theorem 2.A.16 and simplify to obtain

$$\begin{aligned}&\mathbb{E}\min_{\varepsilon\in[\frac{1}{m},1],\eta>0} \frac{\hat{V}_\varepsilon^2}{\eta} + 32\eta|E|^2 m + \varepsilon m \\ &\quad + \tilde{\mathcal{O}}\left(\frac{|E|^6 m^2}{\eta T} + \frac{|E|}{\eta k} + \frac{m}{k} + m\min\left\{\frac{\rho^2|E|^{\frac{7}{2}}}{\eta}, \rho|E|^{\frac{11}{4}}\right\} + \frac{|E|^{\frac{11}{4}} m}{\rho}\left(\frac{1}{\sqrt{T}} + \frac{1}{\rho T}\right)\right)\end{aligned} \tag{2.215}$$

Then substitute $\eta = \frac{\hat{V}_\varepsilon}{4\sqrt{2dm}} + \frac{|E|^2\sqrt{m}}{\sqrt[4]{T}}$, set $\rho = \sqrt[4]{\frac{|E|}{T}}\sqrt[6]{m}$, and note that

$$\hat{V}_\varepsilon^2 = \sum_{\mathbf{a},b\in\mathcal{C}} \log\left(\frac{b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\tilde{\mathbf{x}}})\rangle}{\sqrt[T]{\prod_{t=1}^T b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)\rangle}}\right) = \sum_{\mathbf{a},b\in\mathcal{C}} \log\left(\frac{\frac{1}{T}\sum_{t=1}^T b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)\rangle}{\sqrt[T]{\prod_{t=1}^T b - \langle \mathbf{a}, \mathbf{c}_\varepsilon(\hat{\mathbf{x}}_t)\rangle}}\right) \tag{2.216}$$

□

Omniglot	**1-shot**				**5-shot**			
5-way	evaluation setting		hyperparameters		evaluation setting		hyperparameters	
	regular	transductive	$\eta = \frac{\varepsilon}{\zeta}$	c	regular	transductive	$\eta = \frac{\varepsilon}{\zeta}$	c
MAML (1)		98.3 ± 0.5				99.2 ± 0.2		
Reptile	95.39 ± 0.09	97.68 ± 0.04	$1E-3$		98.90 ± 0.10	99.48 ± 0.06	$1E-3$	
ARUBA	94.57 ± 1.04	97.44 ± 0.32	$1E-1$		98.64 ± 0.04	99.29 ± 0.07	$1E-2$	
ARUBA++	94.80 ± 1.10	97.58 ± 0.13	$1E-1$	10^3	98.93 ± 0.13	99.46 ± 0.02	$1E-2$	10^3
MAML (2)		98.7 ± 0.4				99.9 ± 0.1		
Meta-SGD		99.53 ± 0.26				99.93 ± 0.09		

Table 2.3: Meta-learning evaluations on the 5-way Omniglot classification task.

Omniglot	**1-shot**				**5-shot**			
20-way	evaluation setting		hyperparameters		evaluation setting		hyperparameters	
	regular	transductive	$\eta = \frac{\varepsilon}{\zeta}$	c	regular	transductive	$\eta = \frac{\varepsilon}{\zeta}$	c
MAML (1)		95.8 ± 0.3				98.9 ± 0.2		
Reptile	88.14 ± 0.15	89.43 ± 0.14	$5E-4$		96.65 ± 0.33	97.12 ± 0.32	$5E-4$	
ARUBA	85.61 ± 0.25	86.67 ± 0.17	$5E-3$		96.02 ± 0.12	96.61 ± 0.13	$5E-3$	
ARUBA++	88.38 ± 0.24	89.66 ± 0.3	$5E-3$	10^3	96.99 ± 0.35	97.49 ± 0.28	$5E-3$	10
MAML (2)		95.8 ± 0.3				98.9 ± 0.2		
Meta-SGD		95.93 ± 0.38				98.97 ± 0.19		

Table 2.4: Meta-learning evaluations on the 20-way Omniglot classification task.

2.B Experimental details

2.B.1 Adaptive gradient-based meta-learning

Code is available at `https://github.com/mkhodak/ARUBA`.

Reptile

For our Reptile experiments we use the code and default settings provided by Nichol et al. [2018], except we tune the learning rate, which for ARUBA corresponds to ε/ζ, and the coefficient c in ARUBA++. In addition to the the parameters listed in Tables 2.3, 2.4, and 2.5, we set $\zeta = p = 1.0$ for all experiments. All evaluations are averages of three runs.

FedAvg

For FedAvg we train a 2-layer stacked LSTM model with 256 hidden units, 8-dimensional trained character embeddings, with a maximum input string size of 80 characters; these settings are used to match those of McMahan et al. [2017]. Similarly, we take their approach of only removing those actors from the Shakespeare dataset with fewer than two lines and split each user temporally into train/test sets with a training fraction of 0.8. Unlike McMahan et al. [2017], we also split the users into meta-training and meta-testing sets, also with a fraction of 0.8, in order to evaluate meta-test performance. We run both algorithms for 500 rounds with a batch of 10 users per round and a within-task batch-size of 10, as in Caldas et al. [2018]. For unmodified

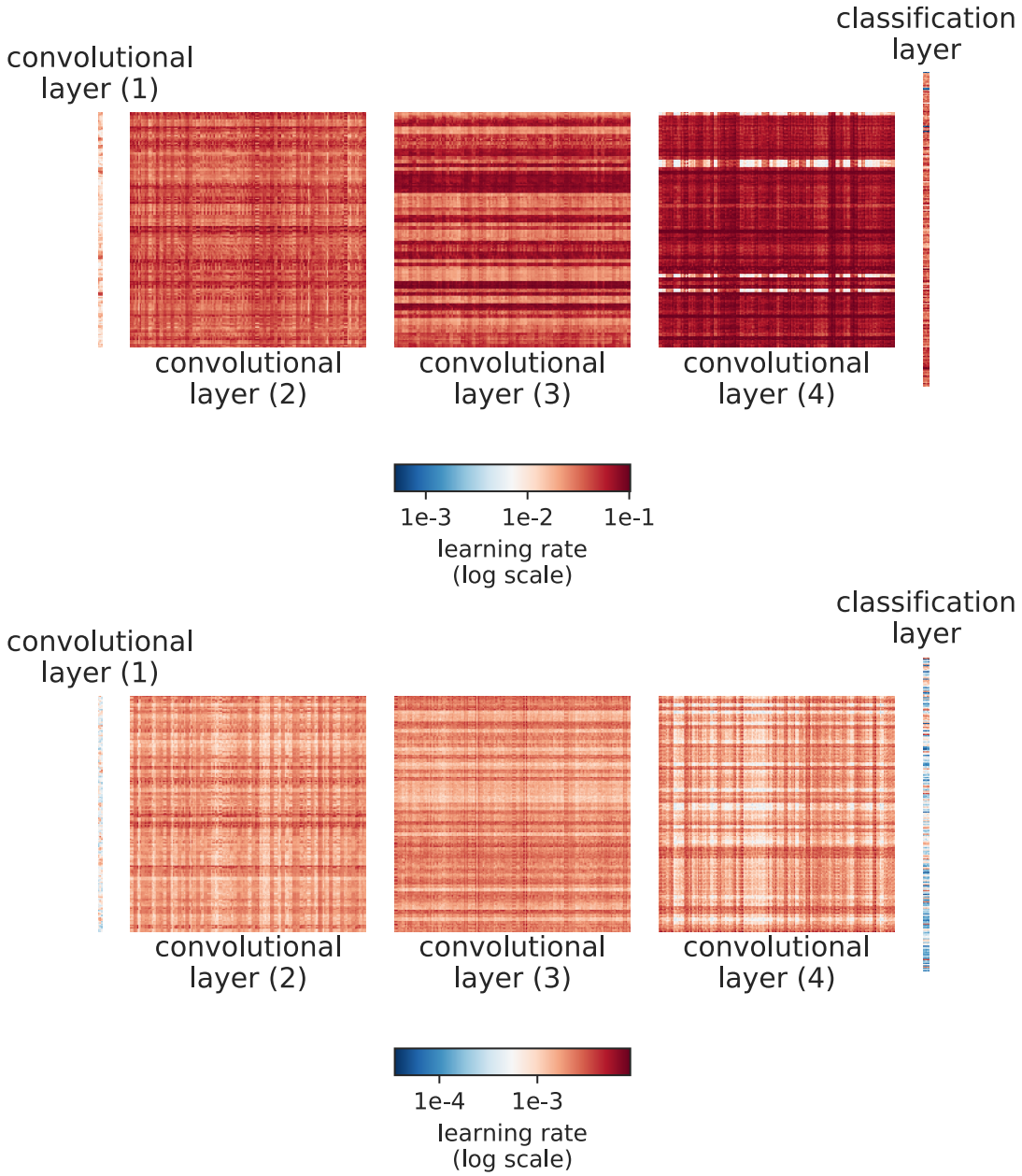

Figure 2.4: Final learning rate η_T across the layers of a CNN trained on 1-shot 5-way Omniglot (top) and 5-shot 5-way Omniglot (bottom) using Algorithm 2 applied to Reptile.

FedAvg we found that an initial learning rate of $\eta = 1.0$ worked well—this is similar to those reported in McMahan et al. [2017] and Caldas et al. [2018]—and for the tuned variant we found that a multiplicative decay of 0.99. At meta-test-time we tuned the refinement learning rate over $\{10^{-3}, 10^{-2}, 10^{-1}\}$. For ARUBA and its isotropic variant we set $\varepsilon = \zeta = 0.05$ and $p = 1.0$, so that $\eta = \varepsilon/\zeta = 1.0$ in our setting as well.

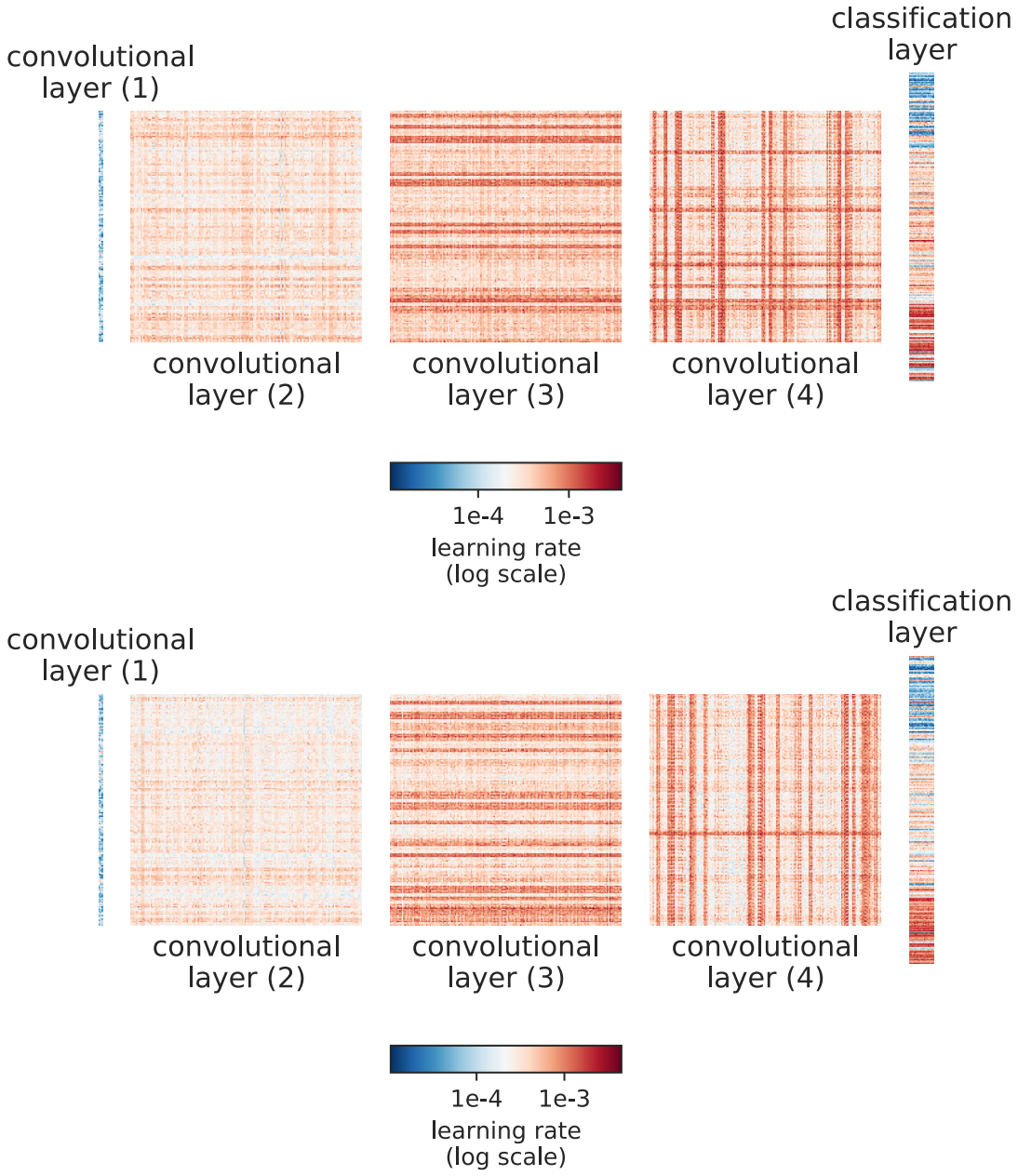

Figure 2.5: Final learning rate η_T across the layers of a CNN trained on 1-shot 20-way Omniglot (top) and 5-shot 20-way Omniglot (bottom) using Algorithm 2 applied to Reptile.

2.B.2 Non-convex meta-learning

Number of training tasks needed for meta-learning

We also examine the number of training tasks that our meta-learning procedure needs to obtain improvements over the single-task baseline. We use a single test task, and a variable number of training tasks (0 through 10) to meta-learn the initialization. We use the same settings as in Section 2.3.4, except the meta-learning experiments have been averaged over 20 iterations (to

Mini-ImageNet	**1-shot**				**5-shot**			
5-way	evaluation setting		hyperparameters		evaluation setting		hyperparameters	
	regular	transductive	$\eta = \frac{\varepsilon}{\zeta}$	c	regular	transductive	$\eta = \frac{\varepsilon}{\zeta}$	c
MAML (1)		48.07 ± 1.75				63.15 ± 0.91		
Reptile	47.07 ± 0.26	49.97 ± 0.32	$1E-3$		62.74 ± 0.37	65.99 ± 0.58	$1E-3$	
ARUBA	47.01 ± 0.37	50.73 ± 0.32	$5E-3$		62.35 ± 0.25	65.69 ± 0.61	$5E-3$	
ARUBA++	47.25 ± 0.61	50.35 ± 0.74	$5E-3$	10	62.69 ± 0.57	65.89 ± 0.34	$5E-3$	10^{-1}
MAML (2)		48.70 ± 1.84				63.11 ± 0.92		
Meta-SGD		50.47 ± 1.87				64.03 ± 0.94		

Table 2.5: Meta-learning evaluations on the 5-way Mini-ImageNet classification task.

average over randomization in the algorithms). In Figure 2.8, we plot the average regret against number of meta-updates performed before starting the test task, and compare against the single-task baselines. We observe gains with meta-learning with just $T = 10$ tasks for the Omniglot dataset, and with even a single task in the Gaussian mixture dataset. The latter is likely due to a very high degree of task similarity across all the tasks (examined below), so learning on any task transfers very well to another task.

Task similarity and dispersion

We also examine the task similarity of the different tasks by plotting the optimal values α_t^* of the clustering parameter α and the corresponding balls $\mathbb{B}(\alpha_t^*, m^{-\beta})$ used in our definition of task similarity (Figure 2.9).

The intervals of the parameter induced by these balls correspond to the discretization used by Algorithm 4. We notice a stronger correlation in task similarity for the Gaussian mixture clustering tasks, which implies that meta-learning is more effective here (both in terms of learning test tasks faster, and with lower regret). For knapsack the task similarity is also high, but it turns out that for our dataset there are very 'sharp peaks' at the optima of the total knapsack values as a function of the parameter ρ. So even though meta-learning helps us get within a small ball of the optima, a few steps are still needed to converge and we do not see the single-shot benefits of meta-learning as we do for the Gaussian clustering experiment.

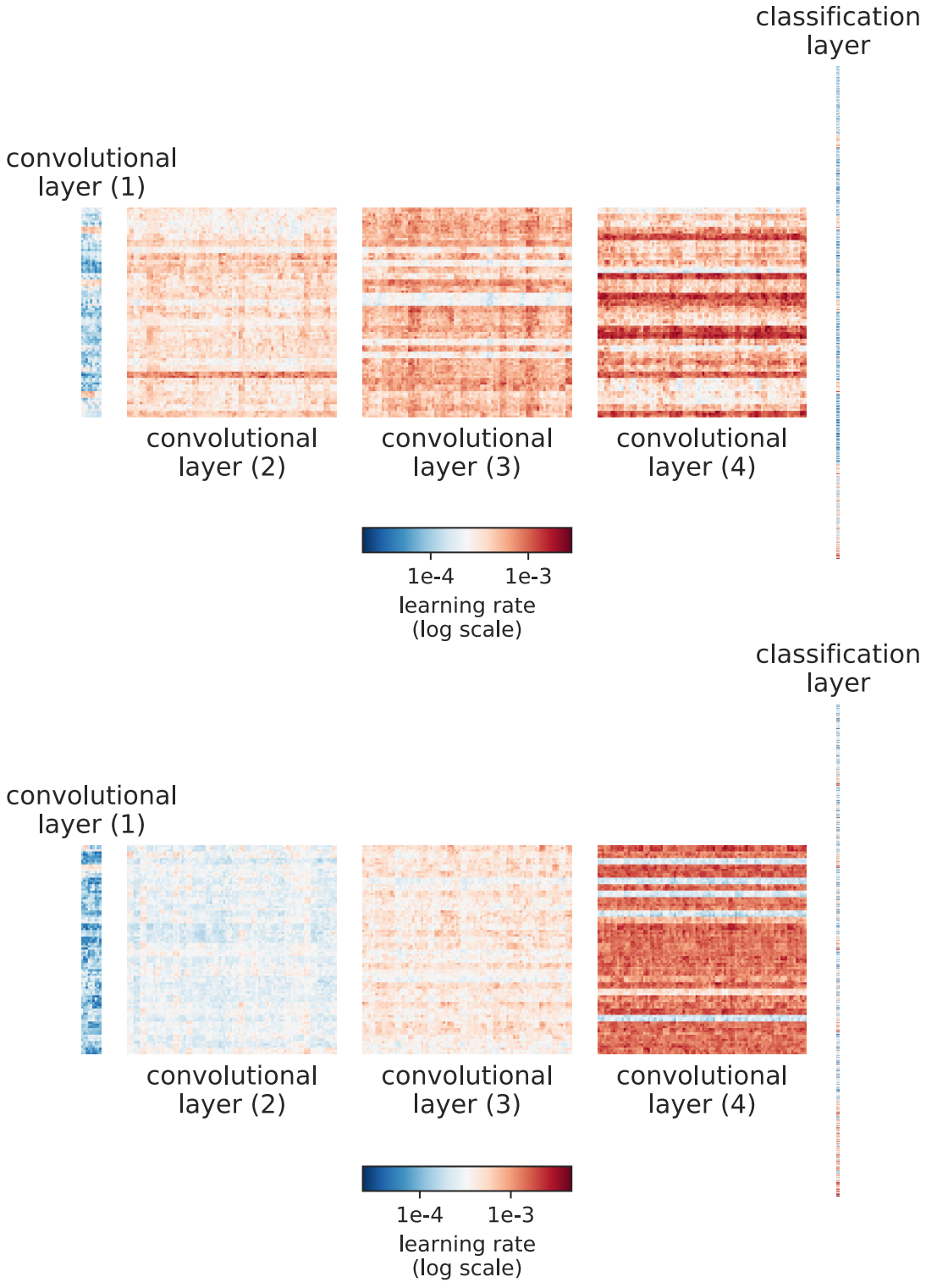

Figure 2.6: Final learning rate η_T across the layers of a CNN trained on 1-shot 5-way Mini-ImageNet (top) and 5-shot 5-way Mini-ImageNet (bottom) using Algorithm 2 applied to Reptile.

Figure 2.7: Final learning rate η_T across the layers of an LSTM trained for next-character prediction on the Shakespeare dataset using Algorithm 2 applied to FedAvg.

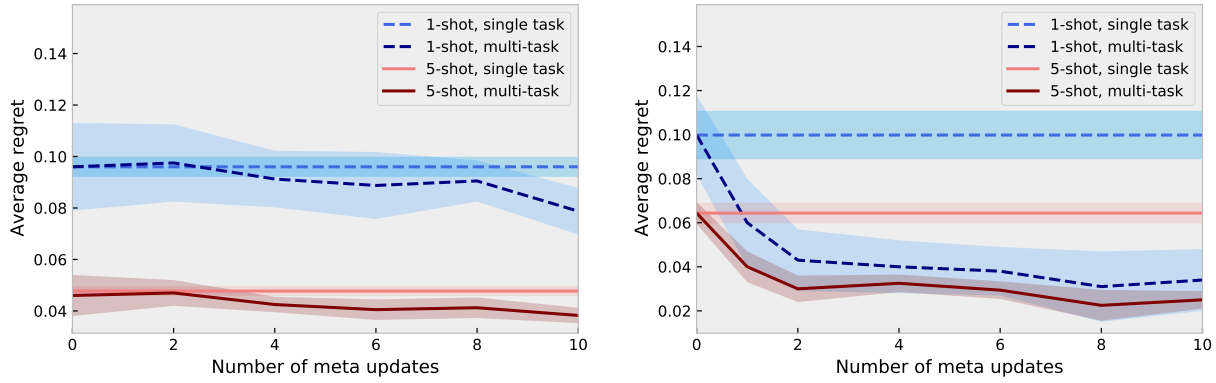

Figure 2.8: Average regret vs. number of training tasks for meta-learning. The clustering data on the left is from Omniglot and on the right it comes a mixture of Gaussians.

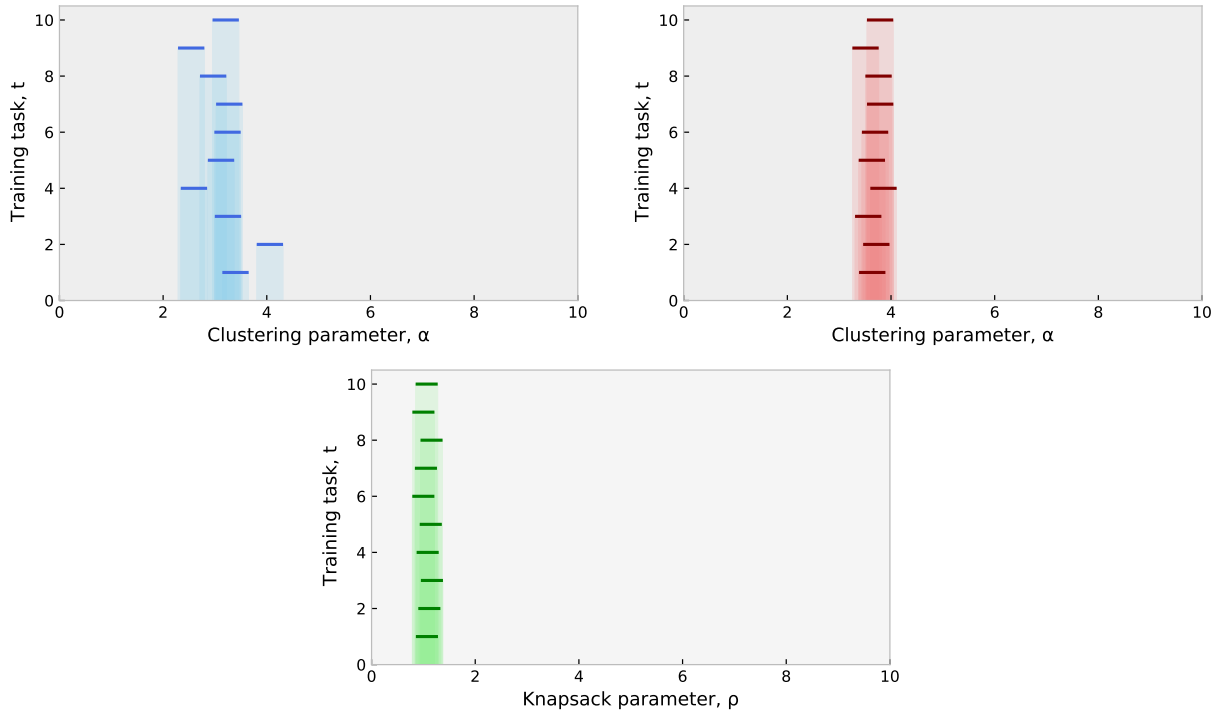

Figure 2.9: Location of optimal parameter values for the training tasks. The left evaluation is for Omniglot clustering, the right for Gaussian mixture clustering, and the bottom is Knapsack.

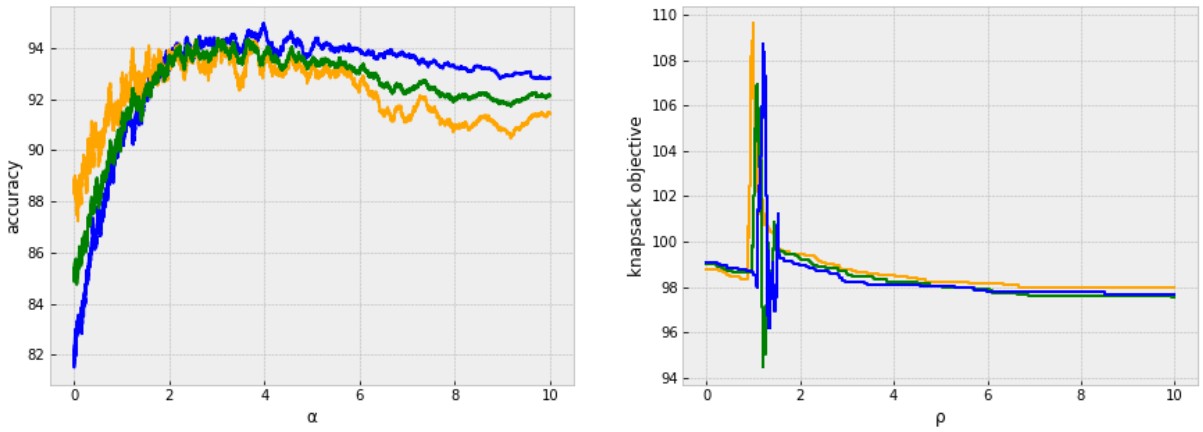

Figure 2.10: Average performance (over algorithm randomization) for a few tasks as a function of the configuration parameter. The left evaluation is Gaussian mixture clustering and the right is Knapsack. This explains why, despite high task similarity in either case, few-shot meta-learning works better for the Gaussian mixture clustering.

Chapter 3

FedEx: Federated hyperparameter tuning

In the previous chapter, we introduced ARUBA, a framework for designing meta-learning algorithms and proving guarantees about them. We then proceeded to study meta-learning in a variety of settings and show many new learning-theoretic guarantees for learning-to-learn across multiple tasks. Now we turn to a more applied study of meta-learning, in which we show how its intersection with federated learning can be exploited to develop hyperparameter tuning algorithms for the latter. At the same time, we still make use of ARUBA to show provable guarantees in a restricted setting for an algorithm we develop called **FedEx**.

3.1 Motivation

Federated learning (FL) is a popular distributed computational setting where training is performed locally or privately on heterogeneous networks [McMahan et al., 2017, Li et al., 2020c] and where hyperparameter tuning has been identified as a critical problem [Kairouz et al., 2021b]. Although general hyperparameter optimization has been the subject of intense study [Hutter et al., 2011, Bergstra and Bengio, 2012, Li et al., 2018a], several unique aspects of the federated setting make tuning hyperparameters especially challenging. We formalize the problem of hyperparameter optimization in FL, introducing the following three key challenges:

1. **Federated validation data:** In federated networks, as the validation data is split across devices, the entire dataset is not available at any one time; instead a central server is given access to some number of devices at each communication round, for one or at most a few runs of local training and validation. Thus, because the standard measure of complexity in FL is the number of communication rounds, computing validation metrics exactly dramatically increases the cost.
2. **Extreme resource limitations:** FL applications often involve training using devices with very limited computational and communication capabilities such as mobile phones. Furthermore, many require the use of privacy techniques such as differential privacy that limit the number times user data can be accessed. Thus we cannot depend on being able to run many different configurations to completion.

[0]The work presented in this chapter first appeared in Khodak et al. [2021].

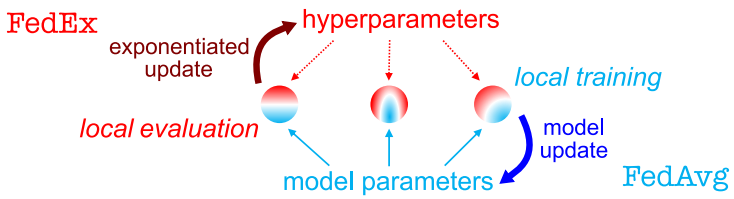

Figure 3.1: FedEx can be applied to any local training-based FL method, e.g. FedAvg, by interleaving standard updates to model weights (computed by aggregating results of local training) with exponentiated gradient updates to hyperparameters (computed by aggregating results of local validation).

3. **Evaluating personalization:** Finally, even with non-federated data, applying common hyperparameter optimization methods to standard personalized FL approaches (such as fine-tuning) can be costly because evaluation may require performing many additional training steps locally.

With these challenges in mind, we propose reasonable baselines for federated hyperparameter tuning by showing how to adapt standard non-federated algorithms. We further study the challenge of noisy validation signal due to federation, and show that simple state-estimation-based fixes do not help.

Our formalization and analysis of this problem leads us to develop FedEx, a method that exploits a novel connection between hyperparameter tuning in FL and the weight-sharing technique widely used in neural architecture search (NAS) [Pham et al., 2018, Liu et al., 2019b, Cai et al., 2019]. In particular, we observe that weight-sharing is a natural way of addressing the three challenges above for federated hyperparameter tuning, as it incorporates noisy validation signal, simultaneously tunes and trains the model, and evaluates personalization as part of training rather than as a costly separate step. Although standard weight-sharing only handles architectural hyperparameters such as the choice of layer or activation, and not critical settings such as those of local stochastic gradient descent (SGD), we develop a formulation that allows us to tune most of these as well via the relationship between local-training and fine-tuning-based personalization. This make FedEx a general hyperparameter tuning algorithm applicable to many local training-based FL methods, e.g. FedAvg [McMahan et al., 2017], FedProx [Li et al., 2020d], and SCAFFOLD [Karimireddy et al., 2020].

In Section 3.5, we next conduct a theoretical study of FedEx in a simple setting: tuning the client step-size. Using our ARUBA framework, we show that a variant of FedEx correctly tunes the on-device step-size to minimize client-averaged regret by adapting to the intrinsic similarity between client data.

Finally, in Section 3.6, we instantiate our baselines and FedEx to tune hyperparameters of FedAvg, FedProx, and Reptile, evaluating on three standard FL benchmarks: Shakespeare, FEMNIST, and CIFAR-10 [McMahan et al., 2017, Caldas et al., 2018]. While our baselines already obtain performance similar to past hand-tuning, FedEx further surpasses them in most settings examined, including by 2-3% on Shakespeare.

3.2 Related work

Several papers have explored limited aspects of hyperparameter tuning in FL [Mostafa and Wang, 2019, Koskela and Honkela, 2018, Dai et al., 2020], focusing on a small number of hyperparameters (e.g. the step-size and sometimes one or two more) in less general settings (studying small-scale problems or assuming server-side validation data). In contrast our methods are able to tune a wide range of hyperparameters in realistic federated networks. Some papers also discussed the challenges of finding good configurations while studying other aspects of federated training [Reddi et al., 2021]. We argue that it is critical to properly address the challenges of federated hyperparameter optimization in practical settings, as we discuss in detail in Section 3.3.

Methodologically, our approach draws on the fact that local training-based methods such as FedAvg can be viewed as optimizing a surrogate objective for personalization, and more broadly leverages the similarity of the personalized FL setup and initialization-based meta-learning [Chen et al., 2018a, Li et al., 2020a, Jiang et al., 2019, Fallah et al., 2020]. While FedEx's formulation and guarantees use this relationship, the method itself is general-purpose and applicable to federated training of a single global model. Many recent papers address FL personalization more directly [Mansour et al., 2020, Yu et al., 2020b, Ghosh et al., 2020, Smith et al., 2017, Li et al., 2021b]. This connection and our use of NAS techniques also makes research connecting NAS and meta-learning relevant [Lian et al., 2020, Elsken et al., 2019b], but unlike these methods we focus on tuning *non*-architectural parameters. In fact, we believe our work is the first to apply weight-sharing to regular hyperparameter search. Furthermore, meta-learning does not have the data-access and computational restrictions of FL, where such methods using the DARTS mixture relaxation [Liu et al., 2019b] are less practical. Instead, FedEx employs the lower-overhead stochastic relaxation [Li et al., 2019, Dong and Yang, 2019], and its exponentiated update is similar to the GAEA algorithm we introduce in Section 9.2.1. Running NAS itself in federated settings has also been studied [Garg et al., 2020, He et al., 2020, Xu et al., 2020a]; while our focus is on non-architectural hyperparameters, in-principle our algorithms can also be used for federated NAS.

3.3 Federated hyperparameter optimization

In this section we formalize the problem of hyperparameter optimization for FL and discuss the connection of its personalized variant to meta-learning. We also review FedAvg [McMahan et al., 2017], a common federated optimization method, and present a reasonable baseline approach for tuning its hyperparameters.

3.3.1 Global and personalized FL

In FL we are concerned with optimizing over a network of heterogeneous clients $i = 1, \dots, n$, each with training, validation, and testing sets T_i, V_i, and E_i, respectively. We use $L_S(\mathbf{w})$ to denote the average loss over a dataset S of some $\mathbf{w}$-parameterized ML model, for $\mathbf{w} \in \mathbb{R}^d$ some real vector. For hyperparameter optimization, we assume a class of algorithms $\texttt{Alg}_a$ hyperparameterized by $a \in \mathcal{A}$ that use *federated access* to training sets T_i to output some element of $\mathbb{R}^d$.

Here by "federated access" we mean that each iteration corresponds to a *communication round* at which $\texttt{Alg}_a$ has access to a batch of B clients[1] that can do local training and validation.

Specifically, we assume $\texttt{Alg}_a$ can be described by two subroutines with hyperparameters encoded by $b \in \mathbb{B}$ and $c \in \mathcal{C}$, so that $a = (b, c)$ and $\mathcal{A} = \mathbb{B} \times \mathcal{C}$. Here c encodes settings of a local training algorithm $\texttt{Loc}_c$ that take a training set S and initialization $\mathbf{w} \in \mathbb{R}^d$ as input and outputs a model $\texttt{Loc}_c(S, \mathbf{w}) \in \mathbb{R}^d$, while b sets those of an aggregation $\texttt{Agg}_b$ that takes the initialization $\mathbf{w}$ and outputs of $\texttt{Loc}_c$ as input and returns a model parameter. For example, in standard FedAvg, $\texttt{Loc}_c$ is T steps of gradient descent with step-size η and $\texttt{Agg}_b$ takes a weighted average of the outputs of $\texttt{Loc}_c$ across clients; here $c = (\eta, T)$ and $b = ()$. As detailed in Appendix 3.B, many FL methods can be decomposed this way, including well-known ones such as FedAvg [McMahan et al., 2017], FedProx [Li et al., 2020d], SCAFFOLD [Karimireddy et al., 2020], and Reptile [Nichol et al., 2018] as well as more recent methods [Li et al., 2021b, Al-Shedivat et al., 2021, Acar et al., 2021]. Our analysis and our proposed FedEx algorithm will thus apply to all of them, up to an assumption detailed next.

Starting from this decomposition, the global hyperparameter optimization problem can be written as

$$\min_{a \in \mathcal{A}} \quad \sum_{i=1}^{n} |V_i| L_{V_i}(\texttt{Alg}_a(\{T_j\}_{j=1}^n)) \tag{3.1}$$

In many cases we are also interested in obtaining a device-specific local model, where we take a model trained on all clients and finetune it on each individual client before evaluating. A key assumption we make is that the finetuning algorithm will be the same as the local training algorithm $\texttt{Loc}_c$ used by $\texttt{Alg}_a$. This assumption can be justified via our work on meta-learning in the previous chapter, where we saw that algorithms that aggregate the outputs of local SGD can be viewed as optimizing for personalization using local SGD. Then, in the personalized setting, the tuning objective becomes

$$\min_{a=(b,c) \in \mathcal{A}} \quad \sum_{i=1}^{n} |V_i| L_{V_i}(\texttt{Loc}_c(T_i, \texttt{Alg}_a(\{T_j\}_{j=1}^n)) \tag{3.2}$$

Our approach will focus on the setting where the hyperparameters c of local training make up a significant portion of all hyperparameters $a = (b, c)$; by considering the personalization objective we will be able to treat such hyperparameters as architectural and thus apply weight-sharing.

3.3.2 Tuning FL methods: Challenges and baselines

In the non-federated setting, the objective (3.1) is amenable to regular hyperparameter optimization methods; for example, a random search approach would repeatedly sample a setting a from some distribution over $\mathcal{A}$, run $\texttt{Alg}_a$ to completion, and evaluate the objective, saving the best setting and output [Bergstra and Bengio, 2012]. With a reasonable distribution and enough samples this is guaranteed to converge and can be accelerated using early stopping methods [Li et al., 2018a], in which $\texttt{Alg}_a$ is not always run to completion if the desired objective is poor at intermediate stages, or by adapting the sampling distribution using the results of previous objective

[1]For simplicity the number of clients per round is fixed, but all methods can be easily generalized to varying B.

Algorithm 7: Successive halving algorithm (SHA) applied to personalized FL. For the non-personalized objective (3.1), replace $L_{V_{ti}}(\mathbf{w}_i)$ by $L_{V_{ti}}(\mathbf{w}_a)$. For random search (RS) with N samples, set $\eta = N$ and $R = 1$.

Input: distribution $\mathcal{D}$ over hyperparameters $\mathcal{A}$, elimination rate $\eta \in \mathbb{N}$, elimination rounds $\tau_0 = 0, \tau_1, \ldots, \tau_R$

sample set of η^R hyperparameters $H \sim \mathcal{D}^{[\eta^R]}$

initialize a model $\mathbf{w}_a \in \mathbb{R}^d$ for each $a \in H$

for *elimination round* $r \in [R]$ **do**

  for *setting* $a = (b, c) \in H$ **do**

    for *comm. round* $t = \tau_{r-1} + 1, \ldots, \tau_r$ **do**

      for *client* $i = 1, \ldots, B$ **do**

        send $\mathbf{w}_a, c$ to client

        $\mathbf{w}_i \leftarrow \texttt{Loc}_c(T_{ti}, \mathbf{w}_a)$

        send $\mathbf{w}_i, L_{V_{ti}}(\mathbf{w}_i)$ to server

      $\mathbf{w}_a \leftarrow \texttt{Agg}_b(\mathbf{w}_a, \{\mathbf{w}_i\}_{i=1}^B)$

      $s_a \leftarrow \sum_{i=1}^B |V_{ti}| L_{V_{ti}}(\mathbf{w}_i) / \sum_{i=1}^B |V_{ti}|$

  $H \leftarrow \{a \in H : s_a \leqslant \frac{1}{\eta}\text{-quantile}(\{s_a : a \in H\})\}$

Output: remaining $a \in H$ and associated model $\mathbf{w}_a$

evaluations [Snoek et al., 2012]. As mentioned in the introduction, applying such methods to FL is inherently challenging due to

1. **Federated validation data:** Separating data across devices means we cannot immediately get a good estimate of the model's validation performance, as we only have access to a possibly small batch of devices at a time. This means that decisions such as which models to flag for early stopping will be noisy and may not fully incorporate all the available validation signal.
2. **Extreme resource limitations:** As FL algorithms can take a very long time to run in practice due to the weakness and spotty availability of devices, we often cannot afford to conduct many training runs to evaluate different configurations. This issue is made more salient in cases where we use privacy techniques that only allow a limited number of accesses to the data of any individual user.
3. **Evaluating personalization:** While personalization is important in FL due to client heterogeneity, checking the performance of the current model on the personalization objective (3.2) is computationally intensive because computing may require running local training multiple times. In particular, while regular validation losses require computing one forward pass per data point, personalized losses require several forward-backward passes, making it many times more expensive if this loss is needed to make a tuning decision such as eliminating a configuration from consideration.

Despite these challenges, we can still devise sensible baselines for tuning hyperparameters in FL, most straightforward of which is to use a regular hyperparameter method but use validation data from a single round as a noisy surrogate for the full validation objective. Specifically,

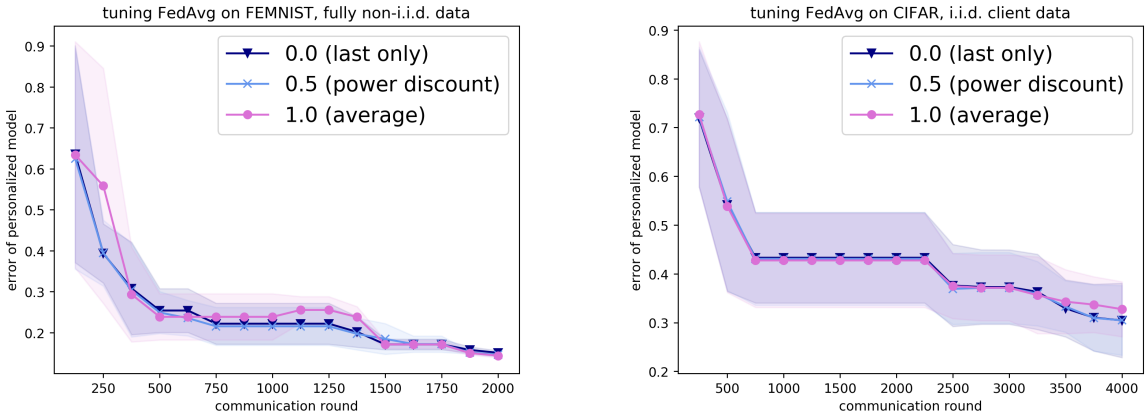

Figure 3.2: Tuning FL with SHA but making elimination decisions based on validation estimates from different discount factors. On both FEMNIST (left) and CIFAR (right) using more of the validation data does not improve upon just using the most recent round's validation error.

one can use random search (RS)—repeatedly evaluate random configurations—and a simple generalization called successive halving (SHA), in which we sample a set of configurations and partially run all of them for some number of communication rounds before eliminating all but the best $\frac{1}{\eta}$ fraction, repeating until only one configuration remains. Note both are equivalent to a "bracket" in Hyperband [Li et al., 2018a] and their adaptation to FL is detailed in Algorithm 7.

As shown in Section 3.6, SHA performs reasonably well on the benchmarks we consider. However, by using validation data from one round it may make noisy elimination decisions, early-stopping potentially good configurations because of a difficult set of clients on a particular round. Here the problem is one of insufficient utilization of the validation data to estimate model performance. A reasonable approach to use more is to try some type of state-estimation: using the performance from previous rounds to improve the noisy measurement of the current one. For example, instead of using only the most recent round for elimination decisions we can use a weighted sum of the performances at all past rounds. To investigate this, we study a power decay weighting, where a round is discounted by some constant factor for each time step it is in the past. We consider factors 0.0 (taking the most recent performance only, as before), 0.5, and 1.0 (taking the average). However, in Figure 3.2 we show that incorporating more validation data this way than is used by Algorithm 7 by default does not significantly affect results.

Thus we may need a better algorithm to use more of the validation signal, most of which is discarded by using the most recent round's performance. We next propose FedEx, a new method that does so by using validation on each round to update a client hyperparameters distribution used to sample configurations to send to devices. Thus it alleviates issue (1) above by updating at each step, not waiting for an elimination round as in RS or SHA. By simultaneously training the model and tuning (client) hyperparameters, it also moves towards a fully single-shot procedure in which we only train once (we must still run multiple times due to server hyperparameters), which would solve issue (2). Finally, FedEx addresses issue (3) by using local training to both update the model and to estimate personalized validation loss, thus not spending extra computation on this more expensive objective.

3.4 Weight-sharing for federated learning

We now present FedEx, a way to tune local FL hyperparameters. This section contains the general algorithm and its connection to weight-sharing; we instantiate it on several FL methods in Section 3.6.

3.4.1 Weight-sharing for architecture search

We first lightly introduce the weight-sharing approach in NAS[2], which for a set $\mathcal{C}$ of network configurations is often posed as the bilevel optimization

$$\min_{c \in \mathcal{C}} \; L_{\text{valid}}(\mathbf{w}, c) \quad \text{s.t.} \quad \mathbf{w} \in \arg\min_{\mathbf{u} \in \mathbb{R}^d} \; L_{\text{train}}(\mathbf{u}, c) \tag{3.3}$$

where $L_{\text{train}}, L_{\text{valid}}$ evaluate a single configuration with the given weights. If, as in NAS, all hyperparameters are architectural, then they are effectively themselves trainable model parameters, so we could instead consider solving the following "single-level" empirical risk minimization (ERM):

$$\min_{c \in \mathcal{C}, \mathbf{w} \in \mathbb{R}^d} \; L(\mathbf{w}, c) \quad = \quad \min_{c \in \mathcal{C}, \mathbf{w} \in \mathbb{R}^d} \; L_{\text{train}}(\mathbf{w}, c) + L_{\text{valid}}(\mathbf{w}, c) \tag{3.4}$$

Solving this instead of the bilevel problem (3.3) has been proposed in several recent papers [Li et al., 2019], including our own work in Section 9.2.1.

Early approaches to solving either formulation of NAS were costly due to the need for full or partial training of many architectures in a very large search space. The weight-sharing paradigm [Pham et al., 2018] reduces the problem to that of training a single architecture, a "supernet" containing all architectures in the search space $\mathcal{C}$. A straightforward way of constructing a supernet is via a "stochastic relaxation" where the loss is an expectation w.r.t. sampling c from some distribution over $\mathcal{C}$ [Dong and Yang, 2019]. Then the shared weights can be updated using SGD by first sampling an architecture c and using an unbiased estimate of $\nabla_{\mathbf{w}} L(\mathbf{w}, c)$ to update $\mathbf{w}$. The distribution over $\mathcal{C}$ may itself be adapted or stay fixed. We focus on the former case, adapting some $\boldsymbol{\theta}$-parameterized distribution $\mathcal{D}_{\boldsymbol{\theta}}$; this yields the stochastic relaxation objective

$$\min_{\boldsymbol{\theta} \in \Theta, \mathbf{w} \in \mathbb{R}^d} \; \mathbb{E}_{c \sim \mathcal{D}_{\boldsymbol{\theta}}} L(\mathbf{w}, c) \tag{3.5}$$

Since architectural hyperparameters are often discrete decisions, e.g. a choice of which of a fixed number of operations to use, a natural choice of $\mathcal{D}_{\boldsymbol{\theta}}$ is as a product of categorical distributions over simplices. In this case, any discretization of an optimum $\boldsymbol{\theta}$ of the relaxed objective (3.5) whose support is in the support of $\boldsymbol{\theta}$ will be an optimum of the original objective (3.4). A natural update scheme here is exponentiated gradient [Kivinen and Warmuth, 1997], where each successive $\boldsymbol{\theta}$ is proportional to $\boldsymbol{\theta} \odot \exp(-\eta \tilde{\nabla})$, η is a step-size, and $\tilde{\nabla}$ an unbiased estimate of $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{c \sim \mathcal{D}_\theta} L(\mathbf{w}, c)$ that can be computed using the re-parameterization trick [Rubinstein and Shapiro, 1993]. By alternating this exponentiated update with the standard SGD update to $\mathbf{w}$ discussed earlier we obtain a simple block-stochastic minimization scheme that is guaranteed to converge, under certain conditions, to the ERM objective (c.f. Section 9.2.1).

[2] A more detailed and NAS-focused introduction is given in Section 8.A.3.

3.4.2 The FedEx method

To obtain FedEx from weight-sharing we restrict to the case of tuning only the hyperparameters c of local training $\texttt{Loc}_c$.[3] Our goal then is just to find the best initialization $\mathbf{w} \in \mathbb{R}^d$ and local hyperparameters $c \in \mathcal{C}$, i.e. we replace the personalized objective (3.2) by

$$\min_{c \in \mathcal{C}, \mathbf{w} \in \mathbb{R}^d} \quad \sum_{i=1}^{n} |V_i| L_{V_i}(\texttt{Loc}_c(T_i, \mathbf{w})) \tag{3.6}$$

Note $\texttt{Alg}_a$ outputs an element of $\mathbb{R}^d$, so this new objective is upper-bounded by the original (3.2), i.e. any solution will be at least as good for the original objective. Note also that for fixed c this is equivalent to the classic train-validation split objective for meta-learning with $\texttt{Loc}_c$ as the base learner. More importantly for us, it is also in the form of the r.h.s. of the weight-sharing objective (3.4), i.e. it is a single-level function of $\mathbf{w}$ and c. We thus apply a NAS-like stochastic relaxation:

$$\min_{\boldsymbol{\theta} \in \Theta, \mathbf{w} \in \mathbb{R}^d} \quad \sum_{i=1}^{n} |V_i| \mathbb{E}_{c \in \mathcal{D}_{\boldsymbol{\theta}}} L_{V_i}(\texttt{Loc}_c(T_i, \mathbf{w})) \tag{3.7}$$

In NAS we would now set the distribution to be a product of categorical distributions over different architectures, thus making $\boldsymbol{\theta}$ an element of a product of simplices and making the optimum of the original objective (3.6) equivalent to the optimum of the relaxed objective (3.7) as an extreme point of the simplex. Unlike in NAS, FL hyperparameters such as the learning rate are not extreme points of a simplex and so it is less clear what parameterized distribution $\mathcal{D}_{\boldsymbol{\theta}}$ to use. Nevertheless, we find that crudely imposing a categorical distribution over $k > 1$ random samples from some distribution (e.g. uniform) over $\mathcal{C}$ and updating $\boldsymbol{\theta}$ using exponentiated gradient over the resulting k-simplex works well. We alternate this with updating $\mathbf{w} \in \mathbb{R}^d$, which in a NAS algorithm involves an SGD update using an unbiased estimate of the gradient at the current $\mathbf{w}$ and $\boldsymbol{\theta}$.

We call this alternating method for solving (3.7) FedEx and describe it for a general $\texttt{Alg}_a$ consisting of sub-routines $\texttt{Agg}_b$ and $\texttt{Loc}_c$ in Algorithm 8; recall from Section 3.3 that many FL methods can be decomposed this way, so our approach is widely applicable. FedEx has a minimal overhead, consisting only of the last four lines of the outer loop updating $\boldsymbol{\theta}$. Thus, as with weight-sharing, FedEx can be viewed as reducing the complexity of tuning local hyperparameters to that of training a single model. Each update to $\boldsymbol{\theta}$ requires a step-size η_t and an approximation $\tilde{\nabla}$ of the gradient w.r.t. $\boldsymbol{\theta}$; for the latter we obtain an estimate $\tilde{\nabla}_{\boldsymbol{\theta}[j]}$ of each gradient entry via the reparameterization trick, whose variance we reduce by subtracting a baseline λ_t. How we set η_t and λ_t is detailed in Appendices 3.C.2 and 3.E, respectively.

To see how FedEx is approximately optimizing the relaxed objective (3.7), we can consider the case where $\texttt{Alg}_a$ is Reptile [Nichol et al., 2018], which was designed to optimize some approximation of (3.6) for fixed c, or equivalently the relaxed objective for an atomic distribution $\mathcal{D}_{\boldsymbol{\theta}}$. As we discussed in the previous chapter, Reptile can be interpreted as optimizing a surrogate objective minimizing the squared distance between $\mathbf{w}$ and the optimum of each task i, with the latter being replaced by the last iterate in practice. It is also shown that the surrogate objective

[3] We will use some wrapper algorithm to tune the hyperparameters b of $\texttt{Agg}_b$.

Algorithm 8: FedEx

Input: configurations $c_1, \ldots, c_k \in \mathcal{C}$, setting b for $\texttt{Agg}_b$, schemes for setting step-size η_t and baseline λ_t, total number of steps $\tau \geqslant 1$

initialize $\theta_1 = \mathbf{1}_k/k$ and shared weights $\mathbf{w}_1 \in \mathbb{R}^d$

for *comm. round* $t = 1, \ldots, \tau$ **do**

  for *client* $i = 1, \ldots, B$ **do**

    send $\mathbf{w}_t, \boldsymbol{\theta}_t$ to client

    sample $c_{ti} \sim \mathcal{D}_{\theta_t}$

    $\mathbf{w}_{ti} \leftarrow \texttt{Loc}_{c_{ti}}(T_{ti}, \mathbf{w}_t)$

    send $\mathbf{w}_{ti}, c_{ti}, L_{V_{ti}}(\mathbf{w}_{ti})$ to server

  $\mathbf{w}_{t+1} \leftarrow \texttt{Agg}_b(\mathbf{w}, \{\mathbf{w}_{ti}\}_{i=1}^B)$

  $\tilde{\nabla}_j \leftarrow \frac{\sum_{i=1}^B |V_{ti}|(L_{V_{ti}}(\mathbf{w}_{ti}) - \lambda_t)\mathbf{1}_{c_{ti}=c_j}}{\boldsymbol{\theta}_{t[j]} \sum_{i=1}^B |V_{ti}|} \; \forall \; j$

  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t \odot \exp(-\eta_t \tilde{\nabla})$

  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_{t+1}/\|\boldsymbol{\theta}_{t+1}\|_1$

Output: model $\mathbf{w}$, hyperparameter distribution $\boldsymbol{\theta}$

is useful for personalization in the online convex setting.[4] As opposed to this past work, FedEx makes two gradient updates in the outer loop, on two disjoint sets of variables: the first is the sub-routine $\texttt{Agg}_b$ of $\texttt{Alg}_a$ that aggregates the outputs of local training and is using the gradient of the surrogate objective, since the derivative of the squared distance is the difference between the initialization $\mathbf{w}$ and the parameter at the last iterate of $\texttt{Loc}_c$; the second is the exponentiated gradient update that is directly using an unbiased estimate of the derivative of the second objective w.r.t. the distribution parameters $\boldsymbol{\theta}$. Thus, roughly speaking FedEx runs simultaneous stochastic gradient descent on the relaxed objective (3.7), although for the variables $\mathbf{w}$ we are using a first-order surrogate. In the theoretical portion of chapter we employ this interpretation to show the approach works for tuning the step-size of online gradient descent in the online convex optimizations setting.

3.4.3 Wrapping FedEx

We can view FedEx as an algorithm of the form tuned by Algorithm 7 that implements federated training of a supernet parameter $(\mathbf{w}, \boldsymbol{\theta})$, with the local training routine $\texttt{Loc}$ including a step for sampling $c \sim \mathcal{D}_{\boldsymbol{\theta}}$ and the server aggregation routine including an exponentiated update of $\boldsymbol{\theta}$. Thus we can wrap FedEx in Algorithm 7, which we find useful for a variety of reasons:

- The wrapper can tune the settings of b for the aggregation step $\texttt{Agg}_b$, which FedEx cannot.
- FedEx itself has a few hyperparameters, e.g. how to set the baseline λ_t, which can be tuned.
- By running multiple seeds and potentially using early stopping, we can run FedEx using more aggressive steps-sizes and the wrapper will discard cases where this leads to poor results.

[4] Formally they study a sequence of upper bounds and not a surrogate objective, as their focus is online learning.

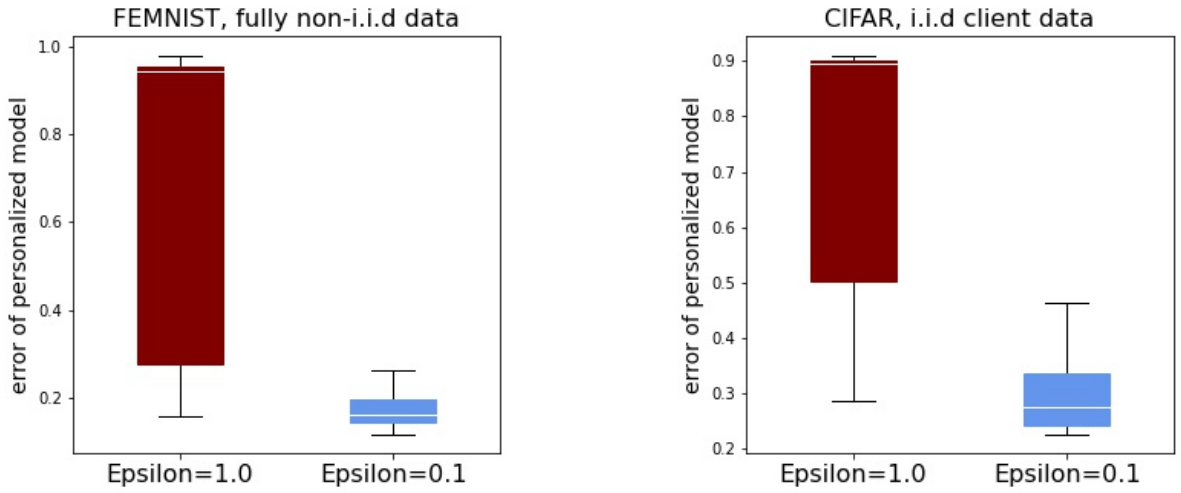

Figure 3.3: Comparison of the range of performance values attained using different perturbation settings. Although the range is much smaller for $\epsilon = 0.1$ than for $\epsilon = 1.0$ (the latter is the entire space), it still covers a large (roughly 10-20%) range of different performance levels on both FEMNIST (left) and CIFAR (right).

- We can directly compare FedEx to a regular hyperparameter optimization scheme run over the original algorithm, e.g. FedAvg, by using the same scheme to both wrap FedEx and tune FedAvg.
- Using the wrapper allows us to determine the configurations $c_1, \ldots, c_k$ given to Algorithm 8 using a local perturbation scheme (detailed next) while still exploring the entire hyperparameter space.

3.4.4 Local perturbation

It remains to specify how to select the configurations $c_1, \ldots, c_k \in \mathcal{C}$ to pass to Algorithm 8. While the simplest approach is to draw from $\mathrm{Unif}^k(\mathcal{C})$, we find that this leads to unstable behavior if the configurations are too distinct from each other. To interpolate between sampling c_i independently and setting them to be identical (which would just be equivalent to the baseline algorithm), we use a simple *local perturbation* method in which c_1 is sampled from $\mathrm{Unif}(\mathcal{C})$ and $c_2, \ldots, c_k$ are sampled uniformly from a local neighborhood of $\mathcal{C}$. For continuous hyperparameters (e.g. step-size, dropout) drawn from an interval $[a, b] \subset \mathbb{R}$ the local neighborhood is $[c \pm (b-a)\varepsilon]$ for some $\varepsilon \geqslant 0$, i.e. a scaled ε-ball; for discrete hyperparameters (e.g. batch-size, epochs) drawn from a set $\{a, \ldots, b\} \subset \mathbb{Z}$, the local neighborhood is similarly $\{c - \lfloor (b-a)\varepsilon \rfloor, \ldots, c + \lceil (b-a)\varepsilon \rceil\}$; in our experiments we set $\varepsilon = 0.1$, which works well, but run ablation studies varying these values in Appendix 3.E showing that a wide range of them leads to improvement. Note that while local perturbation does limit the size of the search space explored by each instance of FedEx, as shown in Figure 3.3 the difference in performance between different configurations in the same ball is still substantial.

3.4.5 Limitations of FedEx

While FedEx is applicable to many important FL algorithms, those that cannot be decomposed into local fine-tuning and aggregation should instead be tuned by one of our baselines, e.g. SHA. FedEx is also limited in that it is forced to rely on such algorithms as wrappers for tuning its own hyperparameters and certain FL hyperparameters such as server learning rate.

3.5 Theoretical analysis for tuning the step-size

As noted in Section 3.4, FedEx can be viewed as alternating minimization, with a gradient step on a surrogate personalization loss and an exponentiated gradient update of the configuration distribution θ. We make this formal and prove guarantees for a simple variant of FedEx in the setting where the server has one client per round, to which the server sends an initialization to solve an online convex optimization (OCO) problem using online gradient descent (OGD) on a sequence of m adversarial convex losses (i.e. one SGD epoch in the stochastic case). Note we use "client" and "task" interchangeably, as the goal is a meta-learning (personalization) result. As in Chapter 2, our performance measure will be *task-averaged regret*, which takes the average over τ clients of the regret they incur on its loss:

$$\overline{\text{Regret}} = \frac{1}{\tau}\sum_{t=1}^{\tau}\sum_{i=1}^{m} \ell_{t,i}(\mathbf{w}_{t,i}) - \ell_{t,i}(\mathbf{w}_t^*) \tag{3.8}$$

Here $\ell_{t,i}$ is the ith loss of client t, $\mathbf{w}_{t,i}$ the parameter chosen on its ith round from a compact parameter space $\mathcal{W}$, and $\mathbf{w}_t^* \in \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i=1}^{m} \ell_{t,i}(\mathbf{w})$ the task optimum. In this setting, our ARUBA framework can be used to show guarantees for a Reptile (i.e. FedEx with a server step-size) variant in which at each round the initialization is updated as $\mathbf{w}_{t+1} \leftarrow (1-\alpha_t)\mathbf{w}_t + \alpha_t \mathbf{w}_t^*$ for server step-size $\alpha_t = 1/t$; observe that the only difference between this update and FedEx's is that the task t optimum $\mathbf{w}_t^*$ is used rather than the last iterate of OGD on that task. Specifically we bound the task-averaged regret by

$$\overline{\text{Regret}} \leqslant \tilde{\mathcal{O}}\left(\frac{1}{\sqrt[4]{\tau}} + V\right)\sqrt{m} \quad \text{for} \quad V^2 = \min_{\mathbf{w}\in\mathcal{W}} \frac{1}{\tau}\sum_{t=1}^{\tau} \|\mathbf{w} - \mathbf{w}_t^*\|_2^2 \tag{3.9}$$

Here V—the average deviation of the optimal actions $\mathbf{w}_t^*$ across tasks—is a measure of *task similarity*: V is small when the tasks (clients) have similar data and thus can be solved by similar parameters in $\mathcal{W}$ but large when their data is different and so the optimum parameters to use are very different. Thus the bound in (2.3) shows that as the server (meta-learning) sees more and more clients (tasks), their regret on each decays with rate $\mathcal{O}(1/\sqrt[4]{\tau})$ to depend only on the task similarity, which is hopefully small if the client data is similar enough that transfer learning makes sense, in particular if $V \ll \text{diam}(\mathcal{W})$. Since single-task regret has lower bound $\Omega(D\sqrt{m})$, achieving asymptotic regret $V\sqrt{m}$ thus demonstrates successful learning of a useful initialization in $\mathcal{W}$ that can be used for personalization. Recall that such bounds can also be converted to obtain guarantees in the statistical meta-learning setting as well (c.f. Section 2.2.3).

A drawback of our past results using the ARUBA framework is that they either assume the task similarity V is known in order to set the client step-size or they employ an OCO method to

learn the local step-size that cannot be applied to other potential algorithmic hyperparameters. In contrast, we prove results for using bandit exponentiated gradient to tune the client step-size, which is precisely the FedEx update. In particular, Theorem 3.5.1 shows that by using a discretization of potential client step-sizes as the configurations in Algorithms 8 we can obtain the following task-averaged regret:

Theorem 3.5.1. Let $\mathcal{W} \subset \mathbb{R}^d$ be convex and compact with diameter $D = \operatorname{diam}(\mathcal{W})$ and let $\ell_{t,i}$ be a sequence of $m\tau$ b-bounded convex losses—m for each of τ tasks—with Lipschitz constant $\leqslant G$. We assume that the adversary is oblivious within-task. Suppose we run Algorithm 8 with $B = 1$, configurations $c_j = \frac{D}{Gj\sqrt{m}}$ for each $j = 1, \ldots, k$ determining the local step-size of single-epoch SGD (OGD), $\mathbf{w}_{ti} = \mathbf{w}_t^*$, regret $\sum_{i=1}^m \ell_{t,i}(\mathbf{w}_{t,i}) - \ell_{t,i}(\mathbf{w}_t)$ used in place of $L_{V_{ti}}(\mathbf{w}_{ti})$, and $\lambda_t = 0 \ \forall \ t \in [\tau]$. Then if $\eta_t = \frac{1}{mb}\sqrt{\frac{\log k}{k\tau}} \ \forall \ t \in [\tau]$, $k^{\frac{3}{2}} = \frac{DG}{b}\sqrt{\frac{\tau}{2m}}$, and $\texttt{Agg}_b(\mathbf{w}, \mathbf{w}_t^*) = (1-\alpha_t)\mathbf{w} + \alpha_t \mathbf{w}_t^*$ for $\alpha_t = 1/t \ \forall \ t \in [\tau]$ we have (taking expectations over sampling from D_{θ_t})

$$\mathbb{E}\overline{\text{Regret}} \leqslant \tilde{\mathcal{O}}\left(\sqrt[3]{m/\tau} + V\right)\sqrt{m} \tag{3.10}$$

The proof of this result, given in the supplement, follows the ARUBA framework of using meta OCO algorithm to optimize the initialization-dependent upper bound on the regret of OGD; in addition we bound errors to the bandit setting and discretization of the step-sizes. Theorem 3.5.1 demonstrates that FedEx is a sensible algorithm for tuning the step-size in the meta-learning setting where each task is an OCO problem, with the average regret across tasks (clients) converging to depend only on the task similarity V, which we hope is small in the setting where personalization is useful. As we can see by comparing to the bound in (2.3), besides holding for a more generally-applicable algorithm our bound also improves the dependence on τ, albeit at the cost of an additional $m^{\frac{1}{3}}$ factor. Note that that the sublinear term can be replaced by $1/\sqrt{\tau}$ in the full-information setting, i.e. where required the client to try SGD with each configuration c_j at each round to obtain regret for all of them.

3.6 Empirical results

In our experiments, we instantiate FedEx on the problem of tuning FedAvg, FedProx, and Reptile; the first is the most popular algorithm for federated training, the second is an extension designed for heterogeneous devices, and the last is a compatible meta-learning method used for learning initializations for personalization. At communication round t these algorithms use the aggregation

$$\texttt{Agg}_b(\mathbf{w}, \{\mathbf{w}_i\}_{i=1}^B) = (1-\alpha_t)\mathbf{w} + \frac{\alpha_t}{\sum_{i=1}^B |T_{ti}|}\sum_{i=1}^B |T_{ti}|\mathbf{w}_i \tag{3.11}$$

for some learning rate $\alpha_t > 0$ that can vary through time; in the case of $FedAvg$ we have $\alpha_t = 1 \ \forall \ t$. The local training sub-routine $\texttt{Loc}_c$ is SGD with hyperparameters c over some objective defined by the training data T_{ti}, which can also depend on c. For example, to include FedProx we include in c an additional local hyperparameter for the proximal term compared with that of FedAvg.

Table 3.1: Final test error obtained when tuning using a standard hyperparameter tuning algorithm (SHA or RS) alone, or when using it for server (aggregation) hyperparameters while FedEx tunes client (on-device training) hyperparameters. The target model is the one used to compute on-device validation error by the wrapper method, as well as the one used to compute test error after tuning. Note that this table reports the final error results corresponding to the online evaluations reported in Figure 3.4, which measure performance as more of the computational budget is expended.

Wrapper method	Target model	Tuning method	Shakespeare i.i.d.	Shakespeare non-i.i.d.	FEMNIST i.i.d.	FEMNIST non-i.i.d.	CIFAR-10 i.i.d.
Random Search (RS)	global	RS (server & client)	60.32 ± 10.03	64.36 ± 14.19	22.81 ± 4.56	22.98 ± 3.41	30.46 ± 9.44
		+ FedEx (client)	53.94 ± 9.13	57.70 ± 17.57	20.96 ± 4.77	22.30 ± 3.66	34.83 ± 14.74
	personalized	RS (server & client)	61.10 ± 9.32	61.71 ± 9.08	17.45 ± 2.82	17.77 ± 2.63	34.89 ± 10.56
		+ FedEx (client)	54.90 ± 9.97	56.48 ± 13.60	16.31 ± 3.77	15.93 ± 3.06	39.13 ± 15.13
Successive Halving (SHA)	global	SHA (server & client)	47.38 ± 3.40	46.79 ± 3.51	18.64 ± 1.68	20.30 ± 1.66	21.62 ± 2.51
		+ FedEx (client)	$\mathbf{44.52} \pm 1.68$	$\mathbf{45.24} \pm 3.31$	19.22 ± 2.05	19.43 ± 1.45	$\mathbf{20.82} \pm 1.37$
	personalized	SHA (server & client)	46.77 ± 3.61	48.04 ± 3.72	$\mathbf{14.79} \pm 1.55$	14.78 ± 1.31	24.81 ± 6.13
		+ FedEx (client)	46.08 ± 2.57	45.89 ± 3.76	14.97 ± 1.31	$\mathbf{14.76} \pm 1.70$	21.77 ± 2.83

We tune several hyperparameters of both aggregation and local training; for the former we tune the server learning rate schedule and momentum, found to be helpful for personalization [Jiang et al., 2019]; for the latter we tune the learning rate, momentum, weight decay, the number of local epochs, the batch-size, dropout, and proximal regularization. Please see the supplementary material for the exact hyperparameter space considered. While we mainly evaluate FedEx in cross-device federated settings, which is generally more difficult than cross-silo in terms of hyperparameter optimization, FedEx can be naturally applied to cross-silo settings, where the challenges of heterogeneity, privacy requirements, and personalization remain.

Because our baseline is running Algorithm 7, a standard hyperparameter tuning algorithm, to tune all hyperparameters, and because we need to also wrap FedEx in such an algorithm for the reasons described in Section 3.4, our empirical results will test the following question: does FedEx, wrapped by random search (RS) or a successive halving algorithm (SHA), do better than RS or SHA run with the same settings directly? Here "better" will mean both the final test accuracy obtained and the online evaluation setting, which tests how well hyperparameter optimization is doing at intermediate phases. Furthermore, we also investigate whether FedEx can improve upon the wrapper alone even when targeting a good *global* and not personalized model, i.e. when elimination decisions are made using the average global validation loss. We run Algorithm 7 on the personalized objective and use RS and SHA with elimination rate $\eta = 3$, the latter following Hyperband [Li et al., 2018a]. To both wrappers we allocate the same (problem-dependent) tuning budget. To obtain the elimination rounds in Algorithm 7 for SHA, we set the number of eliminations to $R = 3$, fix a total communication round budget, and fix a maximum number of rounds to be allocated to any configuration a; as detailed in Appendix 3.D.1, this allows us to determine $T_1, \ldots, T_R$ so as to use up as much of the budget as possible.

We evaluate the performance of FedEx on three datasets (Shakespeare, FEMNIST, and CIFAR-10) on both vision and language tasks. We consider the following two different partitions of data:

1. **Each device holds i.i.d. data.** While data *across* the network can be non-i.i.d., we shuffle local data *within* each device before splitting into train, validation, and test sets.

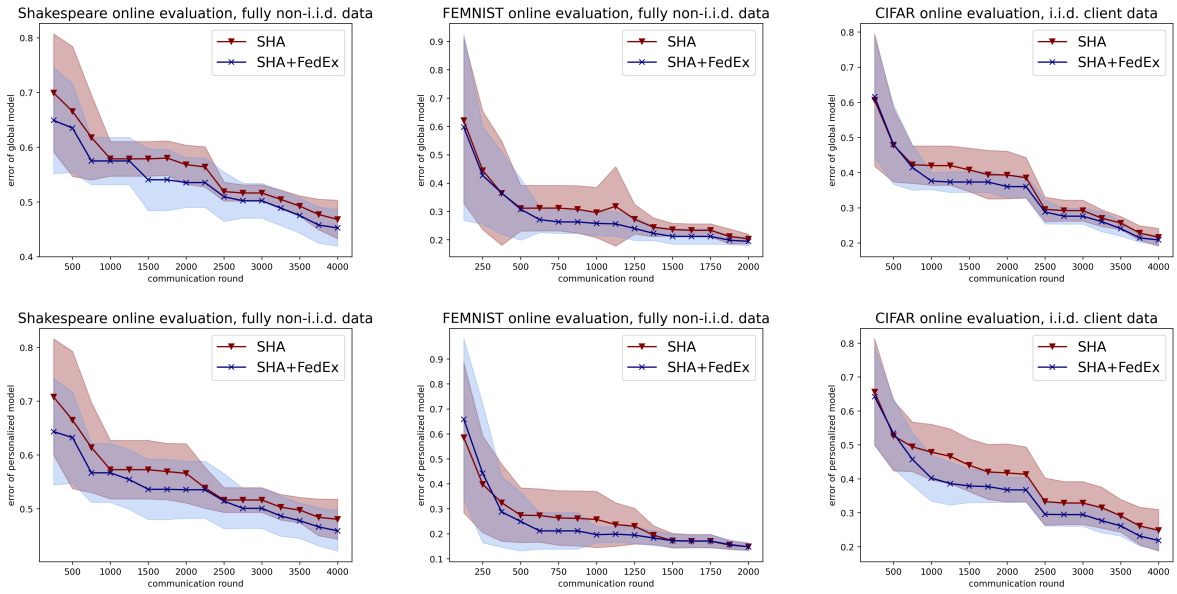

Figure 3.4: Online evaluation of FedEx on the Shakespeare next-character prediction dataset (left), the FEMNIST image classification dataset (middle), and the CIFAR-10 image classification dataset (right) in the fully non-i.i.d. setting (except CIFAR-10). We report global model performance on the top and personalized performance on the bottom. All evaluations are run for three trials.

2. **Each device holds non-i.i.d. data.** In Shakespeare, each device is an actor and local data is split according to temporal position in the play; in FEMNIST, each device is the digit writer and local data is split randomly; in CIFAR-10, we do not study this setting.

For Shakespeare and FEMNIST we use 80% of the data for training and 10% each for validation and testing. In CIFAR-10 we hold out 10K examples from the usual training/testing split for validation. The backbone models used for Shakespeare and CIFAR-10 follow from the FedAvg evaluation [McMahan et al., 2017] and use 4K communications rounds (at most 800 round for each arm), while that of FEMNIST follows from LEAF [Caldas et al., 2018] and uses 2K communication rounds (at most 200 for each arm).

Table 3.1 presents our main results, displaying the final test error of the target model after tuning using either a wrapper algorithm alone or its combination with FedEx. The evaluation shows that using FedEx on the client parameters is either equally or more effective in most cases; in particular, a FedEx-modified method performs best everywhere except i.i.d. FEMNIST, where it is very close. Furthermore, FedEx frequently improves upon the wrapper algorithm by 2 or more percentage points.

We further present online evaluation results in Figure 3.4, where we display the test error of FedEx wrapped with SHA compared to SHA alone as a function of communication rounds. Here we see that for most of training FedEx is either around the same or better then the alternative, except at the beginning; the former is to be expected since the randomness of FedEx leads to less certain updates at initialization. Nevertheless FedEx is usually better than the SHA baseline by the halfway point.

3.7 Conclusion

This concludes our study of the problem of hyperparameter optimization in FL, starting with identifying the key challenges and proposing reasonable baselines that adapts standard approaches to the federated setting. We further make a novel connection to the weight-sharing paradigm from NAS—to our knowledge the first instance of this being used for regular (non-architectural) hyperparameters—and use it to introduce FedEx. This simple, low-overhead algorithm for accelerating the tuning of hyperparameters in federated learning can be theoretically shown to successfully tune the step-size for multi-task OCO problems and effectively tunes FedAvg, FedProx, and Reptile on standard benchmarks. The scope of application of FedEx is very broad, including tuning actual architectural hyperparameters rather than just settings of local SGD, i.e. doing federated NAS, and tuning initialization-based meta-learning algorithms such as Reptile and MAML.

3.A Proof of Theorem 3.5.1

Proof. Let $\gamma_t \sim \mathcal{D}_{\boldsymbol{\theta}_t}$ be the step-size chosen at time t. Then we have that

$$
\begin{aligned}
\tau\mathbb{E}\overline{\mathrm{Regret}} &= \sum_{t=1}^{\tau} \mathbb{E}_{\gamma_t} \sum_{i=1}^{m} \ell_{t,i}\left(\mathbf{w}_{t,i}^{(\mathbf{w}_t,\gamma_t)}\right) - \sum_{i=1}^{m} \ell_{t,i}(\mathbf{w}_t^*) \\
&= \sum_{t=1}^{\tau} \sum_{j=1}^{k} \theta_{t[j]} \sum_{i=1}^{m} \ell_{t,i}\left(\mathbf{w}_{t,i}^{(\mathbf{w}_t,c_j)}\right) - \sum_{i=1}^{m} \ell_{t,i}(\mathbf{w}_t^*) \\
&\leqslant \frac{\log k}{\eta} + \eta k \tau m^2 b^2 \\
&\quad + \min_{j\in[k]} \sum_{t=1}^{\tau} \sum_{i=1}^{m} \ell_{t,i}\left(\mathbf{w}_{t,i}^{(\mathbf{w}_t,c_j)}\right) - \min_{\mathbf{w}\in\mathcal{W}} \sum_{i=1}^{m} \ell_{t,i}(\mathbf{w}_t^*) \\
&\leqslant 2mb\sqrt{\tau k \log k} + \min_{j\in[k]} \sum_{t=1}^{\tau} \frac{1}{2c_j}\|\mathbf{w}_t - \mathbf{w}_t^*\|_2^2 + c_j m G^2 \\
&\leqslant 2mb\sqrt{\tau k \log k} + \min_{j\in[k]} \frac{D^2(1+\log\tau)}{2c_j} + \left(\frac{V^2}{2c_j} + c_j m G^2\right)\tau \\
&\leqslant 2mb\sqrt{\tau k \log k} + \frac{D^2(1+\log\tau) + V^2\tau}{2\gamma^*} + \gamma^* m G^2 \tau \\
&\quad + \min_{j\in[k]} \left(\frac{1}{c_j} - \frac{1}{\gamma^*}\right) \frac{D^2(1+\log\tau) + V^2\tau}{2} + (c_j - \gamma^*) m G^2 \tau \\
&\leqslant 2mb\sqrt{\tau k \log k} + 4D\sqrt{\frac{\tau + \tau\log\tau}{2}} + \left(2V + \frac{D}{k}\right) G\tau\sqrt{\frac{m}{2}} \\
&= mb\sqrt{2\tau\log\tau} + 4D\sqrt{\frac{\tau+\tau\log\tau}{2}} + (DG + 2GV\tau)\sqrt{\frac{m}{2}}
\end{aligned}
\tag{3.12}
$$

where the second line uses linearity of expectations over $\gamma_t \sim \mathcal{D}_{\boldsymbol{\theta}_t}$, the third substitutes the bandit regret of EG [Shalev-Shwartz, 2011, Corollary 4.2], the fourth substitutes $\eta = \frac{1}{mb}\sqrt{\frac{\log k}{\tau k}}$ and the regret of OGD [Shalev-Shwartz, 2011, Corollary 2.7], the fifth substitutes the regret guarantee of Adaptive OGD over functions $\frac{1}{2}\|\mathbf{w}_t - \mathbf{w}_t^*\|_2^2$ [Bartlett et al., 2008, Theorem 2.1] with step-size $\alpha_t = 1/t$ and the definition of V, the sixth substitutes the best discretized step-size c_j for the optimal $\gamma^* \in \left(0, \frac{D}{G\sqrt{2m}}\right]$, and the seventh substitutes $\frac{V}{2G\sqrt{2m}} + \frac{D}{2G}\sqrt{\frac{1+\log\tau}{2m\tau}}$ for γ^* and $\arg\min_{j:c_j\geqslant\gamma^*}$ for $\arg\min_j c_j$. Setting $k^{\frac{3}{2}} = \frac{DG}{b}\sqrt{\frac{\tau}{2m}}$ and dividing both sides by τ yields the result. □

3.B Decomposing federated optimization methods

As detailed in Section 3.3 our analysis and use of FedEx to tune local training hyperparameters depends on a formulation that decomposes FL methods into two subroutines: a local training routine $\texttt{Loc}_c(S, \mathbf{w})$ with hyperparameters c over data S and starting from initialization $\mathbf{w}$ and

an aggregation routine $\texttt{Agg}_b$ with hyperparameters b. In this section we discuss how a variety of federated optimization methods, including several of the best-known, can be decomposed in this manner. This enables the application of FedEx to tune their hyperparameters.

FedAvg [McMahan et al., 2017] The best-known FL method, FedAvg runs SGD on each client in a batch starting from a shared initialization and then updates to the average of the last iterate of the clients, often weighted by the number of data points each client has. The decomposition here is:

$\texttt{Loc}_c$ Local SGD (or another gradient-based algorithm, e.g. Adam [Kingma and Ba, 2015]), with c being the standard hyperparameters such as step-size, momentum, weight decay, etc.

$\texttt{Agg}_b$ Weighted averaging, with no hyperparameters in b.

FedProx [Li et al., 2020d] FedProx has the same decomposition as FedAvg except local SGD is replaced by a proximal version that regularizes the routine to be closer to the initialization, adding another hyperparameter to c governing the strength of this regularization.

Reptile [Nichol et al., 2018] A well-known meta-learning algorithm, Reptile has the same decomposition as FedAvg except the averaged aggregation is replaced by a convex combination of the initialization and the average of the last iterates, as in Equation 3.11. This adds another hyperparameter to b governing the tradeoff between the two.

SCAFFOLD [Karimireddy et al., 2020] SCAFFOLD comes in two variants, both of which compute and aggregate control variates in parallel to the model weights. The decomposition here is:

$\texttt{Loc}_c$ Local SGD starting from a weight initialization with a control variate, which can be merged to form the local training initialization. The hyperparameters in c are the same as in FedAvg.

$\texttt{Agg}_b$ Weighted averaging of both the initialization and the control variates, with the same hyperparameters as Reptile.

FedDyn [Acar et al., 2021] In addition to a FedAvg/FedProx/Reptile-like training routine, this algorithm maintains a regularizer on each device that affects the local training routine. While this statefulness cannot strictly be subsumed in our decomposition, since it does not introduce any additional hyperparameters the remaining hyperparameters can be tuned in the same manner as we do for FedAvg/FedProx/Reptile. In order to choose between using FedDyn or not, one can introduce a binary hyperparameter to c specifying whether or not $\texttt{Loc}_c$ uses that term in the objective it optimizes or not, allowing it also to be tuned via FedEx.

FedPA [Al-Shedivat et al., 2021] This algorithm replaces local SGD in FedAvg by a local Markov-chain Monte Carlo (MCMC) routine starting from the initialization given by aggregating the previous round's MCMC routines. The decomposition is then just a replacement of local

SGD and its associated hyperparameters by local MCMC and its hyperparameters, with the aggregation routine remaining the same.

Ditto [Li et al., 2021b] Although it depends on what solver is used for the local solver and aggregation routines, in the simplest formulation, the local optimization of personalized models involves an additional regularization hyperparameter. While the updating rule of Ditto is different from that of FedProx, the hyperparameters can be decomposed and tuned in a similar manner.

MAML [Finn et al., 2017] A well-known meta-learning algorithm, MAML takes one or more full-batch gradient descent (GD) steps locally and updates the global model using a second-order gradient using validation data. The decomposition here is:

$\texttt{Loc}_c$ Local SGD starting from a weight initialization. The hyperparameters in c are the same as in FedAvg. The algorithm also returns second-order information required to compute the meta-gradient.

$\texttt{Agg}_b$ Meta-gradient computation, summation, and updating using a standard descent method like Adam [Kingma and Ba, 2015]. The hyperparameters in b are the hyperparameters of the latter.

3.C FedEx details

3.C.1 Stochastic gradient used by FedEx

Below is a simple calculation showing that the stochastic gradient used to update the categorical distribution of FedEx is an unbiased estimate of the true gradient w.r.t. its parameters.

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}[j]} \mathbb{E}_{c_{ij}|\boldsymbol{\theta}} L_{V_{ti}}(\mathbf{w}_i) &= \nabla_{\boldsymbol{\theta}[j]} \mathbb{E}_{c_{ij}|\boldsymbol{\theta}} (L_{V_{ti}}(\mathbf{w}_i) - \lambda) \\
&= \mathbb{E}_{c_{ij}|\boldsymbol{\theta}} \left((L_{V_{ti}}(\mathbf{w}_i) - \lambda) \nabla_{\boldsymbol{\theta}[j]} \log \mathbb{P}_{\boldsymbol{\theta}}(c_{ij}) \right) \\
&= \mathbb{E}_{c_{ij}|\boldsymbol{\theta}} \left((L_{V_{ti}}(\hat{w}_k) - \lambda) \nabla_{\boldsymbol{\theta}[j]} \log \prod_{i=1}^{n} \mathbb{P}_{\boldsymbol{\theta}}(c_{ij} = c_j) \right) \\
&= \mathbb{E}_{c_{ij}|\boldsymbol{\theta}} \left((L_{V_{ti}}(\mathbf{w}_i) - \lambda) \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}[j]} \log \mathbb{P}_{\boldsymbol{\theta}}(c_{ij} = c_j) \right) \\
&= \mathbb{E}_{c_{ij}|\boldsymbol{\theta}} \left(\frac{(L_{V_{ti}}(\mathbf{w}_i) - \lambda) 1_{c_{ij}=c_j}}{\boldsymbol{\theta}[j]} \right)
\end{aligned} \tag{3.13}$$

Note that this use of the reparameterization trick has some similarity with a recent RL approach to tune the local step-size and number of epochs [Mostafa and Wang, 2019]; however, FedEx can be rigorously formulated as an optimization over the personalization objective, has provable guarantees in a simple setting, uses a different configuration distribution that leads to our exponentiated update, and crucially for practical deployment does not depend on obtaining aggregate reward signal on each round.

Algorithm 9: FedEx wrapped with SHA.

Input: distribution $\mathcal{D}$ over hyperparameters $\mathcal{A}$, elimination rate $\eta \in \mathbb{N}$, elimination rounds $\tau_0 = 0, \tau_1, \ldots, \tau_R$

sample set of η^R hyperparameters $H \sim \mathcal{D}^{[\eta^R]}$

initialize a model $\mathbf{w}_a \in \mathbb{R}^d$ for each $a \in H$

for *elimination round* $r \in [R]$ **do**

  for *setting* $a = (b, c) \in H$ **do**

    $s_a, \mathbf{w}_a, \boldsymbol{\theta}_a \leftarrow \texttt{FedEx}(\mathbf{w}_a, b, c, \boldsymbol{\theta}_a, \tau_{r+1} - \tau_r)$

  $H \leftarrow \{a \in H : s_a \leqslant \frac{1}{\eta}\text{-quantile}(\{s_a : a \in H\})\}$

Output: remaining $a \in H$ and associated model $\mathbf{w}_a$

$\texttt{FedEx}(\mathbf{w}, b, \{c_1, \ldots, c_k\}, \boldsymbol{\theta}, \tau \geqslant 1)$:

---

initialize $\boldsymbol{\theta}_1 \leftarrow \boldsymbol{\theta}$

initialize shared weights $\mathbf{w}_1 \leftarrow \mathbf{w}$

for *comm. round* $t = 1, \ldots, \tau$ **do**

  for *client* $i = 1, \ldots, B$ **do**

    send $\mathbf{w}_t, \boldsymbol{\theta}_t$ to client

    sample $c_{ti} \sim \mathcal{D}_{\boldsymbol{\theta}_t}$

    $\mathbf{w}_{ti} \leftarrow \texttt{Loc}_{c_{ti}}(T_{ti}, \mathbf{w}_t)$

    send $\mathbf{w}_{ti}, c_{ti}, L_{V_{ti}}(\mathbf{w}_{ti})$ to server

  $\mathbf{w}_{t+1} \leftarrow \texttt{Agg}_b(\mathbf{w}, \{\mathbf{w}_{ti}\}_{i=1}^B)$

  set step-size η_t and baseline λ_t

  $\tilde{\nabla}_j \leftarrow \frac{\sum_{i=1}^B |V_{ti}|(L_{V_{ti}}(\mathbf{w}_{ti}) - \lambda_t)\mathbf{1}_{c_{ti}=c_j}}{\boldsymbol{\theta}_{t[j]} \sum_{i=1}^B |V_{ti}|} \;\forall\; j$

  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t \odot \exp(-\eta_t \tilde{\nabla})$

  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_{t+1} / \|\boldsymbol{\theta}_{t+1}\|_1$

  $s \leftarrow \sum_{i=1}^B |V_{ti}| L_{V_{ti}} / \sum_{i=1}^B |V_{ti}|$

Return s, model $\mathbf{w}$, hyperparameter distribution $\boldsymbol{\theta}$

3.C.2 Hyperparameters of FedEx

We tune the computation of the baseline λ_t, which we set to

$$\lambda_t = \frac{1}{\sum_{s<t} \gamma^{t-s}} \sum_{s<t} \frac{\gamma^{t-s}}{\sum_{i=1}^B |V_{ti}|} \sum_{i=1}^B L_{V_{ti}}(\mathbf{w}_i) \tag{3.14}$$

for discount factor $\gamma \in [0, 1]$. As discussed in Section 3.4, the local perturbation factor is set to $\varepsilon = 0.1$. 27 configurations are used in each arm for SHA and RS. The number of configuration used per arm of FedEx (i.e. the dimensionality of $\boldsymbol{\theta}$) is the same (27).

Table 3.2: Final test error obtained when tuning using a standard hyperparameter tuning algorithm (SHA or RS) alone, or when using it for server (aggregation) hyperparameters while FedEx tunes client (on-device training) hyperparameters. The target model is the one used to compute on-device validation error by the wrapper method, as well as the one used to compute test error after tuning. The confidence intervals displayed are 90% Student-t confidence intervals for the mean estimates from Table 3.1, with 5 independent trials for Shakespeare, 10 for FEMNIST, 10 for RS on CIFAR, and 6 for SHA on CIFAR.

Wrapper method	Target model	Tuning method	Shakespeare i.i.d.	Shakespeare non-i.i.d.	FEMNIST i.i.d.	FEMNIST non-i.i.d.	CIFAR-10 i.i.d.
Random Search (RS)	global	RS (server & client)	60.32 ± 9.56	64.36 ± 13.53	22.81 ± 2.64	22.98 ± 1.98	30.46 ± 5.47
		+ FedEx (client)	53.94 ± 8.70	57.70 ± 16.75	20.96 ± 2.77	22.30 ± 2.12	34.83 ± 8.54
	person-alized	RS (server & client)	61.10 ± 8.89	61.71 ± 8.66	17.45 ± 1.63	17.77 ± 1.52	34.89 ± 6.12
		+ FedEx (client)	54.90 ± 9.50	56.48 ± 12.97	16.31 ± 2.19	15.93 ± 1.77	39.13 ± 8.77
Successive Halving (SHA)	global	SHA (server & client)	47.38 ± 3.24	46.79 ± 3.35	18.64 ± 0.97	20.30 ± 0.96	21.62 ± 1.45
		+ FedEx (client)	$\mathbf{44.52} \pm 1.60$	$\mathbf{45.24} \pm 3.16$	19.22 ± 1.19	19.43 ± 0.84	$\mathbf{20.82} \pm 0.79$
	person-alized	SHA (server & client)	46.77 ± 3.44	48.04 ± 3.54	$\mathbf{14.79} \pm 0.90$	14.78 ± 0.75	24.81 ± 3.55
		+ FedEx (client)	46.08 ± 2.45	45.89 ± 3.58	14.97 ± 0.76	$\mathbf{14.76} \pm 0.99$	21.77 ± 1.64

3.D Experimental details

Code for FedEx is available here: `https://github.com/mkhodak/fedex`. Shakespeare and FEMNIST data can be found here: `https://github.com/TalwalkarLab/leaf`.

3.D.1 Settings of the baseline/wrapper algorithm

We use the same settings of Algorithm 7 for both tuning FedAvg and wrapping FedEx. Given an elimination rate η, number of elimination rounds R, budget B, and maximum rounds per arm M, we assign $T_1, \dots, T_R$ s.t. $T_i - T_{i-1} = (T - M) / \left(\frac{\eta^{n+1}-1}{\eta-1} - n - 1\right)$, with $T_0 = 0$.

3.D.2 Hyperparameters of FedAvg/FedProx/Reptile

Server hyperparameters (learning rate $\alpha_t = \gamma^t$):

- $\log_{10}$ lr: $\mathrm{Unif}[-1, 1]$
- momentum: $\mathrm{Unif}[0, 0.9]$
- $\log_{10}(1-\gamma)$: $\mathrm{Unif}[-4, -2]$

Local hyperparameters (n.b. we only use one epoch for Shakespeare to conserve computation):

- $\log_{10}$(lr): $\mathrm{Unif}[-4, 0]$
- momentum: $\mathrm{Unif}[0.0, 1.0]$
- $\log_{10}$(weight decay): $\mathrm{Unif}[-5, -1]$
- epoch: $\mathrm{Unif}\{1, 2, 3, 4, 5\}$
- $\log_2$(batch): $\mathrm{Unif}\{3, 4, 5, 6, 7\}$
- dropout: $\mathrm{Unif}[0, 0.5]$

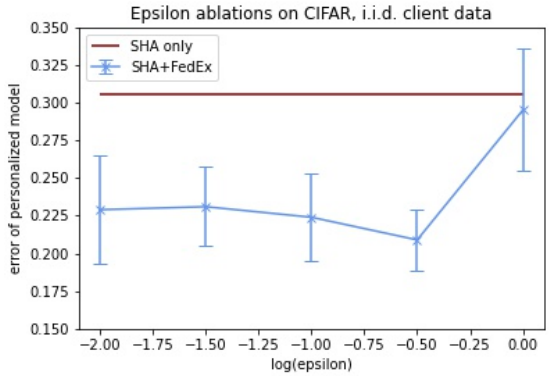

Figure 3.5: Comparison of different ε settings for the local perturbation component of FedEx from Section 3.4.

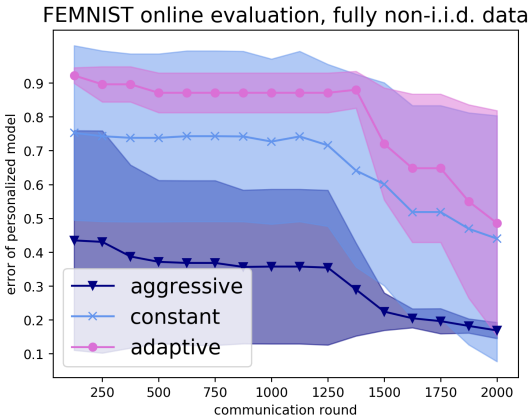

Figure 3.6: Comparison of step-size schedules for η_t in FedEx. In practice we chose the 'aggressive' schedule, which exhibits faster convergence to favorable configurations.

3.E Ablation studies

We now discuss two design choices of FedEx and how they affect performance of the algorithm. First, the choice of the local perturbation $\varepsilon = 0.1$ discussed in Section 3.4; we choose this setting due to its consistent performance across several settings. In Figure 3.5 we plot the performance of FedEx on CIFAR-10 between $\varepsilon = 0.0$ (no FedEx, i.e. SHA only) and $\varepsilon = 1.0$ (full FedEx, i.e. client configurations are chosen independently) and show that while the use of a nonzero ε is important, performance at fairly low values of ε is roughly similar.

We further investigated the setting of the step-size η_t for the exponentiated gradient update in FedEx. We examine three different approaches: a constant rate of $\eta_t = \sqrt{2 \log k}$, an 'adaptive' schedule of $\eta_t = \sqrt{2 \log k} / \sqrt{\sum_{s \leqslant t} \|\tilde{\nabla}_s\|_\infty^2}$, and an 'aggressive' schedule of $\eta_t = \sqrt{2 \log k} / \|\tilde{\nabla}_t\|_\infty$. Here $\tilde{\nabla}_t$ is the stochastic gradient w.r.t. θ computed in Algorithm 8 at step t and the form of the step-size is derived from standard settings for exponentiated gradient in online learning [Shalev-Shwartz, 2011]. We found that the 'aggressive' schedule works best in practice, as shown in Figure 3.6. A key issue with using the 'constant' and 'adaptive' approaches is that they continue to assign high probability to several configurations late in the tuning process; this slows down training of the shared weights. One could consider a tradeoff between allowing FedEx to run longer than while keeping the total budget constant, but for simplicity we chose the more effective 'aggressive' schedule.

Part II

Algorithms with predictions

Chapter 4

Overview

The second part of this thesis moves away from learning to parameterize learning algorithms to learning parameters of regular algorithms. We focus on algorithms with predictions [Mitzenmacher and Vassilvitskii, 2021], a subfield of beyond-worst-case analysis of algorithms that aims to design methods that make use of machine-learned predictions in order to reduce runtime, error, or some other performance cost. Mathematically, given some prediction $\mathbf{x}$, algorithms in this field are designed such that the cost $C_t(\mathbf{x})$ of an instance t while using the prediction is upper-bounded by some measure $U_t(\mathbf{x})$ of the quality of the prediction on that instance. The canonical example here is that the cost of binary search on a sorted array of size n can be improved from $\mathcal{O}(\log n)$ to at most $U_t(\mathbf{x}) = 2 \log \eta_t(\mathbf{x})$, where $\eta_t(\mathbf{x})$ is the distance between the true location of a query t in the array and the location predicted by the predictor $\mathbf{x}$ [Mitzenmacher and Vassilvitskii, 2021]. In recent years, algorithms whose cost depends on the quality of possibly imperfect predictions have been developed for numerous important problems, including caching [Rohatgi, 2020, Jiang et al., 2020, Lykouris and Vassilvitskii, 2021], scheduling [Lattanzi et al., 2020, Scully et al., 2022], ski-rental [Kumar et al., 2018, Anand et al., 2020, Diakonikolas et al., 2021], bipartite matching [Dinitz et al., 2021], page migration [Indyk et al., 2022], and many more [Bamas et al., 2020, Du et al., 2021a, Mitzenmacher and Vassilvitskii, 2021].

While there has been a significant effort to develop algorithms that can take advantage of learned predictions, there has been less focus on actually *learning* to predict. For example, of the works listed above only two focusing on ski-rental [Anand et al., 2020, Diakonikolas et al., 2021] and one other [Dinitz et al., 2021] show sample complexity guarantees, and none consider the important *online learning* setting, in which problem instances are not guaranteed to come from a fixed distribution. This is in contrast to the related area of data-driven algorithm design [Gupta and Roughgarden, 2017, Balcan, 2021], which has established techniques such as dispersion for deriving learning-theoretic guarantees, leading to end-to-end results encompassing both learning and computation [Balcan et al., 2018b]. It is also despite the fact that learning even simple predictors is in many cases a non-trivial problem.

Chapter 5 bridges this gap with a framework for obtaining learning-theoretic guarantees for algorithms with predictions. In addition to better sample complexity bounds, we show how to learn the parameters of interest in *online* via bounds on the overall regret. Moreover, we show the first *instance-dependent* learning guarantees for predictions, showing that one can learn not just static predictions but prediction *policies* that customize their prediction to the instance at-hand;

this hews more closely to what is done in practice. All of this is accomplished using a two-step approach inspired by ARUBA, with regret-upper-bounds replaced by *runtime*-upper-bounds.

To show the above results we study existing algorithms with predictions, which as a subfield has focused on graph algorithms, data structures, and online algorithms. However, incorporating external information via predictions has significant potential across all areas of algorithm design, as we demonstrate in Chapters 6 and7, where we extend the field to two completely new directions: differentially private statistics and scientific computing. Theoretically, these directions yield interesting new phenomena at the intersection of learning and algorithms, including (1) the impossibility of simultaneous robustness and simple learnability for private quantile release and (2) *instance-optimal* prediction for solving certain sequences of linear systems. We also demonstrate the utility of incorporating learned predictions through extensive experiments.

4.1 Literature

Algorithms with predictions is an important approach to the beyond-worst-case analysis of algorithms [Roughgarden, 2020], studies of computation that take advantage of the fact that many real-world instances are not worst-case. Other approaches include smoothed analysis [Spielman and Teng, 2004] and data-driven algorithm design [Balcan, 2021]. Inspired by strong empirical success in applications such as learned index structures [Kraska et al., 2018], the area has seen a great deal of theoretical study, with a particular focus on algorithms whose guarantees depend on the quality of a given predictor (c.f. the survey of Mitzenmacher and Vassilvitskii [2021] and the references therein). The actual learning of this predictor has been studied to a lesser extent [Anand et al., 2020, Diakonikolas et al., 2021, Dinitz et al., 2021, Chen et al., 2022] and very rarely in the online learning setting; in Chapter 5 we present the first general framework for efficiently learning useful predictors.

Data-driven algorithm design is a closely related area that has seen much more learning-theoretic effort [Gupta and Roughgarden, 2017, Balcan et al., 2018b, Balcan, 2021]. At a high-level, it often studies the tuning of algorithmic parameters such as the step-size of gradient descent [Gupta and Roughgarden, 2017] or settings of branch and bound for solving integer programs [Balcan et al., 2018a], whereas the predictors in algorithms with predictions often either try to guess the full sequence in an online algorithm [Indyk et al., 2022] or the actual outcome of the computation such as the dual in a primal-dual method [Dinitz et al., 2021]. The distinction can be viewed as terminological, since a prediction of the outcome can be viewed as a parameter of the algorithm, but it does mean that in the settings we study we have full information about the loss function since it is typically some discrepancy between the full sequence or computational outcome and the prediction. In contrast, in data-driven algorithm design getting the cost of each parameter setting often requires additional computation, and so we are often in a bandit or semi-bandit setting [Balcan et al., 2020a]. A more salient difference is that data-driven algorithm design guarantees compete with the parameter that minimizes average cost but do not always quantify the improvement attainable via learning; in algorithms with predictions we do generally quantify this improvement with an upper bound on the cost that depends on the prediction quality, but we usually only compete with the parameter that is optimal for prediction quality, which is not always optimal for the cost. In Chapter 5 we do adapt ideas from data-driven

algorithm design, specifically dispersions, to show guarantees for algorithms with predictions; later in Chapter 7 we also show data-driven algorithm design-type guarantees for the problem of tuning a linear solver, allowing us to compare results from the two subfields more directly.

4.2 Contributions

The algorithms with predictions results in this thesis stem from centering *learning* as a crucial aspect of the field. This approach allows us to both design algorithms for learning to predict by extending the ARUBA framework, which we do in Chapter 5, and guides the development of new algorithms with predictions in the fields of differentially private statistics (Chapter 6) and scientific computing (Chapter 7). In particular, learning influences our exploration of *what* predictions to use and *how* they might be useful; for example, for private quantiles we study what distributional families to use as priors for releasing statistics and for setting preconditioners in PDE solvers we convert the entire setup into an online learning problem.

4.2.1 Learning predictions, provably

While much work in algorithms with predictions has focused on using predictions to improve competitive ratios, running times, or other performance measures, less effort has been devoted to the question of how to obtain the predictions themselves, especially in the critical online setting. In Chapter 5 we introduce a general design approach for algorithms that learn predictors: (1) identify a functional dependence of the performance measure on the prediction quality and (2) apply techniques from online learning to learn predictors, tune robustness-consistency tradeoffs, and bound the sample complexity. This approach is effectively an extension of the ARUBA framework from Chapter 2—which focused on learning algorithms with learning-theoretic performance measures—to general algorithms and objectives.

We demonstrate the effectiveness of this approach by applying it to several graph algorithms, such as bipartite matching, and online algorithms such as scheduling, ski-rental, and page migration, and job scheduling. For the graph algorithms our optimization-based approach allows us to obtain dramatic improvements in the sample complexity, reducing it by $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$ factors, where n is the number of nodes in the graph. In the online algorithms our guarantees are for the most part the first learning-theoretic guarantees available. Importantly, our approach allows us to reason for the first time about provable *instance-dependent* prediction, in which rather than learning a fixed prediction for every instance we learn a model from instance features to customized predictions. This reflects much better what is done in practice and will be crucial to many of the extensions we show in the applications in Chapters 6 and 7.

4.2.2 Extending algorithms with predictions

While predictions have classically been used to augment online algorithms, graph algorithms, and data structures, our learning-based understanding points to many other applications areas for this paradigm, two of which we study in detail.

Private statistics. In Chapter 6 we use our learning-based design approach to develop learning-augmented algorithms for releasing differentially private (DP) statistics about sensitive data. Here a common way of getting improved performance is to use external information such as other sensitive data, public data, or human priors. We propose to use the algorithms with predictions framework as a powerful way of designing and analyzing privacy-preserving methods that use such external information to improve *utility*. For three important tasks—(multiple) quantile release, covariance estimation, and data release—we construct prediction-dependent DP methods whose utility scales with natural measures of prediction quality. Our analysis enjoys several advantages, including minimal assumptions about the data, natural ways of adding robustness, and the provision of useful surrogate losses for two novel "meta" algorithms that learn predictions from other (potentially sensitive) data. We conclude with experiments in a diverse set of multi-dataset quantile release settings that show how a learning-augmented approach to incorporating external information can lead to large error reductions while preserving privacy.

Scientific computing. Chapter 7 concludes our contributions to data-driven algorithms with a study of linear system solvers, fundamental primitives in scientific computing systems for which numerous solvers and preconditioners have been developed. They come with parameters whose optimal values depend on the system being solved and are often impossible or too expensive to identify; thus in practice sub-optimal heuristics are used. We consider the common setting in which many related linear systems need to be solved, e.g. during a single numerical simulation. In this scenario, can we sequentially choose parameters that attain a near-optimal overall number of iterations, without extra matrix computations? We answer in the affirmative for Successive Over-Relaxation (SOR), a standard solver whose parameter ω has a strong impact on its runtime. For this method, we prove that a bandit online learning algorithm—using only the number of iterations as feedback—can select parameters for a sequence of instances such that the overall cost approaches that of the best fixed ω as the sequence length increases. Furthermore, when given additional structural information, we show that a *contextual* bandit method asymptotically achieves the performance of the *instance-optimal* policy, which selects the best ω for each instance. Our work provides the first learning-theoretic treatment of high-precision linear system solvers and the first end-to-end guarantees for data-driven scientific computing, demonstrating theoretically the potential to speed up numerical methods using well-understood learning algorithms. In addition to these technical contributions, this is also the first instance where the separate fields of learning-augmented algorithms and data-driven algorithm design are used to solve the same problem, allowing us to compare these two paradigms more directly.

4.2.3 Contributions of independent interest

Our study of private estimation schemes parameterized by learned predictions highlights how useful doing such learning-augmented analysis can be to the original field of study, in this case yielding several results of broader interest to DP. First, for the problem of quantile estimation we design the first private algorithm that does not require knowledge of an interval containing the true quantile to be run. While our scheme does take a guess of the interval as an input, it has an error guarantee of $\mathcal{O}(\frac{1}{\varepsilon}\log R)$, where R is the distance between the guess and the true quantile

and $\varepsilon > 0$ is the privacy parameter.; in contrast, past approaches have vacuous guarantee if the interval is misspecified. Our key insight is to use a heavy-tailed (e.g. Cauchy) base distribution in the exponential mechanism instead of the usual uniform prior; this allows sufficient weight to be assigned to the interval containing the prior even in the case of misspecification.

A second contribution is to DP covariance estimation, where our analysis results in a strict improvement to the state-of-the-art trace-dependent guarantees of Dong et al. [2022, Theorem 1] *without changing their algorithm*. In particular, whereas they show error bounds that grow with the trace norm $\|\mathbf{C}\|_{\mathrm{Tr}}$ of the true covariance matrix $\mathbf{C} \in \mathbb{R}^{d \times d}$, we replace this term by $\min_{c \in \mathbb{R}} \|\mathbf{C} - c\mathbf{I}\|_{\mathrm{Tr}}$. The new guarantee can be much stronger because covariance matrices of commonly occurring distributions such as isotropic Gaussians are often close to scalar multiples of the identity, in which case $\min_{c \in \mathbb{R}} \|\mathbf{C} - c\mathbf{I}\|_{\mathrm{Tr}}$ vanishes while $\|\mathbf{C}\|_{\mathrm{Tr}} = \mathcal{O}(d)$. Our result follows from the observation that a prediction-dependent extension the algorithm of Dong et al. [2022] is invariant to perturbing the prediction by a scalar multiple of the identity.

Lastly, we introduce a non-Euclidean extension of DP-FTRL [Kairouz et al., 2021a], a private online convex optimization method. This is the first DP online learning scheme applicable to general convex losses that is customizable to different geometries, e.g. to obtain regret bounds with better dimension-dependence (compared to Euclidean FTRL) when the optimization domain is the simplex or the trace ball.

4.3 Discussion

Our first major contribution to algorithms with predictions is a theoretical framework for analyzing how to learn the predictions themselves, a direction that has seen expanded interest in recent years. Apart from our own work presented in DP statistics and scientific computing, the optimization-based approach we introduced in Chapter 4 was also used to show prediction-learning guarantees in a sequence of work on discrete convex optimization tasks [Sakaue and Oki, 2022, 2023, Oki and Sakaue, 2023]. There have also been several efforts to go beyond the static predictions analyzed in most learning-theoretic results for data-driven algorithms, although the results have generally differed significantly from our approach of learning instance-dependent prediction models on top of instance features. For example, Srinivas and Blum [2024] showed learning guarantees for warm-start-type algorithms—e.g. those for bipartite matching—that are competitive with *multiple* predictions simultaneously, while Eliáš et al. [2024] studied learning to make predictions *while running* an online algorithm.

Beyond learning, interest in learning-augmented algorithms continues to grow, with new directions including the incorporation of uncertainty about the predictions [Christianson et al., 2024, Sun et al., 2024] and proving bounds if predictions are correct with probability ε and arbitrarily poor otherwise [Gupta et al., 2022, Cohen-Addad et al., 2024]. Our own work in Chapters 6 and 7 points to great opportunities to develop learning-augmented methods for statistical and numerical tasks; for example, can predictions be incorporated into probabilistic algorithms such as sampling or numerical subroutines beyond linear solvers? As we demonstrate with our analysis of learning-augmented scientific computing, these directions also have significant potential to connect with other subfields at the intersection of ML and algorithms, including amortized optimization [Amos, 2023] and data-driven algorithm design [Bartlett et al., 2022].

4.A Background

We first give some necessary background on the field of algorithms with predictions, also known as **learning-augmented algorithms**; for a more in-depth introduction see the survey by Mitzenmacher and Vassilvitskii [2021]. The basic requirement for a learning-augmented algorithm is that the cost $C_t(\mathbf{x})$ of running it on an instance t with prediction $\mathbf{x}$ should be upper bounded—usually up to constant or logarithmic factors—by a metric $U_t(\mathbf{x})$ of the quality of the prediction on the instance. We denote this by $C_t \lesssim U_t$.

A *good* guarantee for a learning-augmented algorithm will have several important properties that formally separate its performance from naive upper bounds $U_t \gtrsim C_t$. The first, *consistency*, requires it to be a reasonable indicator of strong performance in the limit of perfect prediction:

Definition 4.A.1. A guarantee $C_t \lesssim U_t$ is c_t**-consistent** if $C_t(\mathbf{x}) \leqslant c_t$ whenever $U_t(\mathbf{x}) = 0$.

Here c_t is a prediction-independent quantity that should depend weakly or not at all on problem difficulty.

Consistency is often presented via a tradeoff with *robustness* [Lykouris and Vassilvitskii, 2021], which bounds how poorly the method can do when the prediction is bad, in a manner similar to a standard worst-case bound:

Definition 4.A.2. A guarantee $C_t \lesssim U_t$ is r_t**-robust** if it implies $C_t(\mathbf{x}) \leqslant r_t$ for all predictions $\mathbf{w}$.

Unlike consistency, robustness usually depends strongly on the difficulty of the instance $\mathbf{x}$, with the goal being to not do much worse than a prediction-free approach. Note that the latter is trivially robust but not (meaningfully) consistent, since it ignores the prediction; this makes clear the need for considering the two properties via a tradeoff between them.

Robustness-consistency tradeoffs are often presented as *parameterized* upper bounds, with a parameter $\lambda \in [0, 1]$ that specifies how tolerant the user is to performance worse than the worst-case guarantee. A typical guarantee then has an upper bound of the form

$$U_t(\mathbf{x}, \lambda) = \min\{f(\lambda)u_t(\mathbf{x}), g_t(\lambda)\} \geqslant C_t(\mathbf{x}) \tag{4.1}$$

with f monotonically increasing (often $f(0) = 1$), g_t monotonically decreasing, and $u_t(\mathbf{x})$ measures the quality of the prediction $\mathbf{x}$ on instance t. A very common structure is $f(\lambda) = 1/(1-\lambda)$ and $g_t(\lambda) \propto 1/\lambda$. Thus specifying a small λ results in good performance under good predictions but sacrifices robustness, while setting λ closer to 1 never does much worse than a prediction-free guarantee but takes less advantage of good predictions.

Chapter 5

Learning predictions

As discussed in Chapter 4, past work in algorithms with predictions has focused on designing methods with good robustness-consistency tradeoffs, i.e. that can take advantage of a good prediction without doing much worse than a default approach if the prediction is poor. However, this focus ignores the question of where the predictions come from, which is usually *learning*: whether or not predictions can be learned has been much less studied, and what guarantees do exist have exclusively shown (a) sample complexity guarantees for (b) *static* predictions [Anand et al., 2020, Diakonikolas et al., 2021, Dinitz et al., 2021, Chen et al., 2022]. This chapter of the thesis introduces a way to systematically obtain learning-theoretic guarantees for algorithms with predictions, with a particular emphasis on avoiding the two restrictions above by (a) proving regret bounds for sequences of adversarial instances (that still imply sample complexity guarantees via online-to-batch conversion) and (b) showing guarantees that can be extended to simple *instance-dependent* predictions such as linear maps from instance features.

Our approach extends the ARUBA framework from Chapter 2—which we introduced for the purposes of designing and analyzing meta-learning—to algorithms with predictions, in particular to learning predictions or predictors using past instances. This is a natural generalization of the framework, as in meta-learning can be viewed as the problem of learning to predict a good initialization for learning algorithms; the main difference now is that the objective is no longer a learning-theoretic quantity such as regret but can rather be any performance measure such as time complexity or competitive ratio.

The generalization manifests as a two-step framework for applying it to any algorithms with prediction problem:

1. For a given algorithm, derive a *meaningful* and convenient-to-optimize upper bound $U_t(\mathbf{x})$ on the cost $C_t(\mathbf{x})$ that depends on both the prediction $\mathbf{x}$ and information specific to instance t returned once the algorithm terminates, e.g. the optimum in combinatorial optimization.
2. Apply online learning to these upper bounds obtain both regret guarantees against adversarial sequences and sample complexity bounds for i.i.d. instances.

The challenging part of this framework is usually the first step, as it is often trivially easy to bound C_t but difficult to do so with a bound that is both meaningful—i.e. non-vacuous and indicative of the underlying performance—and easy-to-optimize. Whether a bound is meaningful

[0]The work presented in this chapter first appeared in Khodak et al. [2022].

can be a subjective question, but since in this chapter we focus on showing learning guarantees for *existing* algorithms with predictions, we can formalize whether or not a bound is meaningful using existing guarantees. In particular, we find that for many tasks—e.g. bipartite matching [Dinitz et al., 2021] and online page migration [Indyk et al., 2022]—past work has shown prediction-dependent upper bounds $U'_t \gtrsim C_t$ that are challenging to optimize, e.g. discontinuous or nonconvex. Assuming the bound U'_t from past work is meaningful, we will say that the new, easy-to-optimize bounds $U_t \gtrsim C_t$ that we derive are also meaningful if they are themselves upper bounded (up to constant or at worst logarithmic factors) by the existing bound U'_t.

Apart from this, our approach is designed to be simple-to-execute, leaving much of the difficulty to what the field is already good at: designing algorithms and proving prediction-quality-dependent upper bounds U_t on their costs C_t Once the latter is accomplished, our framework leverages problem-specific structure to design a customized learning algorithm for each problem, leading to strong regret and sample complexity guarantees. In particular, in multiple settings we improve upon existing results in either sample complexity or generality, and in all cases we are the first to show regret guarantees in the online setting. This demonstrates the usefulness of and need for such a theoretical framework for studying these problems.

We summarize the diverse set of contributions enabled by our theoretical framework below, as well as in Table 5.1:

1. **Bipartite matching:** Our starting example builds upon the work on **minimum-weight bipartite matching** using the Hungarian algorithm by Dinitz et al. [2021]. We show how our framework leads directly to both the first regret guarantees in the online setting and new sample complexity bounds that improve over the previous approach by a factor linear in the number of nodes. In Appendices 5.B and 5.D we show similar strong improvements for b-matching and other graph algorithms.
2. **Page migration:** We next study a more challenging application, **online page migration**, and show how we can adapt the algorithmic guarantee of Indyk et al. [2022] into a learnable upper bound for which we can again provide both adversarial and statistical guarantees.
3. **Learning linear maps with instance-feature inputs:** Rather than assume the existence of a strong fixed prediction, it is often more natural to assume each instance comes with features that can be input into a predictor such as a linear map. Our approach yields the first guarantees for learning linear predictors for algorithms with predictions, which we obtain for the two problem settings above and also for **online job scheduling** using makespan minimization [Lattanzi et al., 2020].
4. **Tuning robustness-consistency tradeoffs:** Many bounds for *online* algorithms with predictions incorporate parameterized tradeoffs between trusting the prediction or falling back on a worst-case approximation. This suggests the usefulness of tuning the tradeoff parameter, which we instantiate on a simple job scheduling problem with a fixed predictor. Then we turn to the more challenging problem of simultaneously tuning the tradeoff and learning predictions, which we achieve on two variants of the **ski-rental** problem. For the discrete case we give the only learning-theoretic guarantee, while for the continuous case our bound uses a dispersion assumption [Balcan et al., 2018b] that, in the i.i.d. setting, is a strictly weaker assumption than the log-concave requirement of Diakonikolas et al. [2021].

Table 5.1: Settings we apply our framework to, our new learning algorithms, and their regret.

Problem	Algorithm with prediction	Feedback	Upper bound (losses)	Learning algo.	Regret
Min. weight bipartite matching (5.2)*,†	Hungarian method initialized by dual $\hat{\mathbf{x}} \in \mathbb{R}^n$	Opt. dual $\mathbf{x}^*(\mathbf{c})$	$\mathcal{O}\left(\lVert\hat{\mathbf{x}} - \mathbf{x}^*(\mathbf{c})\rVert_1\right)$	Proj. online gradient	$\mathcal{O}\left(n\sqrt{T}\right)$
Online page migration (5.3)*	Lazy offline optimal for predictions $\{\hat{s}_{[j]} \sim \mathbf{p}_{[j]}\}_{j=1}^n$	Requests $\{s_{[j]}\}_{j=1}^n$	$\bar{\mathcal{O}}\left(\max_{i\in[n]} \mathbb{E}_{\mathbf{p}} \sum_{j=i}^{i+\gamma D} 1_{\hat{s}_{[j]} \neq s_{[j]}}\right)$	Exponentiated gradient $\times n$	$\mathcal{O}\left(n\sqrt{T}\right)$
Online job scheduling (5.4)*	Corrected offline optimal for predicted logits $\hat{\mathbf{x}} \in \mathbb{R}^m$	Opt. weights $\mathbf{w} \in \triangle^m$	$\mathcal{O}\left(\lVert\hat{\mathbf{x}} - \log \mathbf{w}\rVert_\infty\right)$	Euclidean KT-OCO	$\mathcal{O}\left(\sqrt{mT\log(mT)}\right)$
Non-clairvoyant job scheduling (5.5)‡	Preferential round-robin with tradeoff parameter λ	Prediction quality η	$\min\left\{\frac{1+2\eta/n}{1-\lambda}, \frac{2}{\lambda}\right\}$	Exponential forecaster	$\mathcal{O}\left(\sqrt{T\log T}\right)$
Ski-rental w. integer days $n \in [N]$ (5.5)	Buy if price $b \leqslant x$, λ trade-off with worst-case approx.	Number of ski-days n	$\frac{\min\{\lambda(b1_{x>b}+n1_{x\leqslant b}),b,n\}}{1-(1+1/b)^{-b\lambda}}$	Exponentiated gradient	$\mathcal{O}\left(N\sqrt{T\log(NT)}\right)$
Ski-rental with β-dispersed n (5.5)	Buy after x days, λ trade-off with worst-case approx.	Number of ski-days n	$\min\left\{\frac{e\min\{n,b\}}{(e-1)\lambda}, \frac{n1_{n\leqslant x}+(b+x)1_{n>x}}{1-\lambda}\right\}$	Exponential forecaster	$\mathcal{O}\left(\sqrt{T\log(NT)} + N^2T^{1-\beta}\right)$

* For these we also show guarantees in the statistical (i.i.d.) setting and for learning linear predictors that take instance features as inputs.
† We extend these results to minimum-weight $\mathbf{b}$-matching and other graph algorithms with predictions in Appendices 5.C.1 and 5.D.
‡ We also provide new guarantees for the problem of learning job permutations in the non-clairvoyant setting in Appendix 5.E.

5.1 Related work

While there has been a great deal of theoretical study focusing on algorithms whose guarantees depend on the quality of a given predictor (c.f. the Mitzenmacher and Vassilvitskii [2021] survey), the actual learning of these predictions has been less frequently studied [Anand et al., 2020, Diakonikolas et al., 2021, Dinitz et al., 2021], especially in the online setting; we aim to change this with our framework. Some approaches improve online learning itself using predictions [Rakhlin and Sridharan, 2013, Jadbabaie et al., 2015, Dekel et al., 2017], but they also assume known predictors or only learn over a small set of policies, and their goal is minimizing regret not computation. In-general, we focus on showing how algorithms with predictions can make use of online learning rather than on new methods for the latter. Several works [Balcan and Blum, 2007, Rakhlin and Sridharan, 2017, Anand et al., 2021] use learning *while* advising an algorithm, in-effect taking a learning-inspired approach to better make use of a prediction *within* an algorithm, whereas we focus on learning the prediction *outside* of the target algorithm. We present the first general framework for efficiently learning useful predictors.

5.2 Framework overview and bipartite matching application

In this section we outline the theoretical framework for designing algorithms and proving guarantees for learned predictors. As an illustrative example we will use the Hungarian algorithm for bipartite matching, for which Dinitz et al. [2021] demonstrated an instance-dependent upper bound on the running time using a learned dual vector. Along the way, we will show an improvement to their sample complexity bound together with the first online results for this setting.

We first introduce the problem, **min-weight perfect matching** (MWPM), which for a bipartite graph on n nodes and m edges asks for the perfect matching with the least weight according to edge-costs $\mathbf{c} \in \mathbb{Z}^m_{\geqslant 0}$. The Hungarian algorithm is a popular convex optimization-based approach

for which Dinitz et al. [2021] showed a runtime bound of $\tilde{\mathcal{O}}\left(m\sqrt{n}\min\left\{\|\mathbf{x}-\mathbf{x}^*(\mathbf{c})\|_1,\sqrt{n}\right\}\right)$, where $\mathbf{x}\in\mathbb{Z}^n$ initializes the duals in a primal-dual algorithm and $\mathbf{x}^*(\mathbf{c})\in\mathbb{Z}^n$ is dual of the optimal solution; note that the latter is obtained for free after running the method.

5.2.1 Step 1: Upper bound

The first step of our approach is to find a suitable function $U_t(\mathbf{x})$ of the prediction $\mathbf{x}$ that (a) upper bounds the target algorithm's cost $C_t(\mathbf{x})$, (b) can be constructed completely once the algorithm terminates, and (c) can be efficiently optimized. These qualities allow learning the predictor in the second step. The requirements are similar to those of ARUBA for showing results for meta-learning, although there the quantity being upper-bounded was regret, not algorithmic cost.

Many guarantees for algorithms with predictions are already amenable to being optimized, although we will see that they can require some massaging in order to be useful. In many cases the guarantee is a distance metric between the prediction $\mathbf{x}$ and some instance-dependent perfect prediction $\mathbf{x}^*$, which is convex and thus straightforward to learn. This is roughly true of our bipartite matching example, although taking the minimum of a constant and the distance $\|\mathbf{x}-\mathbf{x}^*(\mathbf{c})\|_1$ between the predicted and actual duals makes the problem nonconvex. However, we can further upper bound their result by $\tilde{\mathcal{O}}\left(m\sqrt{n}\|\mathbf{x}-\mathbf{x}^*(\mathbf{c})\|_1\right)$; note that Dinitz et al. [2021] also optimize this quantity, not the tighter upper bound with the minimum. While this might seem to be enough for step one, Dinitz et al. [2021] also require the prediction $\mathbf{x}$ to be integral, which is difficult to combine with standard online procedures. In order to get around this issue, we show that rounding any nonnegative real vector to the closest integer vector incurs only a constant multiplicative loss in terms of the ℓ_1-distance.

Claim 5.2.1. Given any vectors $\mathbf{x}\in\mathbb{Z}^n$ and $\mathbf{y}\in\mathbb{R}^n$, let $\tilde{\mathbf{y}}\in\mathbb{Z}^n$ be the vector whose elements are those of $\mathbf{y}$ rounded to the nearest integer. Then $\|\mathbf{x}-\tilde{\mathbf{y}}\|_1\leqslant 2\|\mathbf{x}-\mathbf{y}\|_1$.

Proof. Let $S\subset[n]$ be the set of indices $i\in[n]$ for which $\mathbf{x}_{[i]}\geqslant\mathbf{y}_{[i]}\iff\tilde{\mathbf{y}}_{[i]}=\lceil\mathbf{y}_{[i]}\rceil$. For $i\in[n]\backslash S$ we have $|\mathbf{x}_{[i]}-\mathbf{y}_{[i]}|\geqslant 1/2\geqslant|\tilde{\mathbf{y}}_{[i]}-\mathbf{y}_{[i]}|$ so it follows by the triangle inequality that

$$\begin{aligned}\|\mathbf{x}-\tilde{\mathbf{y}}\|_1&=\sum_{i\in S}|\mathbf{x}_{[i]}-\tilde{\mathbf{y}}_{[i]}|+\sum_{i\in[n]\backslash S}|\mathbf{x}_{[i]}-\tilde{\mathbf{y}}_{[i]}|\\&\leqslant\sum_{i\in S}|\mathbf{x}_{[i]}-\mathbf{y}_{[i]}|+\sum_{i\in[n]\backslash S}|\mathbf{x}_{[i]}-\mathbf{y}_{[i]}|+|\mathbf{y}_{[i]}-\tilde{\mathbf{y}}_{[i]}|\\&\leqslant\sum_{i\in S}|\mathbf{x}_{[i]}-\mathbf{y}_{[i]}|+2\sum_{i\in[n]\backslash S}|\mathbf{x}_{[i]}-\mathbf{y}_{[i]}|\leqslant 2\|\mathbf{x}-\mathbf{y}\|_1\end{aligned}\tag{5.1}$$

□

Combining this projection with the convex relaxation above and the result of Dinitz et al. [2021] shows that for any predictor $\mathbf{x}\in\mathbb{R}^n$ we have (up to affine transformation) a convex upper bound $U_t(\mathbf{x})=\|\mathbf{x}-\mathbf{x}^*(\mathbf{c}_t)\|_1$ on the runtime of the Hungarian method, as desired. We now move to step two.

5.2.2 Step 2: Online learning

Once one has an upper bound U_t on the cost, the second component of our approach is to apply standard online learning algorithms and results to these upper bounds to obtain guarantees for learning predictions. In online learning, on each of a sequence of rounds $t = 1, \dots, T$ we predict $\mathbf{x}_t \in \mathcal{X}$ and suffer $U_t(\mathbf{x}_t)$ for some adversarially chosen loss function $U_t : X \mapsto \mathbb{R}$ that we then observe; the goal is to use this information to minimize **regret** $\sum_{t=1}^{T} U_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{X}} U_t(\mathbf{x})$, with the usual requirement being that it is **sublinear** in T and thus decreasing on average over time. For bipartite matching, we can just apply regular **projected online (sub)gradient descent** (**OGD**) to losses $U_t(\mathbf{x}) = \|\mathbf{x}-\mathbf{x}^*(\mathbf{c}_t)\|_1$, i.e. the update rule $\mathbf{x}_{t+1} \leftarrow \arg\min_{\mathbf{x}\in\mathcal{X}} \alpha\langle\nabla U_t(\mathbf{x}_t), \mathbf{x}\rangle + \frac{1}{2}\|\mathbf{x}\|_2^2$ for appropriate step-size $\alpha > 0$; as shown in Theorem 5.2.1, this yields sublinear regret via a textbook result. The simplicity here is the point: by relegating as much of the difficulty as we can to obtaining an easy-to-optimize upper bound in step one, we make the actual learning-theoretic component easy. However, as we show in the following sections, it is not always easy to obtain a suitable upper bound, nor is it always obvious what online learning algorithm to apply, e.g. if the upper bounds are nonconvex.

Our use of online learning is motivated by three factors: (1) doing well on non-i.i.d. instances is important in practical applications, e.g. in job scheduling where resource demand changes over time; (2) its extensive suite of algorithms lets us use different methods to tailor the approach to specific settings and obtain better bounds, as we exemplify via our use of exponentiated gradient over the simplex geometry in Section 5.3 and KT-OCO over unbounded Euclidean space in Section 5.4; (3) the existence of classic online-to-batch procedures for converting regret into sample complexity guarantees [Cesa-Bianchi et al., 2004], i.e. bounds on the number of samples needed to obtain an ε-suboptimal predictor w.p. $\geqslant 1 - \delta$. While online-to-batch conversion can be suboptimal [Hazan and Kale, 2014], as we show in Theorems 5.2.1, 5.B.1, and 5.D.1 its application to various graph algorithms with predictions problems *improves* upon existing sample complexity results. See Appendix B.4 for more details on online-to-batch conversion.

We now show how to apply the second online learning step to bipartite matching by improving upon the result of Dinitz et al. [2021] in Theorem 5.2.1; the improvement is the entirely new regret bound against adversarial cost vectors and a $\tilde{\mathcal{O}}(n)$ lower sample complexity. Note how the proof needs only their existing algorithmic contribution, Claim 5.2.1, and some standard tools in online convex optimization.

Theorem 5.2.1. Suppose we have a fixed bipartite graph with $n \geqslant 3$ vertices and $m \geqslant 1$ edges.

1. For any cost vector $\mathbf{c} \in \mathbb{Z}_{\geqslant 0}^m$ and any dual vector $\mathbf{x} \in \mathbb{R}^n$ there exists an algorithm for MWPM that runs in time
$$\tilde{\mathcal{O}}\left(m\sqrt{n}\min\left\{U(\mathbf{x}), \sqrt{n}\right\}\right) \leqslant \tilde{\mathcal{O}}\left(m\sqrt{n}U(\mathbf{x})\right) \tag{5.2}$$
for $U(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^*(\mathbf{c})\|_1$, where $\mathbf{x}^*(\mathbf{c})$ the optimal dual vector associated with $\mathbf{c}$.
2. There exists a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and distribution $\mathcal{D}$ over integer m-vectors with ℓ_∞-norm $\leqslant C$ it takes $\mathcal{O}\left(\left(\frac{Cn}{\varepsilon}\right)^2 \log\frac{1}{\delta}\right)$ samples from $\mathcal{D}$ and returns $\hat{\mathbf{x}}$ s.t. w.p. $\geqslant 1 - \delta$:
$$\mathbb{E}_{\mathbf{c}\sim\mathcal{D}}\|\hat{\mathbf{x}} - \mathbf{x}^*(\mathbf{c})\|_1 \leqslant \min_{\|\mathbf{x}\|_\infty \leqslant C} \mathbb{E}_{\mathbf{c}\sim\mathcal{D}}\|\mathbf{x} - \mathbf{x}^*(\mathbf{c})\|_1 + \varepsilon \tag{5.3}$$

3. Let $\mathbf{c}_1, \ldots, \mathbf{c}_T \in \mathbb{Z}^m_{\geqslant 0}$ be an adversarial sequence of m-vectors with ℓ_∞-norm $\leqslant C$. Then OGD with appropriate step-size has regret

$$\max_{\|\mathbf{x}\|_\infty \leqslant C} \sum_{t=1}^{T} \|\mathbf{x}_t - \mathbf{x}^*(\mathbf{c}_t)\|_1 - \|\mathbf{x} - \mathbf{x}^*(\mathbf{c}_t)\|_1 \leqslant Cn\sqrt{2T} \tag{5.4}$$

Proof. The first result follows by combining Dinitz et al. [2021, Theorem 13] with Claim 5.2.1. For the third result, let $\mathbf{x}_t$ be the sequence generated by running OGD [Zinkevich, 2003] with step-size $C/\sqrt{2T}$ on the losses $U_t(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^*(\mathbf{c}_t)\|_1$ over domain $[-C, C]^n$. Since these losses are $\sqrt{n}$-Lipschitz and the duals are $C\sqrt{n}$-bounded in Euclidean norm the regret guarantee follows from Shalev-Shwartz [2011, Corollary 2.7]. For the second result, we apply standard online-to-batch conversion to the third result, i.e. we draw $T = \Omega\left(\left(\frac{Cn}{\varepsilon}\right)^2 \log \frac{1}{\delta}\right)$ samples $\mathbf{c}_t$, run OGD as above on the resulting losses U_t, and set $\hat{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}_t$ to be the average of the resulting predictions $\mathbf{x}_t$. The result follows by Lemma B.4.1. □

This concludes an overview of our two-step approach for obtaining learning guarantees for algorithms with predictions. To summarize, we propose to (1) obtain simple-to-optimize upper bounds $U_t(\mathbf{x})$ on the cost of the target algorithm on instance t as a function of prediction $\mathbf{x}$ and (2) optimize $U_t(\mathbf{x})$ using online learning. While conceptually simple, even in this illustrative example it already improves upon past work; in the sequel we demonstrate further results that this approach makes possible. Note that, like Dinitz et al. [2021], we are also able to generalize Theorem 5.2.1 to $\mathbf{b}$-matchings, which we do in Appendix 5.C.1; another advantage of our approach is that it lets us prove online and statistical learning in the case where the demand vector $\mathbf{b}$ varies across instances rather than staying fixed as in Dinitz et al. [2021]. Finally, in Appendix 5.D we also improve upon the more recent learning-theoretic results of Chen et al. [2022] for related graph algorithms with predictions problems.

5.3 Predicting requests for page migration

Equipped with our two-step approach for deriving guarantees for learning predictors, we investigate several more important problems in combinatorial optimization, starting with the page migration problem. Our results demonstrate that even for learning such simple predictors there are interesting technical challenges in deriving a *learnable* upper bound. Nevertheless, once this is accomplished the second step of our approach is again straightforward.

To introduce the online task we consider, suppose we have a server that sees a sequence of requests $s_{[1]}, \ldots, s_{[n]}$ from metric space $(\mathcal{K}, d)$ and at each timestep decides whether to change its state $a_{[i]} \in \mathcal{K}$ at cost $Dd(a_{[i-1]}, a_{[i]})$ for some $D > 1$; it then suffers a further cost $d(a_{[i]}, s_{[i]})$. The **online page migration** (OPM) problem is then to minimize the cost to the server. Recently, Indyk et al. [2022] studied a setting where we are given a sequence of predicted points $\hat{s}_{[1]}, \ldots, \hat{s}_{[n]} \in \mathcal{K}$ to aid the page migration algorithm. They show that if there exists $\gamma, q \in (0, 1)$ s.t. $\gamma D \in [n]$ and for any $i \in [n]$ we have $\sum_{j=i}^{i+\gamma D-1} 1_{s_{[j]} \neq \hat{s}_{[j]}} \leqslant q\gamma D$ then there exists an algorithm with competitive ratio $(1+\gamma)(1+\mathcal{O}(q))$ w.r.t. to the offline optimal. This algorithm depends on γ but not q, so we study the setting where γ is fixed.

5.3.1 Deriving an upper bound

As in the previous section, the predictions are discrete, so to use our approach we must convert it into a continuous problem. As we have fixed γ, the competitive ratio is an affine function of the following upper bound on q:

$$Q(\hat{s}, s) = \frac{1}{\gamma D} \max_{i \in [n-\gamma D+1]} \sum_{j=i}^{i+\gamma D-1} 1_{\hat{s}_{[j]} \neq s_{[j]}} \tag{5.5}$$

We assume that the set of points $\mathcal{K}$ is finite with indexing $k = 1, \ldots, |\mathcal{K}|$ and use this to introduce our continuous relaxation, a natural randomized approach converting the problem of learning a prediction into n experts problems on $|\mathcal{K}|$ experts. For each $j \in [n]$ we define a probability vector $\mathbf{p}_{[j]} \in \triangle_{|\mathcal{K}|}$ governing the categorical r.v. $\hat{s}_{[j]}$, i.e. $\Pr(\hat{s}_{[j]} = k) = \mathbf{p}_{[j,k]} \ \forall \ k \in \mathcal{K}$. Under these distributions the expected competitive ratio will be $(1+\gamma)(1 + \mathcal{O}(\mathbb{E}_{\hat{s} \sim \mathbf{p}} Q(\hat{s}, s)))$, for $\mathbf{p}$ the product distribution of the vectors $\mathbf{p}_j$. Note that forcing each $\mathbf{p}_j$ to be a one-hot vector recovers the original approach with no loss, so optimizing $\mathbb{E}_{\hat{s} \sim \mathbf{p}} Q(\hat{s}, s)$ over $\mathbf{p} \in \triangle_{|\mathcal{K}|}^n$ would find a predictor that fits the original result.

However, $\mathbb{E}_{\hat{s} \sim \mathbf{p}} Q(\hat{s}, s)$ is not convex in $\mathbf{p}$. The simplest relaxation is to replace the maximum by summation, but this leads to a worst-case bound of $\mathcal{O}\left(\frac{n}{\gamma D}\right)$. We instead bound $\mathbb{E}_{\hat{s} \sim \mathbf{p}} Q(\hat{s}, s)$—and thus also the expected competitive ratio—by a function of the following maximum over expectations:

$$U_s(\mathbf{p}) = \max_{i \in [n-\gamma D+1]} \mathbb{E}_{\hat{s} \sim \mathbf{p}} \sum_{j=i}^{i+\gamma D-1} 1_{\hat{s}_{[j]} \neq s_{[j]}} = \max_{i \in [n-\gamma D+1]} \sum_{j=i}^{i+\gamma D-1} 1 - \langle \mathbf{s}_{[j]}, \mathbf{p}_{[j]} \rangle \tag{5.6}$$

where $\mathbf{s}_{[j,k]} = 1_{s_{[j]}=k} \ \forall \ k \in \mathcal{K}$, i.e. $\mathbf{s}_{[j]}$ encodes the location in $\mathcal{K}$ of the jth request. As a maximum over $n - \gamma D + 1$ convex functions this objective is also convex. Note also that if $U_s(\mathbf{p})$ is zero—i.e. the probability vectors are one-hot and perfect—then $\mathbb{E}_{\hat{s} \sim \mathbf{p}} Q(\hat{s}, s) \geqslant q$ will also be zero. In fact, q is upper-bounded by a monotonically increasing function of $U_s(\mathbf{p})$ that is zero at the origin, but as this function is concave and non-Lipschitz (c.f. Figure 5.1) we incur an additive $\mathcal{O}\left(\frac{\log(n-\gamma D+1)}{\gamma D}\right)$ loss to obtain an online-learnable upper bound. This is formalized in the following result (proof in Appendix 5.A.1).

Lemma 5.3.1. There exist constants $a < e, b < 2/e$ and a monotonically increasing function $f : [0, \infty) \mapsto [0, \infty)$ s.t. $f(0) = 0$ and

$$\begin{aligned} \mathbb{E}_{\hat{s} \sim \mathbf{p}} Q(\hat{s}, s) \leqslant \frac{f(U_s(\mathbf{p}))}{\gamma D} &\leqslant \frac{a U_s(\mathbf{p}) + b \log(n - \gamma D + 1)}{\gamma D} \\ &\leqslant a \mathbb{E}_{\hat{s} \sim \mathbf{p}} Q(\hat{s}, s) + \frac{b \log(n - \gamma D + 1)}{\gamma D} \end{aligned} \tag{5.7}$$

As in bipartite matching—where we similarly resorted to a relaxation of a discrete problem—we now have a prediction-dependent convex bound on the competitive ratio for the OPM algorithm of Indyk et al. [2022]. However, whereas before we only incurred a multiplicative loss of two compared to the upper bound of Dinitz et al. [2021] (c.f. Claim 5.2.1), our convex upper

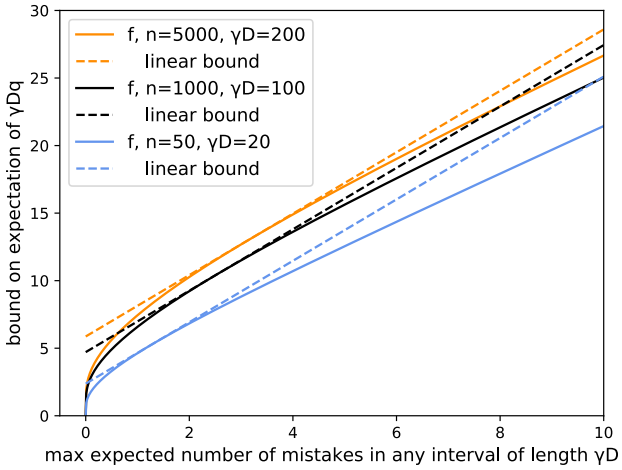

Figure 5.1: Bounds f (c.f. Lemma 5.3.1) for different n and γD on the expected largest number of mistakes in any γD-interval as a function of the maximum expected number $U_s(\mathbf{p})$.

bound for OPM also incurs an *additive* loss relative to the original prediction-dependent of Indyk et al. [2022] that makes it meaningful only for $\gamma D \gg \log n$. However, as the method we propose optimizes $U_s(\mathbf{p})$, which bounds q with no additive error via the function f in Lemma 5.3.1, in practice we may expect it to help minimize q in all regimes. Note that the non-Lipschitzness near zero that prevents using f for regret guarantees comes from poor tail behavior of Poisson-like random variables with small means, which we do not expect can be significantly improved.

5.3.2 Learning guarantees

Having established an upper bound, in Theorem 5.3.1 we again show how a learning-theoretic result follows from standard online learning. This time, instead of OGD we run **exponentiated (sub)gradient** (**EG**) [Shalev-Shwartz, 2011], a classic method for learning from experts, on each of n simplices to learn the probabilities $\mathbf{p}_{[j]}\ \forall\ j \in [n]$. The multiplicative update $\mathbf{x}_{t+1} \propto \mathbf{x}_t \odot \exp(-\alpha \nabla U_t(\mathbf{x}_t))$ of EG is notable for yielding regret logarithmic in the size $|\mathcal{K}|$ of the simplices, which is important for large metric spaces. Note that as the relaxation is randomized, our algorithms output a dense probability vector; to obtain a prediction for OPM we sample $\hat{s}_{t[j]} \sim \mathbf{p}_{[j]}\ \forall\ j \in [n]$.

Theorem 5.3.1. Let $(\mathcal{K}, d)$ be a finite metric space.

1. For any request sequence s and any set of probability vectors $\mathbf{p} \in \triangle^n_{|\mathcal{K}|}$ there exists an algorithm for OPM with expected competitive ratio
$$(1+\gamma)\left(1+\mathcal{O}\left(\frac{U_s(\mathbf{p})+\log(n-\gamma D+1)}{\gamma D}\right)\right) \tag{5.8}$$
2. There exits a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and distribution $\mathcal{D}$ over request sequences $s \in \mathcal{K}^n$ it takes $\mathcal{O}\left(\left(\frac{\gamma D}{\varepsilon}\right)^2\left(n^2 \log|\mathcal{K}| + \log\frac{1}{\delta}\right)\right)$ samples from $\mathcal{D}$ and returns $\hat{\mathbf{p}}$ s.t. w.p. $\geqslant 1-\delta$:
$$\mathbb{E}_{s\sim\mathcal{D}} U_s(\hat{\mathbf{p}}) \leqslant \min_{\mathbf{p}\in\triangle^n_{|\mathcal{K}|}} \mathbb{E}_{s\sim\mathcal{D}} U_s(\mathbf{p}) + \varepsilon \tag{5.9}$$

3. Let $s_1, \ldots, s_T$ be an adversarial sequence of request sequences. Then updating the distribution $\mathbf{p}_{t[j]}$ over $\triangle_{|\mathcal{K}|}$ at each timestep $j \in [n]$ using EG with appropriate step-size has regret

$$\max_{\mathbf{p} \in \triangle_{|\mathcal{K}|}^n} \sum_{t=1}^{T} U_{s_t}(\mathbf{p}_t) - U_{s_t}(\mathbf{p}) \leqslant \gamma D n \sqrt{2T \log |\mathcal{K}|} \tag{5.10}$$

Proof. The first result follows by combining Indyk et al. [2022, Theorem 1] with Lemma 5.3.1. For the third let $\mathbf{p}_t$ be generated by running n exponentiated gradient algorithms with step-size $\sqrt{\frac{\log |\mathcal{K}|}{2\gamma^2 D^2 T}}$ on losses $U_{s_t}(\mathbf{p})$ over $\triangle_{|\mathcal{K}|}^n$. Since these are γD-Lipschitz and the maximum entropy is $\log |\mathcal{K}|$, the regret follows by [Shalev-Shwartz, 2011, Theorem 2.15]. For the second result, apply standard online-to-batch conversion to the third, i.e. draw $T = \Omega\left(\left(\frac{\gamma D}{\varepsilon}\right)^2 \left(n^2 \log |\mathcal{K}| + \log \frac{1}{\delta}\right)\right)$ samples $\mathbf{s}_t$, run EG on $U_{s_t}(\mathbf{p})$ as above, and set $\hat{\mathbf{p}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_t$ to be the average of the resulting actions. The result follows by Lemma B.4.1. □

As before, this result first shows how the quantity of interest—here the competitive ratio—is upper-bounded by an affine function of some quality measure $U_s(\mathbf{p})$, for which we then provide regret and statistical guarantees using online learning. The difficulty deriving a suitable bound exemplifies the technical challenges that arise in learning predictors, and may also be encountered in other sequence prediction problems such as TCP [Bamas et al., 2020]. Nevertheless, our approach does yield an online procedure that incurs only $\mathcal{O}(\frac{\log n}{\gamma D})$ additive error over Indyk et al. [2022] in the case of a perfect predictor and, unlike their work, we provide an algorithm for learning the predictor itself. In Appendix 5.C.2 we also show an auto-regressive extension which does *not* require learning a distribution for each timestep $j \in [n]$.

5.4 Learning linear predictors with instance-feature inputs

So far we have considered only *fixed* predictors, either optima-in-hindsight in the online setting or a population risk minimizers for i.i.d. data. Actual instances can vary significantly and so a fixed predictor may not be very good, e.g. in the example of querying a sorted array it means always returning the same index. In the online setting one can consider methods that adapt to dynamic comparators [Zinkevich, 2003, Jadbabaie et al., 2015, Mokhtari et al., 2016], which are also applicable to our upper bounds; however, these still need measures such as the comparator path-length to be small, which may be more reasonable in some cases but not all.

We instead study the setting where all instances come with instance-specific features, a natural and practical assumption [Kraska et al., 2018, Lattanzi et al., 2020] that encompasses numerical representations of the instance itself—e.g. bits representing a query or a graph—or other information such as weather or day of the week. These are passed to functions—e.g. linear predictors, neural nets, or trees—whose parameters can be learned from data. We study linear predictors, which are often amenable to similar analyses as above since the composition of a convex and affine function is convex. For example, it is straightforward to extend the matching results to learning linear predictors of duals. OPM is more challenging because the outputs must lie in the simplex, which can be solved by learning rectangular stochastic matrices. Both sets of results

are shown in Appendix 5.C. Notably, for page migration our guarantees cover the auto-regressive setting where the server probabilities are determined by a fixed linear transform of past states.

Our main example will be online job scheduling via minimizing the fractional makespan [Lattanzi et al., 2020], where we must assign each in a sequence of variable-sized jobs to one of m machines. Lattanzi et al. [2020] provide an algorithm that uses predictions $\hat{\mathbf{w}} \in \mathbb{R}^m_{>0}$ of "good" machine weights $\mathbf{w} \in \mathbb{R}^m_{>0}$ to assign jobs based on how well $\hat{\mathbf{w}}$ corresponds to machine demand; the method has a performance guarantee of $\mathcal{O}(\log \min\{\max_i \frac{\hat{\mathbf{w}}_{[i]}}{\mathbf{w}_{[i]}}, m\})$. They also discuss learning linear and other predictors, but without guarantees. We study linear prediction of the *logarithm* of the machine weights, which makes the problem convex, and assume features lie in the f-dimensional simplex. For simplicity we only consider learning the linear transform from features to predictors and not the intercept, as the former subsumes the latter. For the online result, we use KT-OCO [Orabona and Pal, 2016, Algorithm 1], a parameter-free subgradient method with update $\mathbf{x}_{t+1} \leftarrow \frac{1+\sum_{s=1}^t \langle \mathbf{g}_s, \mathbf{x}_s \rangle}{t+1} \sum_{s=1}^t \mathbf{g}_s$ for $\mathbf{g}_s = \nabla U_s(\mathbf{x}_s)$; it allows us to not assume any bound on the machine weights and thus to compete with the optimal linear predictor in all of $\mathbb{R}^{m \times f}$.

Theorem 5.4.1. Consider online restricted assignment with $m \geqslant 1$ machines [Lattanzi et al., 2020, Section 2.1].

1. For predicted logits $\mathbf{x} \in \mathbb{R}^m$ there is an algorithm whose fractional makespan has competitive ratio
$$\mathcal{O}(\min\{\|\mathbf{x} - \log \mathbf{w}\|_\infty, \log m\}) \leqslant \mathcal{O}(U(\mathbf{x})) \tag{5.11}$$
for $U(\mathbf{x}) = \|\mathbf{x} - \log \mathbf{w}\|_\infty$, where $\mathbf{w} \in \mathbb{R}^m_{>0}$ are good machine weights [Lattanzi et al., 2020, Section 3].
2. There exists a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and distribution $\mathcal{D}$ over machine (weight, feature) pairs $(\mathbf{w}, \mathbf{f}) \in \mathbb{R}^m_{>0} \times \triangle_f$ s.t. $\|\log \mathbf{w}\|_\infty \leqslant B$ the algorithm takes $\mathcal{O}\left(\left(\frac{B}{\varepsilon}\right)^2 \left(mf + \log \frac{1}{\delta}\right)\right)$ samples from $\mathcal{D}$ and returns $\hat{\mathbf{A}} \in \mathbb{R}^{m \times f}$ s.t. w.p. $\geqslant 1 - \delta$
$$\mathbb{E}_{(\mathbf{w},\mathbf{f}) \sim \mathcal{D}} \|\hat{\mathbf{A}}\mathbf{f} - \log \mathbf{w}\|_\infty \leqslant \min_{\|\mathbf{A}\|_{\max} \leqslant B} \mathbb{E}_{(\mathbf{w},\mathbf{f}) \sim \mathcal{D}} \|\mathbf{A}\mathbf{f} - \log \mathbf{w}\|_\infty + \varepsilon \tag{5.12}$$
3. Let $(\mathbf{w}_1, \mathbf{f}_1), \ldots, (\mathbf{w}_T, \mathbf{f}_T) \in \mathbb{R}^m_{>0} \times \triangle_f$ be an adversarial sequence of (weights, feature) pairs. Then for any $\mathbf{A} \in \mathbb{R}^{m \times f}$ KT-OCO has regret
$$\sum_{t=1}^T \|\mathbf{A}_t\mathbf{f}_t - \log \mathbf{w}_t\|_\infty - \|\mathbf{A}\mathbf{f}_t - \log \mathbf{w}_t\|_\infty \leqslant \|\mathbf{A}\|_F \sqrt{T \log(1 + 24T^2 \|\mathbf{A}\|_F^2)} + 1 \tag{5.13}$$
If the matrices have B-bounded entries then OGD with appropriate step-size has regret
$$\max_{\|\mathbf{A}\|_{\max} \leqslant B} \sum_{t=1}^T \|\mathbf{A}_t\mathbf{f}_t - \log \mathbf{w}_t\|_\infty - \|\mathbf{A}\mathbf{f}_t - \log \mathbf{w}_t\|_\infty \leqslant B\sqrt{2mfT} \tag{5.14}$$

Proof. The first result follows by substituting $\max_i \frac{\exp(\mathbf{x}_{[i]})}{\mathbf{w}_{[i]}}$ for η in Lattanzi et al. [2020, Theorem 3.1] and upper bounding the maximum by the ℓ_∞-norm. For the third, since U_t is 1-Lipschitz w.r.t. the Euclidean norm we apply the guarantee for KT-OCO [Orabona and Pal, 2016, Algorithm 1] using $\varepsilon = 1$ and the subgradients of $\|\mathbf{A}_t\mathbf{f}_t - \log \mathbf{w}_t\|_\infty$ as rewards [Orabona and Pal, 2016,

Corollary 5]. The result for B-bounded $\mathbf{A}$ follows by applying OGD with step-size $B\sqrt{\frac{mf}{2T}}$ over $\|\mathbf{A}\|_{\max} \leqslant B$ [Shalev-Shwartz, 2011, Corollary 2.7]. Finally, the second result follows by applying online-to-batch conversion to the latter result, i.e. draw $T = \Omega\left(\left(\frac{B}{\varepsilon}\right)^2 \left(mf + \log\frac{1}{\delta}\right)\right)$ samples $(\mathbf{w}_t, \mathbf{f}_t)$, run OGD on the resulting losses $\|\mathbf{A}\mathbf{f}_t - \log \mathbf{w}_t\|_\infty$ as above, and set $\hat{\mathbf{A}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{A}_t$ to be the average of the resulting actions $\mathbf{A}_t$. The result follows by Lemma B.4.1. □

This is the first guarantee for learning non-static predictors in the algorithms with predictions literature. It demonstrates both how to extend static predictor results to learning linear predictors—the former is recovered by $\mathbf{f}_t = \mathbf{1}_1 \;\forall\; t$—and how to handle unbounded predictor domains. The ability to provide such guarantees is another advantage of our approach.

5.5 Tuning parameterized robustness-consistency tradeoffs

We turn to tuning robustness-consistency tradeoffs, introduced in Lykouris and Vassilvitskii [2021]. As discussed in the previous chapter, this tradeoff captures the tension between following the predictions when they are good (consistency) and doing not much worse than the worst-case guarantee in either case (robustness). We focus on the case where the tradeoff is made explicit via a parameter $\lambda \in [0, 1]$, the setting of which is crucial but often left to the end-user. Here we show that it is often eminently learnable in an online setting. We then demonstrate how to accomplish a much harder task—tuning λ at the same time as learning to predict—on two related but technically very different variants of the ski-rental problem. This meta-application highlights the applicability of our approach to nonconvex upper bounds.

5.5.1 Robustness-consistency tradeoffs

Recall the parameterized robustness-consistency tradeoff $U_t(\mathbf{x}, \lambda) = \min\{f(\lambda)u_t(\mathbf{x}), g_t(\lambda)\}$ from Equation 4.1, where u_t is some measure of the quality of $\mathbf{x}$ on instance t, f is a monotonically increasing function that ideally satisfies $f(0) = 1$, and g_t is a monotonically decreasing function that (for online algorithms) ideally evaluates to the worst-case competitive ratio at $\lambda = 1$. For example, in job scheduling with predictions, a setting where we are given n jobs and their predicted runtimes with total absolute error η and must minimize the sum of their completion times when running on a single server with pre-emption. Here Kumar et al. [2018, Theorem 3.3] showed that a preferential round-robin algorithm has competitive ratio at most $\min\left\{\frac{1+2\eta/n}{1-\lambda}, \frac{2}{\lambda}\right\}$. Thus if we know the prediction is perfect we can set $\lambda = 0$ and obtain the optimal cost (consistency); on the other hand, if we know the prediction is poor we can set $\lambda = 1$ and get the (tight) worst-case guarantee of two (robustness).

Of course in practice we often do not know how good a prediction is on a specific instance t; we thus would like to learn to set λ, i.e. to learn how trustworthy our prediction is. As a first step, we can consider doing so when we are given a prediction for each instance and thus only need to optimize over λ. For example, the just-discussed problem of job scheduling has competitive ratio upper-bounded by $U_t(\lambda) = \min\left\{\frac{1+2\eta_t/n_t}{1-\lambda}, \frac{2}{\lambda}\right\}$ for n_t and η_t the number of jobs and the prediction

quality, respectively, on instance t. Assuming a bound B on the average error makes U_t Lipschitz, so we can apply the **exponentially weighted average forecaster** [Krichene et al., 2015, Algorithm 1], also known as the **exponential forecaster**. This algorithm, whose action at each time $t+1$ is to sample from the distribution with density $\rho_{t+1}(\cdot) \propto \rho_1(\cdot)\exp(-\alpha\sum_{s=1}^{t} U_s(\cdot))$, has the following regret guarantee (proof in Appendix 5.A.2):

Corollary 5.5.1. For the competitive ratio upper bounds U_t of the job scheduling problem with average prediction error η/n_t at most B the exponential forecaster with appropriate step-size has expected regret

$$\max_{\lambda\in[0,1]} \mathbb{E}\sum_{t=1}^{T} U_t(\lambda_t) - U_t(\lambda) \leqslant 9B\left(1+\sqrt{\frac{T}{2}\log T}\right) \tag{5.15}$$

Thus a standard method produces a sequence λ_t that performs as well as the best λ asymptotically. Next we study the harder problem of simultaneously tuning λ *and* learning to predict.

5.5.2 Ski-rental

We instantiate this challenge on ski-rental, where each task t is a ski season with an unknown number of days $n_t \in \mathbb{Z}_{\geqslant 2}$; to ski each day, we either buy skis at price b_t or rent each day for the price of one. The optimal offline policy is to buy iff $b_t < n_t$, and the best algorithm has worst-case competitive ratio $e/(e-1)$. Kumar et al. [2018] and Bamas et al. [2020, Theorem 2] further derive an algorithm with the following robustness-consistency tradeoff between blindly following a prediction x and incurring cost $u_t(x) = b_t 1_{x>b_t} + n_t 1_{x\leqslant b_t}$ or going with the worst-case guarantee:

$$U_t(x,\lambda) = \frac{\min\{\lambda u_t(x), b_t, n_t\}}{1-e_t(-\lambda)}, e_t(z) = (1+1/b_t)^{b_t z} \tag{5.16}$$

Assuming a bound of $N \geqslant 2$ on the number of days and $B > 0$ on the buy price implies that U_t is bounded and Lipschitz w.r.t. λ. We can thus run exponentiated gradient on the functions U_t to learn a categorical distribution over the product set $[N] \times \{\delta/2, \ldots, 1-\delta/2\}$ for some δ s.t. $1/\delta \in \mathbb{Z}_{\geqslant 2}$. This yields the following bound on the expected regret (proof in Appendix 5.A.3).

Corollary 5.5.2. For the competitive ratio upper bounds U_t of the discrete ski-rental problem the randomized exponentiated gradient algorithm with an appropriate step-size has expected regret

$$\max_{x\in[N],\lambda\in(0,1]} \mathbb{E}\sum_{t=1}^{T} U_t(x_t,\lambda_t) - U_t(x,\lambda) \leqslant 6N\sqrt{T\log(BNT)} \tag{5.17}$$

Thus via an appropriate discretization the sequence of predictions (x_t, λ_t) does as well as the joint optimum on this problem. However, we can also look at a case where we are not able to just discretize to get low regret. In particular, we consider the *continuous* ski-rental problem, where each day $n_t > 1$ is a real number, and study how to pick thresholds x after which to buy skis, which has cost $u_t(x) = n_t 1_{n_t\leqslant x} + (b_t+x)1_{n_t>x}$. Note that $x=0$ and $x=N$ recovers the previous setting where our decision was to buy or not at the beginning. For this setting, Diakonikolas et al. [2021] adapt an algorithm of Mahdian et al. [2012] to bound the cost as follows:

$$C_t(x,\lambda) \leqslant U_t(x,\lambda) = \min\left\{\frac{u_t(x)}{1-\lambda}, \frac{e\min\{n_t,b_t\}}{(e-1)\lambda}\right\} \tag{5.18}$$

While the bound is simpler in λ, it is discontinuous in x because u_t is piecewise-Lipschitz. Since one cannot even attain sublinear regret on adversarially chosen threshold functions, we must make an assumption on the data. In particular, we will assume the days are *dispersed*:

Definition 5.5.1. A set of points $n_1, \ldots, n_T \in \mathbb{R}$ is **β-dispersed** if $\forall\ \varepsilon \geqslant T^{-\beta}$ the expected number in any ε-ball is $\tilde{\mathcal{O}}(\varepsilon T)$, i.e. $\mathbb{E} \max_{x \in [0,N]} \left|[x \pm \varepsilon] \cap \{n_1, \ldots, n_T\}\right| = \tilde{\mathcal{O}}(\varepsilon T)$.

Dispersion encodes the stipulation that the days, and thus the discontinuities of $u_t(x, \lambda)$, are not too concentrated. In the i.i.d. setting, a simple condition that leads to dispersion with $\beta = 1/2$ is the assumption that the points are drawn from a κ-bounded distribution [Balcan et al., 2018b, Lemma 1]. Notably this is a strictly weaker assumption than the log-concave requirement of Diakonikolas et al. [2021] that they used to show statistical learning results for ski-rental. Having stipulated that the ski-days are β-dispersed, we can show that it implies dispersion of the loss functions [Balcan et al., 2018b] and thus obtain the following guarantee for the exponential forecaster applied to $U_t(x, \lambda)$ (proof in Appendix 5.A.4):

Corollary 5.5.3. For cost upper bounds U_t of the continuous ski-rental problem the exponential forecaster with an appropriate step-size has expected regret

$$\max_{x \in [0,N], \lambda \in (0,1]} \mathbb{E} \sum_{t=1}^{T} U_t(x_t, \lambda_t) - U_t(x, \lambda) \leqslant \tilde{\mathcal{O}}\left(\sqrt{T \log(NT)} + (N+B)^2 T^{1-\beta}\right) \tag{5.19}$$

Thus in two mathematically quite different settings of ski-rental we can directly apply online learning to existing bounds to not only learn online the best action for ski-rental, but to at the same time learn how trustworthy the best action is via tuning the robustness-consistency tradeoff.

5.6 Conclusion

The field of algorithms with predictions has been successful in circumventing worst case lower bounds and showing how simple predictions can improve algorithm performance. However, except for a few problem-specific approaches, the question of *how* to predict has largely been missing from the discussion. In this chapter we presented the first general framework for efficiently learning useful predictions and applied it to a diverse set of previously studied problems, giving the first low regret learning algorithms, reducing sample complexity bounds, and showing how to learn the best robustness-consistency tradeoff. One current limitation is the lack of more general-case guarantees for *simultaneously* tuning robustness-consistency and learning the predictor, which we only show for ski-rental. There are also several other avenues for future work. The first is to build on our results and provide learning guarantees for other problems where the algorithmic question of *how* to use predictions is already addressed. Another is to try to improve known bounds by solving the problems holistically: developing easy-to-learn parameters in concert with developing algorithms that can use them. We make progress in this direction in the next chapter. Finally, there is the direction of identifying hard problems: what are the instances where no reasonable prediction can help improve an algorithm's performance?

5.A Proofs of main results

5.A.1 Proof of Lemma 5.3.1

Proof. Note that the second line follows directly by Jensen's inequality, so we focus on showing the first two inequalities. For each $j \in [n]$ define $p_j = 1 - \langle \mathbf{s}_{[j]}, \mathbf{p}_{[j]} \rangle$, i.e. the probability that $\hat{s}_j \neq s_j$, and the r.v. $X_j \sim \mathrm{Ber}(p_j)$. Define also the r.v. $S_i = \sum_{j=i}^{i+\gamma D-1} X_j$, s.t. we have

$$\gamma D \mathbb{E}_{\mathbf{p}} q = \mathbb{E}_{\mathbf{p}} \max_{i \in [n-\gamma D+1]} S_i = \mathbb{E}_{\mathbf{p}} \max_{i \in [n-\gamma D+1]} \sum_{j=i}^{i+\gamma D-1} X_j \tag{5.20}$$

Note that S_i is a Poisson binomial and so has moment-generating function $\mathbb{E}_{\mathbf{p}} \exp(tS_i) = \prod_{j=i}^{i+\gamma D-1}(1 - p_j + p_j e^t)$. Therefore applying Jensen's inequality and the union bound yields

$$\begin{aligned}
\exp\left(t \mathbb{E}_{\mathbf{p}} \max_{i \in [n-\gamma D+1]} S_i\right) \leqslant \mathbb{E}_{\mathbf{p}} \exp\left(t \max_{i \in [n-\gamma D+1]} S_i\right) &= \mathbb{E}_{\mathbf{p}} \max_{i \in [n-\gamma D+1]} \exp\left(tS_i\right) \\
&\leqslant \sum_{i=1}^{n-\gamma D+1} \mathbb{E}_{\mathbf{p}} \exp\left(tS_i\right) \\
&\leqslant (n - \gamma D + 1) \prod_{j=i^*}^{i^*+\gamma D-1} (1 - p_j + p_j e^t)
\end{aligned} \tag{5.21}$$

for all $t > 0$ and $i^* \in \arg\max_{i \in [n-\gamma D+1]} \mathbb{E}_{\mathbf{p}} S_i$. We then have

$$\begin{aligned}
t \mathbb{E}_{\mathbf{p}} \max_{i \in [n-\gamma D+1]} S_i &\leqslant \log(n - \gamma D + 1) + \sum_{j=i^*}^{i^*+\gamma D-1} \log(1 - p_j + p_j e^t) \\
&\leqslant \log(n - \gamma D + 1) + \sum_{j=i^*}^{i^*+\gamma D-1} \log \exp(p_j(e^t - 1)) \\
&\leqslant \log(n - \gamma D + 1) + \sum_{j=i^*}^{i^*+\gamma D-1} p_j(e^t - 1) \\
&\leqslant \log(n - \gamma D + 1) + \mathbb{E}_{\mathbf{p}} S_{i^*}(e^t - 1)
\end{aligned} \tag{5.22}$$

Dividing by $t = W\left(\frac{\log(n-\gamma D+1)}{x}\right) + 1$ shows that $f(x) = \frac{x(\exp(t)-1)+\log(n-\gamma D+1)}{t\gamma D}$, where $W : [0, \infty) \mapsto [0, \infty)$ is the Lambert W-function. Define $L = \log(n-\gamma D+1)/e$, so we are interested in bounding $f(x) = \frac{x(\exp(W(L/x)+1)-1)+eL}{W(L/x)+1}$. We compute its derivative:

$$f'(x) = \frac{x(e^{W(L/x)+1} - 1)W(L/x)^2 - 3x(W(L/x) + 1/3) + xe^{W(L/x)+1} + 2eL}{x(W(L/x) + 1)^3} \tag{5.23}$$

and second derivative:

$$f''(x) = -\frac{W(L/x)\left((x+eL)W(L/x)^2 + 2(2x+eL)W(L/x) + eL\right)}{x^2(W(L/x)+1)^5} \tag{5.24}$$

Since the second derivative is always negative, f is a concave function on $x \geqslant 0$. Thus for $\omega = W(1)$ we have

$$\begin{aligned} f(x) &\leqslant \min_{y>0} f(y) + f'(y)(x-y) \\ &\leqslant \frac{L(e/\omega - 1 + e)}{\omega+1} + \frac{L(e/\omega-1)\omega^2 - 3L(\omega+1/3) + Le/\omega + 2eL}{L(\omega+1)^3}(x-L) \\ &= \left(e/\omega + 1/(\omega+1)^3 - (e+1)/(\omega+1) - 1/(\omega+1)^2\right)x \\ &\quad + \left(1/(\omega+1)^2 + e/(\omega+1) - 1/(\omega+1)^3\right)L \\ &< ex + \frac{2}{e}\log(n - \gamma D + 1) \end{aligned} \tag{5.25}$$

□

5.A.2 Proof of Corollary 5.5.1

Proof. We have that $U_t(\lambda)$ is bounded above by $3(1+2B)$, its largest gradient is attained at $2/(3+2\eta_t/n)$ where it is bounded by $(3+2B)/2$. Applying Krichene et al. [2015, Corollary 2] and simplifying yields the result. □

5.A.3 Proof of Corollary 5.5.2

Proof. $U_t(x,\lambda)$ is bounded above by $2N$ and its largest gradient is attained at $\lambda = \frac{\min\{b_t,n_t\}}{u_t(x)} \geqslant \frac{1}{N}$ with norm bounded by $\frac{B\exp(1/N)}{(\exp(1/N)-1)^2}$. Let $\Lambda = \{k\delta\}_{k=1}^{\lfloor 1/\delta \rfloor}$ for some $\delta \in (0,1]$. Then we run EG on the simplex over $[N] \times \Lambda$ and with step-size $\frac{1}{2N}\sqrt{\frac{\log\frac{N}{\delta}}{2T}}$ to obtain regret compared to the best element of $[N] \times \Lambda$ of $2N\sqrt{2T\log\frac{N}{\delta}}$ [Shalev-Shwartz, 2011, Theorem 2.15]. Setting

$\delta = \min\left\{\frac{N(\exp(1/N)-1)^2}{B\exp(1/N)}\sqrt{\frac{2}{T}}, 1\right\}$ yields

$$\begin{aligned}
\mathbb{E}\sum_{t=1}^{T} U_t(x_t,\lambda_t) &\leqslant 2N\sqrt{2T\log(N\lfloor 1/\delta\rfloor)} + \min_{x\in[N],\lambda\in\Lambda}\sum_{t=1}^{T} U_t(x,\lambda) \\
&\leqslant 2N\sqrt{2T\log\frac{N}{\delta}} + \frac{B\exp(1/N)\delta T}{(\exp(1/N)-1)^2} + \min_{x\in[N],\lambda\in(0,1]}\sum_{t=1}^{T} U_t(x,\lambda) \\
&\leqslant 2N\sqrt{2T\left(\log(BT) + \max\left\{\log N, \frac{1}{N} - 2\log\left(\exp\left(\frac{1}{N}\right)-1\right)\right\}\right)} \\
&\quad + N\sqrt{2T} + \min_{x\in[N],\lambda\in(0,1]}\sum_{t=1}^{T} U_t(x,\lambda) \\
&\leqslant 6N\sqrt{T\log(BNT)} + \min_{x\in[N],\lambda\in(0,1]}\sum_{t=1}^{T} U_t(x,\lambda)
\end{aligned} \tag{5.26}$$

□

5.A.4 Proof of Corollary 5.5.3

Proof. $U_t(x,\lambda)$ is bounded above by $e(N+B)$, its largest gradient w.r.t. λ is attained at $\lambda = \frac{e\min\{n_t,b_t\}}{(e-1)u_t(x)+e\min\{n_t,b_t\}}$, where it is bounded by $\left(\frac{2e(N+B)}{e-1}\right)^2$, and its largest gradient w.r.t. x is $e(N+B)$. Thus the function is $5e(N+B)^2$-Lipschitz w.r.t. the Euclidean norm, apart from discontinuities at $x = n_t$. Now, note that our assumption that the points $n_1,\dots,n_T$ are β-dispersed implies exactly that the functions U_t are β-dispersed (c.f. Definition 2.3.1), so the exponentially-weighted forecaster attains expected regret $\tilde{\mathcal{O}}\left(\sqrt{T\log(NT)} + (N+B)^2T^{1-\beta}\right)$. □

5.B b-matching

Definition 5.B.1. For $\mathbf{b}\in\mathbb{R}^n_{\geqslant 0}$ the **b-seminorm** $\|\cdot\|_{\mathbf{b},1}:\mathbb{R}^n\mapsto\mathbb{R}_{\geqslant 0}$ is $\|\mathbf{x}\|_{\mathbf{b},1} = \sum_{i=1}^n \mathbf{b}_{[i]}|\mathbf{x}_{[i]}|$.

Claim 5.B.1. Given any vectors $\mathbf{x}\in\mathbb{Z}^n$ and $\mathbf{y}\in\mathbb{R}^n$, let $\tilde{\mathbf{y}}\in\mathbb{Z}^n$ be the vector whose elements are those of $\mathbf{y}$ rounded to the nearest integer. Then for all $\mathbf{b}\in\mathbb{Z}^n$ we have $\|\mathbf{x}-\tilde{\mathbf{y}}\|_{\mathbf{b},1} \leqslant 2\|\mathbf{x}-\mathbf{y}\|_{\mathbf{b},1}$.

Proof. Let $S\subset[n]$ be the set of indices $i\in[n]$ for which $\mathbf{x}_{[i]} \geqslant \mathbf{y}_{[i]} \iff \tilde{\mathbf{y}}_{[i]} = \lceil\mathbf{y}_{[i]}\rceil$. For

$i \in [n] \backslash S$ we have $|\mathbf{x}_{[i]} - \mathbf{y}_{[i]}| \geqslant 1/2 \geqslant |\tilde{\mathbf{y}}_{[i]} - \mathbf{y}_{[i]}|$ so it follows by the triangle inequality that

$$\begin{aligned}
\|\mathbf{x} - \tilde{\mathbf{y}}\|_{\mathbf{b},1} &= \sum_{i \in S} \mathbf{b}_{[i]} |\mathbf{x}_{[i]} - \tilde{\mathbf{y}}_{[i]}| + \sum_{i \in [n] \backslash S} \mathbf{b}_{[i]} |\mathbf{x}_{[i]} - \tilde{\mathbf{y}}_{[i]}| \\
&\leqslant \sum_{i \in S} \mathbf{b}_{[i]} |\mathbf{x}_{[i]} - \mathbf{y}_{[i]}| + \sum_{i \in [n] \backslash S} \mathbf{b}_{[i]} (|\mathbf{x}_{[i]} - \mathbf{y}_{[i]}| + |\mathbf{y}_{[i]} - \tilde{\mathbf{y}}_{[i]}|) \\
&\leqslant \sum_{i \in S} \mathbf{b}_{[i]} |\mathbf{x}_{[i]} - \mathbf{y}_{[i]}| + 2 \sum_{i \in [n] \backslash S} \mathbf{b}_{[i]} |\mathbf{x}_{[i]} - \mathbf{y}_{[i]}| \leqslant 2 \|\mathbf{x} - \mathbf{y}\|_{\mathbf{b},1}
\end{aligned} \tag{5.27}$$

□

Theorem 5.B.1. Suppose we have a fixed graph with $n \geqslant 3$ vertices and $m \geqslant 1$ edges.

1. For any cost vector $\mathbf{c} \in \mathbb{Z}^m_{\geqslant 0}$, any demand vector $\mathbf{b} \in \mathbb{Z}^n_{\geqslant 0}$, and any dual vector $\mathbf{x} \in \mathbb{R}^n$ there exists an algorithm for minimum weight perfect $\mathbf{b}$-matching that runs in time $\tilde{\mathcal{O}}\left(mnU(\mathbf{x})\right)$, where $U(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^*(\mathbf{c}, \mathbf{b})\|_{\mathbf{b},1}$ for $\mathbf{x}^*(\mathbf{c}, \mathbf{b})$ the optimal dual vector associated with $\mathbf{c}$ and $\mathbf{b}$.
2. There exists a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and any distribution $\mathcal{D}$ over (cost, demand) vector pairs in $\mathbb{Z}^m_{\geqslant 0} \times \mathbb{Z}^n_{\geqslant 0}$ with respective ℓ_∞-norms bounded by C and B the algorithm takes $\mathcal{O}\left(\left(\frac{CBn}{\varepsilon}\right)^2 \log \frac{1}{\delta}\right)$ samples from $\mathcal{D}$ and returns $\hat{\mathbf{x}}$ s.t. w.p. $\geqslant 1 - \delta$:
$$\mathbb{E}_{(\mathbf{c},\mathbf{b}) \sim \mathcal{D}} \|\hat{\mathbf{x}} - \mathbf{x}^*(\mathbf{c}, \mathbf{b})\|_{\mathbf{b},1} \leqslant \min_{\|\mathbf{x}\|_\infty \leqslant C} \mathbb{E}_{(\mathbf{c},\mathbf{b}) \sim \mathcal{D}} \|\mathbf{x} - \mathbf{x}^*(\mathbf{c}, \mathbf{b})\|_{\mathbf{b},1} + \varepsilon \tag{5.28}$$
3. Let $(\mathbf{c}_1, \mathbf{b}_1), \ldots, (\mathbf{c}_T, \mathbf{b}_T) \in \mathbb{Z}^m_{\geqslant 0} \times \mathbb{Z}^n_{\geqslant 0}$ be an adversarial sequence of (cost, demand) vector pairs with ℓ_∞-norms bounded by C and B, respectively. Then OGD with appropriate step-size has regret
$$\max_{\|\mathbf{x}\|_\infty \leqslant C} \sum_{t=1}^{T} \|\mathbf{x}_t - \mathbf{x}^*(\mathbf{c}_t, \mathbf{b}_t)\|_{\mathbf{b}_t,1} - \|\mathbf{x} - \mathbf{x}^*(\mathbf{c}_t, \mathbf{b}_t)\|_{\mathbf{b}_t,1} \leqslant CBn\sqrt{2T} \tag{5.29}$$

Proof. The first result follows by Dinitz et al. [2021, Theorem 31] and Claim 5.B.1. For the third, let $\mathbf{x}_t$ be the sequence generated by running OGD [Zinkevich, 2003] with step-size $\frac{C}{B\sqrt{2T}}$ on the losses $U_t(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^*(\mathbf{c}_t, \mathbf{b}_t)\|_{\mathbf{b}_t,1}$ over domain $[-C, C]^n$. Since these losses are $B\sqrt{n}$-Lipschitz and the duals are $C\sqrt{n}$-bounded in Euclidean norm the regret guarantee follows from Shalev-Shwartz [2011, Corollary 2.7]. For the second result, apply online-to-batch conversion to the third result, i.e. draw $T = \Omega\left(\left(\frac{CBn}{\varepsilon}\right)^2 \log \frac{1}{\delta}\right)$ samples $(\mathbf{c}_t, \mathbf{b}_t)$, run OGD as above on the resulting losses U_t, and set $\hat{\mathbf{x}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{x}_t$ to be the average of the resulting predictions $\mathbf{x}_t$. Applying Lemma B.4.1 yields the result. □

5.C Learning linear predictors with instance-feature inputs

Computational instances on which we want to run algorithms with predictions often come with instance-specific features, e.g. ones derived from text descriptions of the instance or summary

statistics about related graphs or environments [Kraska et al., 2018, Lattanzi et al., 2020]. It is thus natural to learn parameterized functions, e.g. linear mappings or neural networks, from these features to predictions. However, there has been very little work, in either the statistical or online setting, showing that such predictions are learnable. In this section we show how our framework naturally handles this setting by exploiting the convexity of compositions of convex and affine functions, resulting in the first formal guarantees for linear predictors for algorithms with predictions. While the first application to the matching problem of Dinitz et al. [2021] is a straightforward extension, we also show how to handle more complicated cases, such as when the output space is constrained to probability simplices as in the page migration problem. Note we assume all feature vectors lie in the f-dimensional simplex; this is generally easy to accomplish by normalization. For simplicity we also only consider learning the linear transform from features to predictors and not the intercept, as the latter follows from the former by appending an extra dimension with value $1/2$ to the feature vector and doubling the bound on the norm of the linear transform.

5.C.1 b-matching

Our first application for learning mappings from instance features is to the $\mathbf{b}$-matching setting. Note that the learning-theoretic results for the regular bipartite matching setting in Section 5.2 follow directly by setting $\mathbf{b} = \mathbf{1}_n$ for all instances, and that the learning-theoretic results of Theorems 5.2.1 and 5.B.1 are also special cases of the following when $\mathbf{f} = \mathbf{1}_1$ for all instances. Note that we optimize only over $\mathbf{A} \in [-C, C]^{n \times f}$, but unlike in the $f = 1$ case the optimal $\mathbf{A}$ may be unbounded; to handle that setting, one could again use an algorithm such as KT-OCO that does not depend on knowing the set size [Orabona and Pal, 2016].

Theorem 5.C.1. Consider the setting of Theorem 5.B.1.

1. There exists a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and any distribution $\mathcal{D}$ over (cost, demand, feature) vector triples in $\mathbb{Z}^m_{\geqslant 0} \times \mathbb{Z}^n_{\geqslant 0} \times \triangle_f$ s.t. the respective ℓ_∞-norms of the first two are bounded by C and B, respectively, the algorithm takes $\mathcal{O}\left(\left(\frac{CBn}{\varepsilon}\right)^2 \left(f^2 + \log \frac{1}{\delta}\right)\right)$ samples from $\mathcal{D}$ and returns $\hat{\mathbf{A}} \in \mathbb{R}^{n \times f}$ s.t. w.p. $\geqslant 1 - \delta$:

$$\mathbb{E}_{(\mathbf{c},\mathbf{b},\mathbf{f})\sim\mathcal{D}} \|\hat{\mathbf{A}}\mathbf{f} - \mathbf{x}^*(\mathbf{c}, \mathbf{b})\|_{\mathbf{b},1} \leqslant \min_{\|\mathbf{A}\|_{\max} \leqslant C} \mathbb{E}_{(\mathbf{c},\mathbf{b},\mathbf{f})\sim\mathcal{D}} \|\mathbf{A}\mathbf{x} - \mathbf{x}^*(\mathbf{c}, \mathbf{b})\|_{\mathbf{b},1} + \varepsilon \tag{5.30}$$

2. Let $(\mathbf{c}_1, \mathbf{b}_1, \mathbf{f}_1), \ldots, (\mathbf{c}_T, \mathbf{b}_T, \mathbf{f}_T) \in \mathbb{Z}^m_{\geqslant 0} \times \mathbb{Z}^n_{\geqslant 0} \times \triangle_f$ be an adversarial sequence of (cost, demand, feature) vector triples s.t. the ℓ_∞-norms of the first two are bounded by C and B, respectively. Then OGD with appropriate step-size has regret

$$\max_{\|\mathbf{A}\|_{\max} \leqslant C} \sum_{t=1}^{T} \|\mathbf{A}_t\mathbf{f}_t - \mathbf{x}^*(\mathbf{c}_t, \mathbf{b}_t)\|_{\mathbf{b}_t,1} - \|\mathbf{A}\mathbf{f}_t - \mathbf{x}^*(\mathbf{c}_t, \mathbf{b}_t)\|_{\mathbf{b}_t,1} \leqslant CBnf\sqrt{2T} \tag{5.31}$$

Proof. For the second result let $\mathbf{A}_t$ be generated by running OGD with step-size $\frac{C}{B\sqrt{2T}}$ on the losses $U_t(\mathbf{A}\mathbf{f}) = \|\mathbf{A}\mathbf{f} - \mathbf{x}^*(\mathbf{c}_t, \mathbf{b}_t)\|_{\mathbf{b}_t,1}$ over $[-C, C]^{n \times f}$. Since these are $B\sqrt{nf}$-Lipschitz and the duals are $C\sqrt{nf}$-bounded in the Euclidean norm, the regret follows from Shalev-Shwartz [2011, Corollary 2.7]. For the first result, apply online-to-batch conversion to the second result,

i.e. draw $T = \Omega\left(\left(\frac{CBn}{\varepsilon}\right)^2\left(f^2 + \log\frac{1}{\delta}\right)\right)$ samples $(\mathbf{c}_t, \mathbf{b}_t, \mathbf{f}_t)$, run OGD as above on the resulting losses U_t, and set $\hat{\mathbf{A}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{A}_t$ to be the average of the resulting predictions $\mathbf{A}_t$. Applying Lemma B.4.1 yields the result. □

5.C.2 Online page migration

Using instance features for online page migration is more involved because the output space must be constrained to the product of n $|\mathcal{K}|$-dimensional simplices. However, we can solve this by restricting to tensors consisting of matrices whose columns sum to one, also known as rectangular stochastic matrices. Note that the learning-theoretic results of Theorem 5.3.1 are special cases of the following when $\mathbf{f} = \mathbf{1}_1$ for all instances.

Theorem 5.C.2. In the setting of Theorem 5.3.1 let $\mathbb{S}^{n\times|\mathcal{K}|\times f}$ be the set of stacks of $|\mathcal{K}| \times f$ nonnegative matrices whose columns have unit ℓ_1-norm.

1. There exists a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and distribution $\mathcal{D}$ over request sequences s of length n in $\mathcal{K}$ and associated feature vectors $\mathbf{f} \in \triangle_f$ it takes

$$\mathcal{O}\left(\left(\frac{\gamma D}{\varepsilon}\right)^2\left(n^2 f^2 \log|\mathcal{K}| + \log\frac{1}{\delta}\right)\right) \tag{5.32}$$

   samples from $\mathcal{D}$ and returns $\hat{\mathbf{A}}$ s.t. w.p. $\geqslant 1-\delta$:

$$\mathbb{E}_{(s,\mathbf{f})\sim\mathcal{D}} U_s(\hat{\mathbf{A}}\mathbf{f}) \leqslant \min_{\mathbf{A}\in\mathbb{S}^{n\times|\mathcal{K}|\times f}} \mathbb{E}_{(s,\mathbf{f})\sim\mathcal{D}} U_s(\mathbf{A}\mathbf{f}) + \varepsilon \tag{5.33}$$

2. Let $(s_1, \mathbf{f}_1), \ldots, (s_T, \mathbf{f}_T)$ be an adversarial sequence of (request sequence, feature) pairs. Then updating the distribution $\mathbf{A}_{t[j,,k]}$ over $\triangle_{|\mathcal{K}|}$ at each (timestep,column) pair $(j,k) \in [n]\times[f]$ using EG with appropriate step-size has regret

$$\max_{\mathbf{A}\in\mathbb{S}^{n\times|\mathcal{K}|\times f}} \sum_{t=1}^{T} U_{s_t}(\mathbf{A}_t\mathbf{f}_t) - U_{s_t}(\mathbf{A}\mathbf{f}_t) \leqslant \gamma D n f\sqrt{2T\log|\mathcal{K}|} \tag{5.34}$$

Proof. For the second result let $\mathbf{A}_t$ be the sequence generated by running nf EG algorithms with step-size $\sqrt{\frac{\log|\mathcal{K}|}{2\gamma^2 D^2 T}}$ on the losses $U_{s_t}(\mathbf{A}\mathbf{f})$ over $\mathbb{S}^{n\times|\mathcal{K}|\times f}$. Since these losses are γD-Lipschitz and the maximum entropy over the simplex is $\log|\mathcal{K}|$, the regret guarantee follows from Shalev-Shwartz [2011, Theorem 2.15]. For the first result, apply standard online-to-batch conversion to the second result, i.e. draw $T = \Omega\left(\left(\frac{\gamma D}{\varepsilon}\right)^2\left(n^2 f^2\log|\mathcal{K}| + \log\frac{1}{\delta}\right)\right)$ samples $(\mathbf{s}_t, \mathbf{f}_t)$, run EG on the resulting losses $U_{s_t}(\mathbf{A}\mathbf{f}_t)$ as above, and set $\hat{\mathbf{A}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{A}_t$ to be the average of the resulting actions $\mathbf{A}_t$. Applying Lemma B.4.1 yields the result. □

We can further also show a result in the perhaps more-natural setting where the linear predictor $\mathbf{A}$ is the same for each element in the sequence, and maps directly from features to the $|\mathcal{K}|$-simplex. Notably, the linear auto-regressive setting, in which we want a linear map from the past k sequence elements to a probabilistic prediction of the next one, is covered by this result if we allow the features to be $k|\mathcal{K}|$-dimensional concatenations of k one-hot $|\mathcal{K}|$-length vectors.

Theorem 5.C.3. In the setting of Theorem 5.3.1 let $\mathbb{S}^{a\times b}$ be the set of $a \times b$ nonnegative matrices whose columns have unit ℓ_1-norm.

1. There exists a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and distribution $\mathcal{D}$ over request sequences s of length n in $\mathcal{K}$ and associated feature sequence $\mathbf{F}^\top \in \mathbb{S}^{f\times n}$ it takes $\mathcal{O}\left(\left(\frac{\gamma D}{\varepsilon}\right)^2 \left(n^2 f^2 \log|\mathcal{K}| + \log\frac{1}{\delta}\right)\right)$ samples from $\mathcal{D}$ and returns $\hat{\mathbf{A}}$ s.t. w.p. $\geqslant 1-\delta$:

$$\mathbb{E}_{(s,\mathbf{F})\sim\mathcal{D}} U_s(\mathbf{F}\hat{\mathbf{A}}^\top) \leqslant \min_{\mathbf{A}\in\mathbb{S}^{|\mathcal{K}|\times f}} \mathbb{E}_{(s,\mathbf{F})\sim\mathcal{D}} U_s(\mathbf{F}\mathbf{A}^\top) + \varepsilon \tag{5.35}$$

2. Let $(s_1, \mathbf{F}_1), \ldots, (s_T, \mathbf{F}_T)$ be an adversarial sequence of (request sequence, feature sequence) pairs. Then updating the distribution $\mathbf{A}_{t[,k]}$ over $\triangle_{|\mathcal{K}|}$ at each column $k \in [f]$ has regret

$$\max_{\mathbf{A}\in\mathbb{S}^{|\mathcal{K}|\times f}} \sum_{t=1}^{T} U_{s_t}(\mathbf{F}_t\mathbf{A}_t^\top) - U_{s_t}(\mathbf{F}_t\mathbf{A}^\top) \leqslant \gamma D f\sqrt{2T\log|\mathcal{K}|} \tag{5.36}$$

Proof. For the second result let $\mathbf{A}_t$ be the sequence generated by running f EG algorithms with step-size $\sqrt{\frac{\log|\mathcal{K}|}{2\gamma^2 D^2 T}}$ on the losses $U_{s_t}(\mathbf{F}\mathbf{A}^\top)$ over $\mathbb{S}^{|\mathcal{K}|\times f}$. Since these losses are γD-Lipschitz and the maximum entropy over the simplex is $\log|\mathcal{K}|$, the regret guarantee follows from [Shalev-Shwartz, 2011, Theorem 2.15]. For the first result, apply standard online-to-batch conversion to the second result, i.e. draw $T = \Omega\left(\left(\frac{\gamma D}{\varepsilon}\right)^2 \left(f^2\log|\mathcal{K}| + \log\frac{1}{\delta}\right)\right)$ samples $(\mathbf{s}_t, \mathbf{f}_t)$, run EG on the resulting losses $U_{s_t}(\mathbf{F}_t\mathbf{A}^\top)$ as above, and set $\hat{\mathbf{A}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{A}_t$ to be the average of the resulting actions $\mathbf{A}_t$. Applying Lemma B.4.1 yields the result. □

5.D Faster graph algorithms with predictions

In this section we compare to the results of Chen et al. [2022], who analyze several prediction-based graph algorithms, including one with an improved prediction-dependent runtime for the matching approach of Dinitz et al. [2021] and a prediction-dependent bound for single-source shortest path. From the learnability perspective, they observe two important error metrics in the analysis of graph algorithms with predictions: the ℓ_1-metric of Dinitz et al. [2021] measuring the ℓ_1-norm between the prediction and a ground truth vector such as the dual and the ℓ_∞-metric measuring the ℓ_∞-norm between the same quantities. In the first case their setting and results are equivalent to those of Dinitz et al. [2021], so we improve upon this by a factor of $\mathcal{O}(d)$, where d is the dimension of the hint.

To analyze the ℓ_∞ case, we start by showing that—as in the ℓ_1 case—we can round integer vectors with only a multiplicative factor loss:

Claim 5.D.1. Given any vectors $\mathbf{x} \in \mathbb{Z}^n$ and $\mathbf{y} \in \mathbb{R}^n$, let $\tilde{\mathbf{y}} \in \mathbb{Z}^n$ be the vector whose elements are those of $\mathbf{y}$ rounded to the nearest integer. Then we have $\|\mathbf{x}-\tilde{\mathbf{y}}\|_\infty \leqslant 2\|\mathbf{x}-\mathbf{y}\|_\infty$.

Proof. Let $S \subset [n]$ be the set of indices $i \in [n]$ for which $\mathbf{x}_{[i]} \geqslant \mathbf{y}_{[i]} \iff \tilde{\mathbf{y}}_{[i]} = \lceil \mathbf{y}_{[i]} \rceil$. For

$i \in [n] \backslash S$ we have $|\mathbf{x}_{[i]} - \mathbf{y}_{[i]}| \geqslant 1/2 \geqslant |\tilde{\mathbf{y}}_{[i]} - \mathbf{y}_{[i]}|$ so it follows by the triangle inequality that

$$\begin{aligned}
\|\mathbf{x} - \tilde{\mathbf{y}}\|_\infty &= \max\left\{\max_{i \in S} |\mathbf{x}_{[i]} - \tilde{\mathbf{y}}_{[i]}|, \max_{i \in [n] \backslash S} |\mathbf{x}_{[i]} - \tilde{\mathbf{y}}_{[i]}|\right\} \\
&\leqslant \max\left\{\max_{i \in S} |\mathbf{x}_{[i]} - \mathbf{y}_{[i]}|, \max_{i \in [n] \backslash S} |\mathbf{x}_{[i]} - \mathbf{y}_{[i]}| + |\mathbf{y}_{[i]} - \tilde{\mathbf{y}}_{[i]}|\right\} \\
&\leqslant \max\left\{\max_{i \in S} |\mathbf{x}_{[i]} - \mathbf{y}_{[i]}|, 2 \max_{i \in [n] \backslash S} |\mathbf{x}_{[i]} - \mathbf{y}_{[i]}|\right\} \leqslant 2\|\mathbf{x} - \mathbf{y}\|_\infty
\end{aligned} \tag{5.37}$$

□

We are thus able to also use online convex optimization in this setting and apply the rounded outputs to graph algorithms. In particular, we can use regular OGD to improve upon the ℓ_∞-learnability result of Chen et al. [2022] by a factor of $\mathcal{O}(d^2)$, where d is the dimension of the prediction:

Theorem 5.D.1. Consider any graph algorithm with optimal d-dimensional M-bounded predictions $\mathbf{h}(c)$ associated with every instance c.

1. There exists a poly-time algorithm s.t. for any $\delta, \varepsilon > 0$ and distribution $\mathcal{D}$ over instances it takes $\mathcal{O}\left(\left(\frac{M}{\varepsilon}\right)^2 \left(d + \log \frac{1}{\delta}\right)\right)$ samples from $\mathcal{D}$ and returns $\hat{\mathbf{h}} \in \mathbb{R}^d$ s.t. w.p. $\geqslant 1 - \delta$

$$\mathbb{E}_{c \sim \mathcal{D}} \|\hat{\mathbf{h}} - \mathbf{h}(c)\|_\infty \leqslant \min_{\|\mathbf{h}\|_\infty \leqslant M} \mathbb{E}_{c \sim \mathcal{D}} \|\mathbf{h} - \mathbf{h}(c)\|_\infty + \varepsilon \tag{5.38}$$

2. Let $c_1, \ldots, c_T$ be an adversarial sequence of instances. Then OGD with appropriate step-size achieves regret

$$\max_{\|\mathbf{h}\|_\infty \leqslant M} \sum_{t=1}^{T} \|\mathbf{h}_t - \mathbf{h}(c_t)\|_\infty - \|\mathbf{h} - \mathbf{h}(c_t)\|_\infty \leqslant M\sqrt{2dT} \tag{5.39}$$

Proof. The proof is the same as for the last two results of Theorem 5.4.1 in the special case $f = 1$. □

5.E Permutation predictions for non-clairvoyant scheduling

Finally, we discuss the the applicability of our framework to the results in Lindermayr and Megow [2022], who study how to prioritize among n jobs by predicting the best permutation of them under weights $\mathbf{w} \in \mathbb{R}^n_{\geqslant 0}$ and processing requirements $\mathbf{p} \in \mathbb{R}^n_{\geqslant 0}$ that are only known after completion. Ignoring robustness-consistency tradeoffs and terms that do not depend on the prediction, they show that in several settings the competitive ratio depends linearly on the following error of an $n \times n$ permutation matrix $\mathbf{X}$:

$$U_{\mathbf{w},\mathbf{p}}(\mathbf{X}) = \mathrm{Tr}(\mathbf{X}\mathbf{w}((\mathbf{U} \odot \mathbf{X})\mathbf{p})^\top) = \mathrm{Tr}((\mathbf{U} \odot \mathbf{X})^\top \mathbf{X}\mathbf{w}\mathbf{p}^\top) \tag{5.40}$$

where $\mathbf{U} \in \{0,1\}^{n \times n}$ is upper triangular. The above expression is derived from the third equation in the proof of Theorem 4.1 of Lindermayr and Megow [2022] for the case of $z = 1$ sample; we construct the matrix form to reason about its online learnability.

Naively, a sequence of bounded functions of permutations is *computationally inefficiently* learnable by using randomized EG over the $n!$ experts corresponding to each permutation:

Theorem 5.E.1. Consider the setting of Lindermayr and Megow [2022] with n jobs with W-bounded weights and P-bounded processing times. Let $\mathbb{P}^{n\times n}$ be the set of $n \times n$ permutation matrices.

1. There exists an algorithm that s.t. for any $\delta, \varepsilon > 0$ and distribution $\mathcal{D}$ over weights and processing requirements it takes $\mathcal{O}\left(\left(\frac{WPn}{\varepsilon}\right)^2 \left(n \log n + \log \frac{1}{\delta}\right)\right)$ samples from $\mathcal{D}$ and returns a discrete distribution $\hat{\mathbf{x}} \in \triangle_{n!}$ over $\mathbb{P}^{n\times n}$ such that

$$\mathbb{E}_{\mathbf{X}\sim\hat{\mathbf{x}}}\mathbb{E}_{(\mathbf{w},\mathbf{p})\sim\mathcal{D}} U_{\mathbf{w},\mathbf{p}}(\mathbf{X}) \leqslant \min_{\mathbf{X}\in\mathbb{P}^{n\times n}} \mathbb{E}_{(\mathbf{w},\mathbf{p})\sim\mathcal{D}} U_{\mathbf{w},\mathbf{p}}(\mathbf{X}) + \varepsilon \tag{5.41}$$

2. Let $(\mathbf{w}_1, \mathbf{p}_1), \ldots, (\mathbf{w}_T, \mathbf{p}_T)$ be an adversarial sequence of job (weight, processing requirement) pairs. Then running EG with appropriate step-size over $\triangle_{|\mathbb{P}^{n\times n}|}$ has regret

$$\mathbb{E} \max_{\mathbf{X}\in\mathbb{P}^{n\times n}} \sum_{t=1}^{T} U_{\mathbf{w}_t,\mathbf{p}_t}(\mathbf{X}_t) - U_{\mathbf{w}_t,\mathbf{p}_t}(\mathbf{X}) \leqslant WPn\sqrt{2nT\log n} \tag{5.42}$$

   where the expectation is over the randomness of the algorithm.

Proof. For the second result let $\mathbf{x}_t \in \triangle_{n!}$ be the sequence generated by running EG with step-size $\sqrt{\frac{\log(n!)}{2T}}$ over the $n!$ experts corresponding to each element of $\mathbb{P}^{n\times n}$. Then the sequence of permutation matrices $\mathbf{X}_t \sim \mathbf{x}_t$ sampled from this distribution satisfies the guarantee on the expected regret [Shalev-Shwartz, 2011, Corollary 2.14] since $\log(n!) \leqslant n \log n$ and $U_{\mathbf{w},\mathbf{p}}$ is WPn-bounded. The first result follows by applying online-to-batch conversion to this sequence, i.e. we draw $T = \Omega\left(\left(\frac{WPn}{\varepsilon}\right)^2 \left(n \log n + \log \frac{1}{\delta}\right)\right)$ samples $(\mathbf{w}_t, \mathbf{p}_t)$, run randomized EG as above, and set $\hat{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}_t$ to be the average of the resulting distributions $\mathbf{x}_t$. Applying Lemma B.4.1 yields the result. □

The sample complexity guarantee resulting from online-to-batch conversion matches that of Lindermayr and Megow [2022], except that the output is a distribution over permutation matrices so the error is in expectation over that distribution. However, randomized EG is incredibly inefficient due to the need to store and sample from a distribution over $n!$ variables. Another way of learning over permutation matrices is to run an online learning algorithm over the set of doubly stochastic matrices [Helmbold and Warmuth, 2009]. When the losses are linear functions of the permutation matrices this is yields efficient low-regret algorithms because each doubly stochastic matrix corresponds to a *small* convex combination of permutation matrices, i.e. a distribution from which one can sample an action. However, the losses $U_{\mathbf{w},\mathbf{p}}$ are nonlinear and so a different approach is needed.

Chapter 6

Private algorithms with private predictions

In the previous chapter we provided learning-theoretic guarantees for several existing learning-augmented algorithms. We now show how our framework can be deployed when extending algorithms with predictions in new directions, starting with differential privacy (DP). The differentially private release of statistics about a sensitive dataset $\mathbf{x} \in \mathbb{R}^n$ is an inevitably error-prone task because we are by definition precluded from revealing exact information about the instance at hand [Dwork and Roth, 2014]. However, DP instances rarely occur in a vacuum: even in the simplest practical settings, we usually know basic information such as the fact that all individuals have a nonnegative age. Often, the dataset we are considering is drawn from a similar population as a public dataset $\mathbf{x}' \in \mathbb{R}^N$ and should thus have similar statistics, a case known as the *public-private* setting [Liu et al., 2021a, Bie et al., 2022]. Alternatively, in what we call *sequential release*, we aim to release information about each of a sequence of datasets $\mathbf{x}_1, \ldots, \mathbf{x}_T$ one-by-one. These could be generated by a stationary or other process that allows information derived from prior releases to inform predictions of future releases. In all of these settings, we might hope to incorporate external information to reduce error, but approaches for doing so tend to be *ad hoc* and assumption-heavy.

We propose that the framework of algorithms with predictions [Mitzenmacher and Vassilvitskii, 2021]—provides the right tools for deriving DP algorithms in this setting, and instantiate this idea for multiple quantile release [Gillenwater et al., 2021, Kaplan et al., 2022], covariance estimation [Biswas et al., 2020, Amin et al., 2019, Dong et al., 2022], and data release [Hardt et al., 2012, Liu et al., 2021a]. Whereas in past algorithms with predictions work the goal is usually to bound the cost $C_{\mathbf{x}}(\mathbf{w})$ of running on instance $\mathbf{x}$ given a prediction $\mathbf{w}$ by some metric $U_{\mathbf{x}}(\mathbf{w})$ of the *quality* of the prediction on that instance, we instead aim to design learning-augmented algorithms where it captures the error of some statistic—e.g. quantiles—computed privately on instance an $\mathbf{x}$ given a prediction $\mathbf{w}$. We are interested in bounding this cost in terms of the quality of the external information provided to our algorithm, which we denote by $U_{\mathbf{x}}(\mathbf{w})$.

While incorporating external information into DP is well-studied, c.f. public-private methods [Bie et al., 2022, Liu et al., 2021a] and private posterior inference [Dimitrakakis et al., 2017, Geumlek et al., 2017, Seeman et al., 2020], by deriving and analyzing a learning-augmented algorithm for multiple quantiles we show numerous comparative advantages, including:

[0]The work presented in this chapter first appeared in Amin et al. [2023] and Khodak et al. [2023a].

1. Minimal data assumptions, sometimes even fewer than used by the unaugmented baseline.
2. Existing tools for studying the robustness of algorithms to noisy predictions [Lykouris and Vassilvitskii, 2021].
3. Co-design of algorithms with predictions with methods from Chapter 5 for *learning* those predictions from data, which we show is crucial for both the public-private and sequential release settings.

We derive learning-augmented extensions of the state-of-the-art ApproximateQuantiles (AQ) method [Kaplan et al., 2022] for quantile release and of the covariance estimation algorithms SeparateCov [Dong et al., 2022] and IterativeEigenvectorSampling [Amin et al., 2019]; for data release we show how our framework applies to MWEM [Hardt et al., 2012], for which using a non-uniform (i.e. prediction-based) prior has been studied in past work [Liu et al., 2021a]. In all cases our instance-dependent guarantees (nearly) match past worst-case bounds while being much better if a natural measure $U_{\mathbf{x}}(\mathbf{w})$ of prediction quality is small. We also show how these algorithms can be made robust to poor predictions $\mathbf{w}$ and how they can be efficiently and privately *learned* by optimizing $U_{\mathbf{x}}$ across related datasets $\mathbf{x}$. In addition, our analysis yields several contributions of independent interest for differential privacy:

1. The first robust algorithm for (single or multiple) private quantile release that avoids assuming the data is bounded on some interval, specifically by using a heavy-tailed prior.
2. Prediction-free trace-sensitive guarantees for SeparateCov (for both the pure and zCDP versions) that strictly improve upon the original bounds of Dong et al. [2022] for the same algorithm.
3. A non-Euclidean extension of DP-FTRL [Kairouz et al., 2021a] that is the first DP online convex optimization method that can be easily customized to obtain better regret guarantees on different geometries.

Finally, we conclude with an empirical study where we use our framework to design algorithms to reduce the error of private quantile release in both the public-private and sequential release settings described above. Our technical approach takes advantage of a novel connection between DP quantiles and censored regression to obtain both guarantees and practical algorithms. The experimental results highlight the effectiveness of our framework for ensuring robust performance in the face of noisy predictions and for designing surrogate loss functions that can be optimized to yield useful predictions.

6.1 Problem formulation

As discussed in Section 4.A, in algorithms with predictions we seek to bound some algorithmic performance measure $C_{\mathbf{x}}(\mathbf{w})$ by a prediction-dependent upper bound $U_{\mathbf{x}}(\mathbf{w})$ that measures the quality of a prediction $\mathbf{w}$ for the instance $\mathbf{x}$. In our work this cost will be the error of a privately released statistic, as compared to some ground truth. We will use the following privacy notion:

Definition 6.1.1 ([Dwork and Roth, 2014])**.** Algorithm $\mathcal{A}$ is (ε, δ)**-differentially private** if for all subsets S of its range, $\Pr(\mathcal{A}(\mathbf{x}) \in S) \leqslant e^{\varepsilon} \Pr(\mathcal{A}(\tilde{\mathbf{x}}) \in S) + \delta$ whenever $\mathbf{x} \sim \tilde{\mathbf{x}}$ are **neighboring datasets**.

Using ε-DP to denote $(\varepsilon, 0)$-DP, our broad goal will be to reduce the error $C_{\mathbf{x}}(\mathbf{w})$ of ε-DP multiple quantile release while fixing the privacy level ε. For easier comparison to past prediction-free results, we will define neighboring datasets differently depending on the application; specifically, for quantile release we use **add-remove privacy**, where $\mathbf{x}$ can be obtained from $\tilde{\mathbf{x}}$ by adding or removing an entry, while for covariance estimation and data release we use **swap privacy**, in which $\mathbf{x}$ can be obtained from $\tilde{\mathbf{x}}$ by replacing one entry with another.

Working with the learning-augmented algorithms framework when incorporating external information into DP methods allows us to make use of its existing language for quantifying useful properties such as robustness and learning. In past work robustness-consistency tradeoffs have mainly been studied for *online* algorithms with predictions, as for runtime we can easily show robustness by running the learning-augmented algorithm with a worst-case optimal algorithm in parallel. However, DP statistics are similar to online algorithms in that we have limited access to data, albeit in a very different manner, and so robustness to poor predictions is nontrivial to show.

In the previous chapter we argued that the prediction quality measures $U_{\mathbf{x}}(\mathbf{w})$ we derive should be useful for making good predictions, e.g. by $U_{\mathbf{x}_t}$ being *learnable* from multiple instance $\mathbf{x}_t$. We will again mainly focus on *online* learnability, i.e. bounds on the regret $\max_{\mathbf{w} \in W} \sum_{t=1}^{T} U_{\mathbf{x}_t}(\mathbf{w}_t) - U_{\mathbf{x}_t}(\mathbf{w})$ of predictions $\mathbf{w}_t$ in some space $\mathcal{W}$ given instances $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$. Since $U_{\mathbf{x}_t}$ roughly upper-bounds the error $C_{\mathbf{x}_t}$, this means that asymptotically the average error is governed by the average prediction quality $\min_{\mathbf{w} \in W} \frac{1}{T} \sum_{t=1}^{T} U_{\mathbf{x}_t}(\mathbf{w})$ of the optimal $\mathbf{w} \in W$. As in Chapter 5, we will seek to derive upper bounds $U_{\mathbf{X}}$ that are amenable to familiar gradient-based optimization schemes, which will also enable *instance-dependent* linear prediction: setting $\mathbf{w}_t$ using a learned function of some instance features $\mathbf{f}_t$. However, since the upper bounds depend on *sensitive* datasets $\mathbf{x}_t$, the learning algorithms we use will themselves have to be private, so in Section 6.5 we derive a non-Euclidean extension of DP-FTRL (c.f. Theorem 6.5.1) to show online and PAC learnability of the prediction quality measures $U_{\mathbf{x}}$ for all three DP tasks we consider.

The usefulness of both the learning-theoretic and robustness-consistency analysis is demonstrated in Section 6.6 on two applications where it is reasonable to have external information about the sensitive dataset(s). In the **public-private** setting, the prediction $\mathbf{w}$ is obtained from a public dataset $\mathbf{x}'$ that is assumed to be similar to $\mathbf{x}$ but is not subject to privacy-protection. In **sequential release**, we privately release information about each dataset in a sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$; the release at time t can depend on $\mathbf{x}_t$ and on a prediction $\mathbf{w}_t$, which can be derived (privately) from past observations. We show that sequential release can be posed directly as a private online learning problem, while the public-private setting can be approached via online-to-batch conversion [Cesa-Bianchi et al., 2004]. Both can thus be solved by treating the prediction quality measures $U_{\mathbf{x}_t}$ as surrogate objectives for the actual cost functions $C_{\mathbf{x}}$ and applying standard optimization techniques, as we demonstrated in Chapter 5.

6.2 Overview of theoretical results

We now summarize the main results for the three tasks we consider, focusing on the prediction-dependent performance bounds $U_{\mathbf{x}} \gtrsim C_{\mathbf{x}}$ that we show for our learning-augmented private algorithms. These will be stated more formally in Section 6.3. We also highlight the utility of these results in ensuring robustness and enabling learning, which will be further detailed in Sections 6.4 and 6.5, respectively.

6.2.1 Related work

There has been significant work on incorporating external information to improve DP methods. A major line of work is the public-private framework, where we have access to public data that is related in some way to the private data [Liu et al., 2021a, Amid et al., 2022, Li et al., 2022, Bie et al., 2022, Bassily et al., 2022]. The use of public data can be viewed as using a prediction, but such work starts by making (often strong) distributional assumptions on the public and private data; we instead derive instance-dependent upper bounds with minimal assumptions that we then apply to such public-private settings. Furthermore, our framework allows us to ensure robustness to poor predictions without distributional assumptions, and to derive learning algorithms using training data that may itself be sensitive. Another approach is to treat DP mechanisms (e.g. the exponential) as Bayesian posterior sampling [Dimitrakakis et al., 2017, Geumlek et al., 2017, Seeman et al., 2020]. Our work can be viewed as an adaptation where we give explicit prior-dependent utility bounds. To our knowledge, no such guarantees exist in the literature. Moreover, our approach does not necessitate specifying the external information in the form of (explicit) priors, e.g. for covariance estimation we use matrix predictions.

Our approach for augmenting DP with external information centers the algorithms with predictions framework, where past work has focused on using predictions to improve metrics related to time, space, and communication complexity. Tuning DP algorithms has been an important topic in private machine learning, e.g. for hyperparameter tuning [Chaudhuri and Vinterbo, 2013] and federated learning [Andrew et al., 2021], but these have not considered incorporating per-instance predictions.

6.2.2 Multiple quantile release

In the quantile problem, given a quantile q and a sorted dataset $\mathbf{x} \in \mathbb{R}^n$ of n distinct points, the goal is to release a number o that upper bounds exactly $\lfloor qn \rfloor$ of the entries. The error metric, $\mathrm{Gap}_q(\mathbf{x}, o)$, is the number of entries between the released number o and $\lfloor qn \rfloor$. A straightforward application of the well-known exponential mechanism [McSherry and Talwar, 2007] with utility $-\mathrm{Gap}_q$ outputs o that satisfies $\mathrm{Gap}_q(\mathbf{x}, o) \leqslant \frac{2}{\varepsilon} \log \frac{1}{\beta \Psi_{\mathbf{x}}^{(q)}}$ w.p. $\geqslant 1 - \beta$, where $\Psi_{\mathbf{x}}^{(q)}$ is the probability $\mu((\mathbf{x}_{[\lfloor qn \rfloor]}, \mathbf{x}_{[\lfloor qn \rfloor + 1]}])$ that the prior assigns to the optimal interval. We thus use $U_{\mathbf{x}}^{(q)}(\mu) = -\log \Psi_{\mathbf{x}}^{(q)}$ as our measure of prediction quality in the single-quantile setting, which allows us to recover standard guarantees that assume $\mathbf{x} \in (a, b)^n$ is bounded and set μ to be the uniform measure on (a, b). As our first major contribution, we show by studying $U_{\mathbf{x}}$ how to dispense with this assumption by instead using the Cauchy distribution with location $\frac{a+b}{2}$ and scale

$\frac{b-a}{2}$. If the boundedness assumption holds then the resulting mechanism has nearly the same bound on Gap_q as the uniform measure, up to an additive $\frac{2}{\varepsilon}\log\pi$ factor, but if does not—e.g. if all points $\mathbf{x}_{[i]}$ in the dataset are a distance $R > \frac{b-a}{2}$ away from $\frac{a+b}{2}$—then we still have the guarantee $\mathrm{Gap}_q = \tilde{\mathcal{O}}(\frac{\log R}{\varepsilon})$ w.h.p. (c.f. Corollary 6.3.1). In contrast, the error of the released quantile when using the uniform measure in the latter scenario is $\Omega(n)$ a.s.

The main technical challenge is then to extend the single-quantile guarantee to the case where we must estimate $m > 1$ quantiles $q_1, \ldots, q_m \in (0,1)$ while making use of m priors $\mu_1, \ldots, \mu_m$. In particular, we want a guarantee on the maximum gap that encodes how useful each prior μ_i is for its quantile q_i *and* that grows sublinearly in m, ideally recovering the $\max_i \mathrm{Gap}_{q_i} = \mathcal{O}(\frac{\mathrm{polylog}(m)}{\varepsilon})$ bound of Kaplan et al. [2022] in the prediction-free limit. Although it requires several major modifications to AQ, we are able to nearly achieve this goal, devising a method (c.f. Algorithm 13) that guarantees a bound of $\tilde{\mathcal{O}}(\frac{r(m)}{\varepsilon}\log\sum_{i=1}^m e^{U_{\mathbf{x}}^{(q_i)}(\mu_i)})$ on the maximum gap w.h.p. (c.f. Theorem 6.3.3), where $r(m)$ is sub-polynomial but super-polylogarithmic in m. This yields a quality measure $U_{\mathbf{x}}$ for $\mu_1, \ldots, \mu_m$ that aggregates the single-quantile measures $U_{\mathbf{x}}^{(q_i)}(\mu_i)$ via their log-sum-exp, a convenient form that allows us to easily extend single-quantile robustness and learning-theoretic results to multiple quantiles.

Our quantile results exemplify the advantages of our approach to incorporating external information into DP algorithms that we discussed in the introduction: minimal assumptions, robustness-consistency tradeoffs, and learning. In-fact, the first outcome of our analysis was *removing* a boundedness assumption. This contrasts with past public-private work [Liu et al., 2021a, Bie et al., 2022], which makes distributional assumptions, and is why we can obtain guarantees in two very distinct settings in Section 6.6. We next highlight how our results imply convenient robustness-consistency tradeoffs and efficient learnability.

Robustness

Using the formalization of robustness and consistency in Definitions 4.A.1 and 4.A.2, algorithms with predictions provides a convenient way to deploy them by *parameterizing* the robustness-consistency tradeoff, in which methods are designed to be $r_{\mathbf{x}}(\lambda)$-robust and $c_{\mathbf{x}}(\lambda)$-consistent for a user-specified parameter $\lambda \in [0,1]$ [Bamas et al., 2020, Lykouris and Vassilvitskii, 2021]. For quantiles, we can obtain an elegant parameterized tradeoff by interpolating prediction priors with a "robust" prior. In particular, we can pick ρ to be a trusted prior such as the uniform or Cauchy and for any prediction μ use $\mu^{(\lambda)} = (1-\lambda)\mu + \lambda\rho$ instead. Then since $\Psi_{\mathbf{x}}^{(q)}$ is linear we have $\Psi_{\mathbf{x}}^{(q)}(\mu^{(\lambda)}) = (1-\lambda)\Psi_{\mathbf{x}}^{(q)}(\mu) + \lambda\Psi_{\mathbf{x}}^{(q)}(\rho)$, which implies the following guarantee:

Corollary 6.2.1 (of Lem. 6.A.1; c.f. Cor. 6.4.1)**.** For any quantile $q \in (0,1)$, applying EM with prior $\mu^{(\lambda)} = (1-\lambda)\mu + \lambda\rho$ is $\left(\frac{2}{\varepsilon}\log\frac{1/\beta}{\lambda\Psi_{\mathbf{x}}^{(q)}(\rho)}\right)$-robust and $\left(\frac{2}{\varepsilon}\log\frac{1/\beta}{1-\lambda}\right)$-consistent.

Thus w.h.p. error is simultaneously at most $\frac{2}{\varepsilon}\log\frac{1}{\lambda}$ worse than that of only using the robust prior ρ and we only have error $\frac{2}{\varepsilon}\log\frac{1/\beta}{1-\lambda}$ if the prediction μ is perfect, i.e. if it is only supported on the optimal interval. This is easy to extend to the multiple-quantile metric $U_{\mathbf{x}} = -\log\Psi_{\mathbf{x}}$. In fact, we can even interpolate between the $\mathrm{polylog}(m)$ prediction-free guarantee of past work and our learning-augmented guarantee with the worse dependence on m (c.f. Corollary 6.4.2); thus

if the prediction is not good enough to overcome the worse rate we can still ensure that we do not do much worse than the original guarantee. These results show the advantage of our framework in designing algorithms that make robust use of possibly noisy predictions. Notably, related public-private work that studies robustness still assumes source and target data are Gaussian [Bie et al., 2022], whereas we make no distributional assumptions. We demonstrate the importance of our robustness techniques throughout the experiments in Section 6.6.

Learning

A last important use for prior-dependent bounds is as surrogate objectives for optimization. Being able to learn across upper bounds $U_{\mathbf{x}_1}, \ldots, U_{\mathbf{x}_T}$ of a sequence of (possibly sensitive) datasets $\mathbf{x}_t$ is useful for both the public-private setting and for the sequential release setting (c.f. Section 6.6). As we saw in Chapter 5, algorithms with predictions guarantees are often sufficiently nice to do this using off-the-shelf online learning, a property that largely holds for our upper bounds as well. Most saliently, the bound $U_{\mathbf{x}}^{(q)} = -\log \Psi_{\mathbf{x}}^{(q)}$ is a convex function of an inner product $\Psi_{\mathbf{x}}^{(q)}$ between the EM score and the prior μ; thus by discretizing one can learn over a large family of piecewise-constant priors, which themselves approximate Lipschitz priors over a bounded domain. The same is true of the multiple quantile bound $U_{\mathbf{x}}$ because it is the log-sum-exp over $U_{\mathbf{x}}^{(q_i)}$ and thus also convex. We therefore can apply an entropic variant of DP-FTRL to (privately) online learn the sequence $U_{\mathbf{x}_t}$ with low-regret w.r.t. any set of m Lipschitz priors (c.f. Theorem 6.5.2). However, in practice we may not want to learn in the high dimensions needed by the discretization, and rather than fixed priors we may wish to learn a mapping from dataset-specific features.

Thus, in Section 6.6 we focus on the less-expressive family of location-scale models, which allows us to develop algorithms that are amenable to both analysis and implementation. In particular, we show that $U_{\mathbf{x}}$ has the same form as the negative log-likelihood of censored regression, which for log-concave location-scale families is convex in a convenient reparameterization of the location and scale [Pratt, 1981, Burridge, 1981]. We can thus show DP online learning guarantees in the sequential release setting (c.f. Theorem 6.6.3) and derive an algorithm for public-private transfer whose error is bounded by the TV-distance between the order statistics of the public and private distributions (c.f. Theorem 6.6.2).

6.2.3 Covariance estimation

While encoding predictions via base measures of DP mechanisms is a natural starting point for learning-augmented algorithms, it is not the only way of doing so. We can instead start with existing algorithms whose errors have explicit or implicit dependence on some measure of complexity of the data and use this to convert them into algorithms with predictions. The errors will then have an (explicit) dependence on a related measure of the error between the data and a point (rather than distributional) prediction, leading to highly interpretable bounds $U_{\mathbf{x}}(\mathbf{w})$ on the utility loss.

Our application to covariance estimation exemplifies this approach. For this task we take advantage of recent "trace-sensitive" results, which bound the Frobenius error between the covariance matrix $\mathbf{C} = \mathbf{X}\mathbf{X}^\top/n$ of a dataset $\mathbf{X} \in \mathbb{R}^{d \times n}$ by some function of its trace [Amin et al.,

2019, Dong et al., 2022]. Since the core component of these algorithms is a DP estimate of a symmetric $d \times d$ matrix, if we have a symmetric prediction $\mathbf{W} \in \mathbb{R}^{d\times d}$ we can try to use the methods to instead privately estimate the error $\mathbf{C}-\mathbf{W}$ and then add $\mathbf{W}$ to the result; we can then hope to show that the error depends on the trace norm $\|\mathbf{C}-\mathbf{W}\|_{\mathrm{Tr}}$ of the error rather than the trace of $\mathbf{C}$. We achieve exactly this and more by extending the analysis in this prior work to the negative spectrum, in order to handle the possibly negative eigenvalues of $\mathbf{C}-\mathbf{W}$. The result below, for the learning-augmented extension of the state-of-the-art SeparateCov algorithm [Dong et al., 2022], is characteristic of these results (c.f. Section 6.3.2):

Corollary 6.2.2 (of Thm. 6.3.4; c.f. Cor 6.3.2)**.** If $\mathbf{X} \in \mathbb{R}^{d\times n}$ has columns bounded by 1 in ℓ_2-norm then applying SeparateCov to $\mathbf{C}-\mathbf{W}$ and obtaining $\hat{\mathbf{C}}$ by adding $\mathbf{W}$ to the result is ε-DP and satisfies $\|\hat{\mathbf{C}}-\mathbf{C}\|_F^2 \leqslant \tilde{\mathcal{O}}\left(\frac{d}{\varepsilon^2 n^2} + \frac{d\sqrt{d}}{\varepsilon n}\min_{c\in\mathbb{R}}\|\mathbf{C}-\mathbf{W}-c\mathbf{I}_d\|_{\mathrm{Tr}}\right)$ w.h.p.

Notably, for $\mathbf{W} = \mathbf{0}_{d\times d}$ this bound improves upon the corresponding prediction-free result of Dong et al. [2022], who only show it for $c = 0$. A simple setting where this improvement is tangible is when the columns of $\mathbf{X}$ are drawn from a bounded distribution whose covariance is a scalar multiple of the identity, in which case w.h.p. $\min_{c\in\mathbb{R}}\|\mathbf{X}\mathbf{X}^\top/n - c\mathbf{I}_d\|_{\mathrm{Tr}} \leqslant \tilde{\mathcal{O}}(d\min\{1,\sqrt{d/n}\})$ but $\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}} \geqslant \tilde{\mathcal{O}}(d)$; therefore for constant ε the bound in Corollary 6.3.2 becomes $\tilde{\mathcal{O}}(\frac{d^2}{n}\sqrt{\min\{d, d^2/n\}})$ whereas the bound of Dong et al. [2022, Lemma 19] is no better than $\tilde{\mathcal{O}}(d^2\sqrt{d}/n)$. In particular, for $d = O(1)$ our bound is asymptotically dominated by the error $\tilde{\mathcal{O}}(\frac{d^2}{n})$ of (non-privately) estimating the population covariance.

Robustness

Because of its nonconvexity, we drop the minimum over $c \in \mathbb{R}$ for our robustness and learning-theoretic analyses of covariance estimation, using the looser bound at $c = 0$ to define our prediction quality metric $U_{\mathbf{X}}(\mathbf{W}) = \|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}$. To ensure robustness, we take the approach of privately checking if the quality $U_{\mathbf{X}}(\mathbf{W})$ of the prediction $\mathbf{W} \in \mathbb{R}^{d\times d}$ is better than $U_{\mathbf{X}}(\mathbf{0}_{d\times d})$, i.e. that of the prediction-free approach. In doing so we pay for robustness by a factor of $\sqrt{d}$ in the leading (non-trace-sensitive) term, although as we discuss later this may be an artifact of the setting.

Corollary 6.2.3 (of Thm. 6.3.4; c.f. Cor. 6.3.2)**.** Running SeparateCov with the prediction $\mathbf{W}$ only if its trace distance $\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}$ is smaller than $\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}$ according to the Laplace mechanism is $\tilde{\mathcal{O}}\left(\frac{d\sqrt{d}}{\varepsilon n}\left(\frac{1}{\varepsilon n} + \|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}\right)\right)$-robust and $\tilde{\mathcal{O}}\left(\frac{d\sqrt{d}}{\varepsilon^2 n^2}\right)$-consistent.

Learning

Similar to before, we can pose the problem of learning to release covariance estimates across multiple datasets as the online learning problem of obtaining low regret w.r.t. any matrix $\mathbf{W} \in \mathbb{R}^{d\times d}$ for the functions $U_{\mathbf{X}_t}(\mathbf{W}) = \|\mathbf{X}_t\mathbf{X}_t^\top/n_t - \mathbf{W}\|_{\mathrm{Tr}}$ determined by the sequence of datasets $\{\mathbf{X}_t \in \mathbb{R}^{d\times n_t}\}_{t=1}^T$. We apply DP-FTRL with with a Schatten p-norm regularizer, which applies p-norm regularization to the spectrum of the matrix [Duchi et al., 2010]; this yields a $\mathcal{O}(\sqrt{d})$-improvement in the regret—and a corresponding $\mathcal{O}(d)$-improvement in sample complexity—over regular DP-FTRL, highlighting the usefulness of our non-Euclidean analysis.

Theorem 6.2.1 (c.f. Thm. 6.5.3)**.** There exists an (ε', δ')-DP online learner whose regret w.r.t. all symmetric $\mathbf{W} \in \mathbb{R}^{d\times d}$ is bounded w.h.p. by $\tilde{\mathcal{O}}\left(\sqrt{(1+d/\varepsilon')T}\right)$. Furthermore, if the datasets $\mathbf{X}_t$ are all drawn i.i.d. from the same distribution and we set $\hat{\mathbf{W}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{W}_t$ to be the average iterate then $T = \tilde{\Omega}\left(\frac{1+d/\varepsilon'}{\alpha^2}\right)$ samples suffice to ensure that w.h.p. its excess risk is at most α.

6.2.4 Data release

In our last application we study private data release, where we seek to construct a synthetic dataset $\hat{\mathbf{x}} \in \mathbb{R}^d_{\geqslant 0}$ using sensitive data $\mathbf{x} \in \mathbb{Z}^d_{\geqslant 0}$ such that the maximum error of a finite set $\mathcal{Q}$ of linear queries $\mathbf{q} \in [-1,1]^d$ is bounded. To do so we use the well-known MWEM method of Hardt et al. [2012], which has an implicit dependence on the KL-divergence $D_{\mathrm{KL}}(\mathbf{x}/n||\mathbf{1}_d/d)$ between the data distribution and the uniform distribution it uses to initialize its iterative approach; by instead initializing with a prediction $\mathbf{w} \in \triangle_d$ in the d-dimensional simplex one can instead obtain a dependence on $D_{\mathrm{KL}}(\mathbf{x}/n||\mathbf{w})$:

Lemma 6.2.1 (c.f. Lem. 6.3.2)**.** Initializing MWEM with $\mathbf{w} \in \triangle_d$ and running it for m iterations on dataset $\mathbf{x}$ is ε-DP and w.p. $\geqslant 1-\beta$ produces a synthetic dataset s.t. the largest mean squared error of any linear query in Q is bounded by $\mathcal{O}\left(\frac{n}{m}D_{\mathrm{KL}}(\frac{\mathbf{x}}{n}||\mathbf{w}) + \frac{m^2}{\varepsilon^2 n}\log^2\frac{m}{\beta}\log^4|Q|\right)$, where $n = \|\mathbf{x}\|_1$.

As in quantile release, for this task we can again ensure robustness via an interpolation-based approach, although here we are mixing finite-dimensional vectors rather than probability distributions. Note that using the uniform prior guarantees $\tilde{\mathcal{O}}\left(\sqrt[3]{\frac{n\log^2 d}{\varepsilon^2}}\right)$ error, so since the data-dimension d can be very large in this application, if we use small enough λ we can obtain a strong advantage under perfect predictions while ensuring performance similar to the prediction-free guarantee.

Corollary 6.2.4 (of Lem. 6.3.2; c.f. Cor. 6.4.4)**.** There exists a fixed number of iterations s.t. using $\mathbf{w}^{(\lambda)} = (1-\lambda)\mathbf{w} + \lambda\mathbf{1}_d/d$ instead of the prediction $\mathbf{w} \in \triangle_d$ to initialize MWEM is $\tilde{\mathcal{O}}\left(\sqrt[3]{\frac{n}{\varepsilon^2\log d}}\log\frac{d}{\lambda}\right)$-robust, and $\tilde{\mathcal{O}}\left(\lambda\sqrt[3]{\frac{n\log^2 d}{\varepsilon^2}}\right)$-consistent, where n is the number of records.

The observation that MWEM can be initialized non-uniformly is not novel, having been used by both the original authors and by subsequent public-private work [Liu et al., 2021a]. However, our learning-theoretic analysis reveals interesting aspects that this prior work does not consider as closely, such as how the optimal choice for *other* parameters of the algorithm are influenced by the prediction quality. In-particular, when online learning the sequence of prediction quality measures $U_{\mathbf{x}_t}(\mathbf{w}) \simeq \frac{n_t}{m}D_{\mathrm{KL}}(\mathbf{x}_t/n_t||\mathbf{w}) + \frac{m^2}{\varepsilon^2 n_t}$ that bound the error of data release—here n_t is the number of examples in $\mathbf{x}_t$ and m is the number of iterations—we note that the optimal setting of m depends on the similarity between instances: if $\min_{\mathbf{w}}\sum_{t=1}^{T} n_t D_{\mathrm{KL}}(\mathbf{x}_t/n_t||\mathbf{w})$, i.e. the entropy of the average distribution $\left(\sum_{t=1}^{T}\mathbf{x}_t\right)/\sum_{t=1}^{T}n_t$, is small then we can take advantage of this by taking fewer iterations. However, we do not know this entropy *a priori*, so we can instead adapt to it by competing with the best step-size—which will encode the unknown entropy—by

simultaneously running online learners both for $\mathbf{w}$ and for m, with the optimization domain of the latter being the m-simplex $\triangle_m$. We again apply entropic DP-FTRL to get the following guarantee:

Theorem 6.2.2 (c.f. Thm. 6.5.4)**.** There exists an (ε', δ')-DP algorithm that adaptively sets the initializations $\mathbf{w}_t \in \triangle_d$ and number of iterations $m_t > 0$ s.t. the regret w.r.t. the optimal (initialization, iteration) pair $(\mathbf{w}, m)$ is $\tilde{\mathcal{O}}\left(\frac{dN^{\frac{4}{3}}}{\lambda \min\{1,\varepsilon^2\}}\sqrt{T/\varepsilon'}\right)$, where $N = \max_t n_t$ is the maximum number of entries in any dataset $\mathbf{x}_t$.

6.2.5 Discussion

This concludes our overview of our theoretical results, where we highlight multiple ways of incorporating predictions—as priors in DP mechanisms, as offsets to be corrected using sensitive data, or as initializations for iterative methods—as well as two ways of making the methods robust to noisy predictions: (1) interpolating with a default prediction and (2) privately checking whether the quality of the default prediction is better. We also illustrate how learning-augmented analysis can yield new insights in the prediction-free setting, as demonstrated by our results for unbounded quantile release and trace-sensitive covariance estimation. Next we will go into further detail about these prediction-dependent guarantees, robustness-consistency tradeoffs, and learning-theoretic results in Sections 6.3, 6.4, and 6.5, respectively. Then in Section 6.6 we will present a theoretical and empirical investigation of of how to use predictions to improve multiple quantile release in both the public-private and sequential release settings.

6.3 Prediction-dependent utility bounds

As formulated in Section 6.1, the basic guarantee of learning-augmented private algorithm is an upper bound $U_{\mathbf{x}}(\mathbf{w})$ on the error $C_{\mathbf{x}}(\mathbf{w})$ of the statistic it releases about a dataset $\mathbf{x}$ when using a prediction $\mathbf{w}$. We now demonstrate how to design methods for different DP tasks that enjoy such guarantees. While for single quantile release and data release we take the straightforward approach of incorporating a prediction-dependent prior into the EM mechanism, we also show how to handle difficulties that arise when multiple mechanisms need to be combined for releasing multiple quantiles and how to incorporate matrix predictions instead of explicit distributional priors by estimating the additive error between true and predicted covariances. This section also discusses DP contributions of independent interest that arise from our study of measures of prediction quality, specifically our Cauchy-based approach for releasing quantiles without assuming boundedness (Corollary 6.3.1) and our improved bounds for the SeparateCov algorithm proposed by Dong et al. [2022] (Corollary 6.3.2).

6.3.1 Quantile estimation via prediction-dependent priors

Given a quantile $q \in (0, 1)$ and a sorted dataset $\mathbf{x} \in \mathbb{R}^n$ of n distinct points, we want to release $o \in [\mathbf{x}_{[\lfloor qn \rfloor]}, \mathbf{x}_{[\lfloor qn \rfloor + 1]})$, i.e. such that the proportion of entries less than o is q. As in prior work [Kaplan

et al., 2022], the error of o will be the number of points between it and the desired interval:

$$\mathrm{Gap}_q(\mathbf{x}, o) = |\{i : \mathbf{x}_{[i]} < o\}| - \lfloor qn \rfloor| = |\max_{\mathbf{x}_{[i]}<o} i - \lfloor qn \rfloor| \tag{6.1}$$

$\mathrm{Gap}_q(\mathbf{x}, o)$ is constant on intervals $I_k = (\mathbf{x}_{[k]}, \mathbf{x}_{[k+1]}]$ in the partition by $\mathbf{x}$ of $\mathbb{R}$ (let $I_0 = (-\infty, \mathbf{x}_{[1]}]$ and $I_n = (\mathbf{x}_{[n]}, \infty)$), so we also say that $\mathrm{Gap}_q(\mathbf{x}, I_k)$ is the same as $\mathrm{Gap}_q(\mathbf{x}, o)$ for some o in the interior of I_k.

Warm-up: Releasing one quantile

For single quantile release we choose perhaps the most natural way of specifying a prediction for a DP algorithm: via the base measure $\mu : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$ of the exponential mechanism:

Theorem 6.3.1 (McSherry and Talwar [2007])**.** If the **utility** $u(\mathbf{x}, o)$ of an outcome o of a query over dataset $\mathbf{x}$ has **sensitivity** $\max_{o,\mathbf{x}\sim\tilde{\mathbf{x}}} |u(\mathbf{x}, o) - u(\tilde{\mathbf{x}}, o)| \leqslant \Delta$ then the **exponential mechanism**, which releases o w.p. $\propto \exp(\frac{\varepsilon}{2\Delta} u(\mathbf{x}, o))\mu(o)$ for some base measure μ, is ε-DP.

The utility function we use is $u_q = -\mathrm{Gap}_q$, so since this is constant on each interval I_k the mechanism here is equivalent to sampling k w.p. $\propto \exp(\varepsilon u_q(\mathbf{x}, I_k)/2)\mu(I_k)$ and then sampling o from I_k w.p. $\propto \mu(o)$. While the idea of specifying a prior for EM is well-known, the key idea here is to obtain a prediction-dependent bound on the error that reveals a useful measure of the *quality* of the prediction. In particular, we can show (c.f. Lemma 6.A.1) that running EM in this way yields o that w.p. $\geqslant 1 - \beta$ satisfies

$$\mathrm{Gap}_q(\mathbf{x}, o) \leqslant \frac{2}{\varepsilon} \log \frac{1/\beta}{\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu)} \leqslant \frac{2}{\varepsilon} \log \frac{1/\beta}{\Psi_{\mathbf{x}}^{(q)}(\mu)} \tag{6.2}$$

where the quantity $\Psi_{\mathbf{x}}^{(q,\varepsilon)} = \int \exp(-\frac{\varepsilon}{2} \mathrm{Gap}_q(\mathbf{x}, o))\mu(o)do$ is the inner product between the prior and the EM score while $\Psi_{\mathbf{x}}^{(q)} = \lim_{\varepsilon\to\infty} \Psi_{\mathbf{x}}^{(q,\varepsilon)} = \mu((\mathbf{x}_{[\lfloor qn \rfloor]}, \mathbf{x}_{[\lfloor qn \rfloor+1]}])$ is the probability that the prior assigns to the optimal interval.

This suggests two metrics of prediction quality: the negative log-inner-products $U_{\mathbf{x}}^{(q,\varepsilon)}(\mu) = -\log \Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu)$ and $U_{\mathbf{x}}^{(q)}(\mu) = -\log \Psi_{\mathbf{x}}^{(q)}(\mu)$. Both make intuitive sense: we expect predictions μ that assign a high probability to intervals that the EM score weighs heavily to perform well, and EM assigns the most weight to the optimal interval. There are also many ways that these metrics are useful. For one, in the case of perfect prediction—i.e. if μ assigns probability one to the optimal interval $I_{\lfloor qn \rfloor}$—then $\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu) = \Psi_{\mathbf{x}}^{(q)}(\mu) = 1$, yielding an upper bound on the error of only $\frac{2}{\varepsilon} \log \frac{1}{\beta}$. Secondly, as we will see, both are also amenable for analyzing robustness (the mechanism's sensitivity to *incorrect* priors) and learning. A final and important quality is that the guarantees using these metrics hold under no extra assumptions. Between the two, the first metric provides a tighter bound on the utility loss while the second does not depend on ε, which may be desirable.

It is also fruitful to analyze the metrics for specific priors. When $\mathbf{x}$ is in a bounded interval (a, b) and $\mu(o) = \frac{1_{o\in(a,b)}}{b-a}$ is the uniform measure, then $\Psi_{\mathbf{x}}^{(q)}(\mu) \geqslant \frac{\psi_{\mathbf{x}}}{b-a}$, where $\psi_{\mathbf{x}}$ is the minimum distance between entries; thus we recover past bounds, e.g. Kaplan et al. [2022, Lemma A.1], that implicitly use this measure to guarantee $\mathrm{Gap}_q(\mathbf{x}, o) \leqslant \frac{2}{\varepsilon} \log \frac{b-a}{\beta\psi_{\mathbf{x}}}$. Here the support of the

uniform distribution is correct by assumption as the data is assumed bounded. However, analyzing $\Psi_{\mathbf{x}}^{(q)}$ also yields a novel way of removing this assumption: if we suspect the data lies in (a, b), we set μ to be the Cauchy prior with location $\frac{a+b}{2}$ and scale $\frac{b-a}{2}$. Even if we are wrong about the interval, there exists an $R > 0$ s.t. the data lies in the interval $(\frac{a+b}{2} \pm R)$, so using the Cauchy yields $\Psi_{\mathbf{x}}^{(q)} \geqslant \frac{2(b-a)\psi_{\mathbf{x}}/\pi}{(b-a)^2+4R^2}$ and thus the following guarantee:

Corollary 6.3.1 (of Lem. 6.A.1)**.** If the data lies in the interval $(\frac{a+b}{2} \pm R)$ and μ is the Cauchy measure with location $\frac{a+b}{2}$ and scale $\frac{b-a}{2}$ then the output of the exponential mechanism satisfies $\mathrm{Gap}_q(\mathbf{x}, o) \leqslant \frac{2}{\varepsilon} \log\left(\pi \frac{b-a+\frac{4R^2}{b-a}}{2\beta\psi_{\mathbf{x}}}\right)$ w.p. $\geqslant 1 - \beta$.

If $R = \frac{b-a}{2}$, i.e. we get the interval right, then the bound is only an additive factor $\frac{2}{\varepsilon}\log\pi$ worse than before, but if we are wrong then performance degrades as $\mathcal{O}(\log(1 + R^2))$, unlike the $\mathcal{O}(R)$ error of the uniform prior. Note our use of a heavy-tailed distribution here: a sub-exponential density decays too quickly and leads to error $\mathcal{O}(R)$ rather than $\mathcal{O}(\log(1 + R^2))$. We can also adapt this technique if we know only a single-sided bound, e.g. if values must be positive, by using an appropriate half-Cauchy distribution.

Releasing multiple quantiles

To simultaneously estimate quantiles $q_1, \ldots, q_m$ we adapt the ApproximateQuantiles [Kaplan et al., 2022], which assigns each q_i to a node in a binary tree and, starting from the root, uses EM with the uniform prior to estimate a quantile before sending the data below the outcome o to its left child and the data above o to its right child. Thus each entry is only involved in $\lceil \log_2 m \rceil$ exponential mechanisms, and so for data in (a, b) the maximum Gap_{q_i} across quantiles is $\mathcal{O}\left(\frac{\log^2 m}{\varepsilon} \log \frac{m(b-a)}{\beta\psi_{\mathbf{x}}}\right)$, which is much better than the naive bound of a linear function of m.

Given one prior μ_i for each q_i, a naive extension of (6.2) gets a similar $\mathrm{polylog}(m)$ bound (c.f. Lem 6.A.2); notably we extend the Cauchy-unboundedness result to multiple quantiles (c.f. Corollary 6.A.1). However the upper bound is not a deterministic function of μ_i, as it depends on restrictions of $\mathbf{x}$ and μ_i to subsets (o_j, o_k) of the domain induced by the outcomes of EM for quantiles q_j and q_k earlier in the tree. It thus does not encode a direct relationship between the prediction and instance data and is less amenable for learning.

We instead want guarantees depending on a more natural metric, e.g. one aggregating $\Psi_{\mathbf{x}}^{(q_i,\varepsilon_i)}(\mu_i)$ from the previous section across pairs (q_i, μ_i). The core issue is that the data splitting makes the probability assigned by a prior μ_i to data outside the interval (o_j, o_k) induced by the outcomes of quantiles q_j and q_k earlier in the tree not affect the distribution of o_i. One way to handle this is to assign this probability mass to the edges of (o_j, o_k), rather than the more natural conditional approach of ApproximateQuantiles. We refer to this as "edge-based prior adaptation" and use it to bound $\mathrm{Gap}_{\max} = \max_i \mathrm{Gap}_{q_i}(\mathbf{x}, o_i)$ via the harmonic mean $\Psi_{\mathbf{x}}^{(\varepsilon)}$ of the inner products $\Psi_{\mathbf{x}}^{(q_i,\varepsilon_i)}(\mu_i)$:

Theorem 6.3.2 (c.f. Thm. 6.A.1)**.** If $m = 2^k - 1$ for some k, quantiles $q_1, \ldots, q_m$ are uniformly spaced, and for each we have a prior $\mu_i : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$, then running ApproximateQuantiles with edge-based prior adaptation (c.f. Algorithm 13) is ε-DP, and w.p. $\geqslant 1 - \beta$

$$\mathrm{Gap}_{\max} \leqslant \frac{2}{\varepsilon}\phi^{\log_2(m+1)}\lceil\log_2(m+1)\rceil \log\frac{m/\beta}{\Psi_{\mathbf{x}}^{(\varepsilon)}} \qquad \text{for} \quad \Psi_{\mathbf{x}}^{(\varepsilon)} = \left(\sum_{i=1}^{m} \frac{1/m}{\Psi_{\mathbf{x}}^{(q_i,\varepsilon_i)}(\mu_i)}\right)^{-1} \tag{6.3}$$

Here $\varepsilon_i = \frac{\varepsilon}{\lceil\log_2(m+1)\rceil}$ and $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

The golden ratio is due to a Fibonacci-type recurrence bounding the maximum Gap_{q_i} at each depth of the tree. $\Psi_{\mathbf{x}}^{(\varepsilon)}$ depends only on $\mathbf{x}$ and predictions μ_i, and it yields a nice error metric $U_{\mathbf{x}}^{(\varepsilon)} = -\log\Psi_{\mathbf{x}}^{(\varepsilon)} = \log\sum_{i=1}^{m} e^{U_{\mathbf{x}}^{(q_i,\varepsilon_i)}}$. However, the dependence of the error on m is worse than that of ApproximateQuantiles, as $\phi^{\log_2 m}$ is roughly $\mathcal{O}(m^{0.7})$, although the bound is still sublinear and thus better than the naive baseline of running EM m times. Note that, as in the single-quantile case, we can construct a looser but ε-independent upper bound

$$U_{\mathbf{x}} = -\log\Psi_{\mathbf{x}} = \log\sum_{i=1}^{m} e^{U_{\mathbf{x}}^{(q_i)}} \geqslant U_{\mathbf{x}}^{(\varepsilon)} \tag{6.4}$$

using the harmonic mean $\Psi_{\mathbf{x}}$ of $\Psi_{\mathbf{x}}^{(q_i)}$. We will make heavy use of this prediction quality measure as a surrogate loss function in applications (c.f. Section 6.6).

The $\tilde{\mathcal{O}}(\phi^{\log_2 m})$ dependence on the number of quantiles m in Theorem 6.3.2 results from error compounding across depths of the tree, so we can try to reduce depth by going from a binary to a K-ary tree. This involves running EM $K-1$ times at each node—and paying $K-1$ more in budget—to split the data into K subsets; the resulting estimates may also be out of order. However, by showing that sorting them back into order does not increase the error and then controlling the maximum Gap_{q_i} at each depth via another recurrence relation, we prove the following:

Theorem 6.3.3 (c.f. Thm. 6.A.2)**.** For any $q_1, \ldots, q_m$, using $K = \lceil\exp(\sqrt{\log 2\log(m+1)})\rceil$ and edge-based adaptation guarantees ε-DP and w.p. $\geqslant 1-\beta$ has

$$\mathrm{Gap}_{\max} \leqslant \frac{2\pi^2}{\varepsilon}\exp\left(2\sqrt{\log(2)\log(m+1)}\right)\log\frac{m/\beta}{\Psi_{\mathbf{x}}^{(\varepsilon)}} \tag{6.5}$$

The rate in m is both sub-polynomial and super-poly-logarithmic ($o(m^\alpha)$ and $\omega(\log^\alpha m)\ \forall\,\alpha > 0$); while asymptotically worse than the prediction-free original result [Kaplan et al., 2022], for almost any practical value of m (e.g. $m \in [3, 10^{12}]$) it does not exceed a small constant (e.g. nine) times $\log^3 m$. Thus if the error $-\log\Psi_{\mathbf{x}}^{(\varepsilon)}$ of the prediction is small—i.e. the inner products between priors and EM scores are large on (harmonic) average—then we may do much better with this approach.

We compare K-ary AQ with edge-based adaptation to regular AQ in Figure 6.1. The original is better at higher ε but similar or worse at higher privacy. We also find that conditional adaptation is only better on discretized data with repetitions, where neither method provides guarantees. Overall, we find that our prior-dependent analysis covers a useful algorithm, but for consistency with past work and due to its better performance at high ε we focus on the original binary approach in experiments.

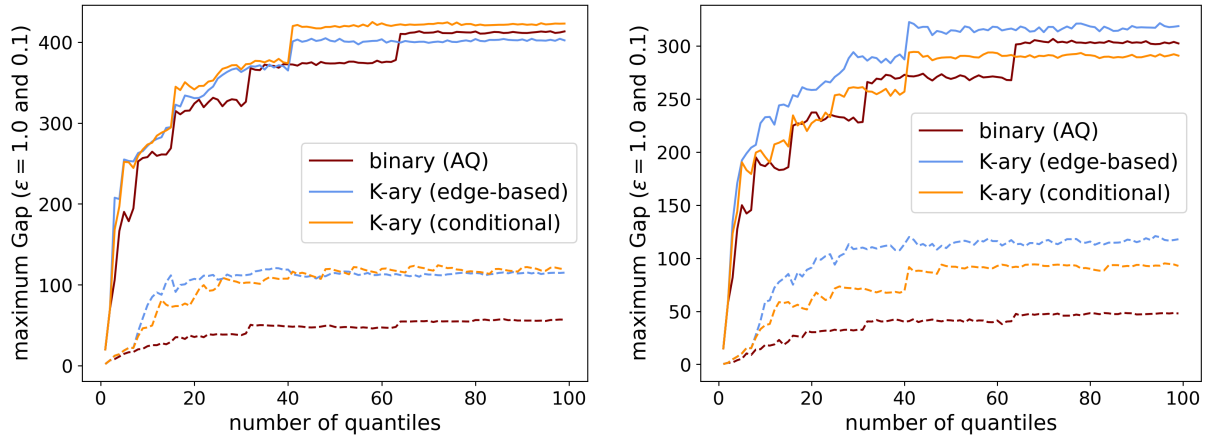

Figure 6.1: Maximum gap as a function of m for different variants of AQ when using the uniform prior, evaluated on 1000 samples from a standard Gaussian (left) and the Adult "age" dataset (right). The dashed and solid lines correspond to $\varepsilon = 1$ and 0.1, respectively.

6.3.2 Covariance estimation by estimating the prediction error

Encoding predictions as priors for EM and other mechanisms is a natural starting point for integrating external information into DP algorithms, but one might also wish to use a point prediction directly and hope to perform well if some distance measure between it and the output is small. While this is a less natural requirement for quantile release, where errors are measured using data points rather than metrics over the domain they live in, we show how this is easily achievable for the important problem of covariance estimation. In this setting we have a dataset $\mathbf{X} \in \mathbb{R}^{d \times n}$, where each of n records is a d-dimensional column with ℓ_2-norm bounded by 1, and we want to privately release an approximation $\hat{\mathbf{C}}$ of its covariance matrix $\mathbf{C} = \mathbf{X}\mathbf{X}^\top/n$ such that the Frobenius distance between the two is small.

Given a prediction $\mathbf{W} \in \mathbb{R}^{d \times d}$ of $\mathbf{C}$, one can immediately construct the trivial, private, prediction-sensitive algorithm of just releasing $\mathbf{W}$, which has the obvious prediction-dependent performance guarantee of $\|\mathbf{W} - \mathbf{C}\|_F$. However, we can hope to use the data to get an error that both decreases with n and is small if some distance between the prediction and ground truth is small. To do so, we make use of recent approaches that enjoy *trace-sensitive* guarantees, i.e. their utility improves if $\mathrm{Tr}(\mathbf{X}\mathbf{X}^\top)$ is small [Amin et al., 2019, Dong et al., 2022]; for example, the state-of-the-art method SeparateCov returns $\hat{\mathbf{C}}$ that is ε-DP and satisfies $\|\hat{\mathbf{C}} - \mathbf{C}\|_F^2 = \tilde{\mathcal{O}}(\frac{d}{\varepsilon^2 n^2} + \frac{d\sqrt{d}}{\varepsilon n}\mathrm{Tr}(\mathbf{X}\mathbf{X}^\top/n))$ w.h.p. [Dong et al., 2022, Lemma 18]. This suggests a natural way to incorporate a symmetric prediction matrix $\mathbf{W}$: use the existing algorithm to privately estimate its difference $\mathbf{C} - \mathbf{W}$ with the ground truth, and then add $\mathbf{W}$ to the result; since $\mathbf{C} - \mathbf{W}$ is no longer PSD, the hope would be to obtain error that scales with its trace norm.

We do exactly this in Algorithm 10, which uses the SeparateCov approach of separately estimating and combining eigenvalues and eigenvectors but applies it to $\mathbf{C} - \mathbf{W}$. The one potential issue is showing that their main error bound holds for symmetric matrices with negative eigenvalues, but this follows in Lemma 6.3.1 by applying their argument to both sides of the spectrum (c.f. Appendix 6.B.1):

Algorithm 10: SeparateCov with predictions

Input: data $\mathbf{X} \in \mathbb{R}^{d\times n}$, symmetric prediction matrix $\mathbf{W} \in \mathbb{R}^{d\times d}$, privacy $\varepsilon > 0$

$\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \leftarrow \mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$

$\hat{\mathbf{\Lambda}} \leftarrow \mathbf{\Lambda} + \operatorname{diag}(\mathbf{z})$ where $\mathbf{z}_{[i]} \sim \operatorname{Lap}\left(\frac{4}{\varepsilon n}\right)$ `// add noise to error eigenvalues`

$\tilde{\mathbf{C}} \leftarrow \mathbf{X}\mathbf{X}^\top/n + \mathbf{Z}$ for $\mathbf{Z}_{[i,j]} = \mathbf{Z}_{[j,i]} \sim \operatorname{Lap}\left(\frac{2d\sqrt{2}}{\varepsilon n}\right)$

$\tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top \leftarrow \tilde{\mathbf{C}} - \mathbf{W}$ `// get eigenvectors of noised prediction error`

Output: $\hat{\mathbf{C}} = \tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top + \mathbf{W}$ `// combine to estimate` $\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$ `and add` $\mathbf{W}$

Lemma 6.3.1. For $\mathbf{X} \in \mathbb{R}^{d\times n}$ and symmetric $\mathbf{W} \in \mathbb{R}^{d\times d}$, if $\tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top = \mathbf{X}\mathbf{X}^\top/n - \mathbf{W} + \mathbf{Z}$ for some symmetric $\mathbf{Z} \in \mathbb{R}^{d\times d}$ and $\hat{\mathbf{\Lambda}} = \mathbf{\Lambda} + \operatorname{diag}(\mathbf{z})$ for $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$ and some vector $\mathbf{z} \in \mathbb{R}^d$ then

$$\|\tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top + \mathbf{W} - \mathbf{X}\mathbf{X}^\top/n\|_F^2 \leqslant 4\left(\|\mathbf{z}\|_2^2 + \|\mathbf{Z}\|_\infty \|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}\right) \tag{6.6}$$

We can then apply Laplace concentration to obtain the performance-dependent guarantee in Theorem 6.3.4, which recovers the guarantee of Dong et al. [2022, Lemma 18] when $\mathbf{W} = \mathbf{0}_{d\times d}$.[1] The result shows that if we have a good guess of the prediction matrix in terms of trace distance then the error can be made to depend mostly on the first term—which has a better dependence on both d and n—without sacrificing privacy. Note that the algorithm requires the same number of eigen-decompositions as the one without predictions [Dong et al., 2022] and only requires some extra matrix additions to implement.

Theorem 6.3.4. If $\mathbf{X}$ has columns bounded by 1 in ℓ_2-norm then Algorithm 10 is ε-DP and w.p. $\geqslant 1 - \beta$

$$\|\hat{\mathbf{C}} - \mathbf{X}\mathbf{X}^\top/n\|_F^2 \leqslant \frac{144d + \mathcal{O}(\log^2 \frac{1}{\beta} \log^2 d)}{\varepsilon^2 n^2} + \frac{48d\sqrt{2d} + \mathcal{O}(d \log \frac{1}{\beta} \log d)}{\varepsilon n} \|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}} \tag{6.7}$$

Proof. Following the analysis in Amin et al. [2019, Theorem 1] (c.f. Lemma 6.B.1) the ℓ_1-sensitivity of the eigenvalues of $\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$ is $2/n$, and upper-bounding the ℓ_2-sensitivity of the covariance $\mathbf{X}\mathbf{X}^\top/n$ of $\sqrt{2}/n$ [Biswas et al., 2020, Lemma 3.2] shows that its ℓ_1-sensitivity is $d\sqrt{2}/n$. Thus the privacy guarantee follows from the composition of two Laplace mechanisms with budget $\varepsilon/2$ each. For the utility guarantee we use concentration of $\|\mathbf{y}\|_2 \leqslant 3\sqrt{d}/2 + \mathcal{O}\left(\log \frac{1}{\beta} \log d\right)$ w.p. $\geqslant 1 - \beta/2$ for i.i.d. $\mathbf{y}_{[i]} \sim \operatorname{Lap}(1)$ [Dong et al., 2022, Lemma 15] and $\|\mathbf{Y}\|_\infty \leqslant 3\sqrt{d} + \mathcal{O}\left(\log \frac{1}{\beta} \log d\right)$ w.p. $\geqslant 1 - \beta/2$ for i.i.d. $\mathbf{Y}_{[i,j]} \sim \operatorname{Lap}(1)$ for $i \geqslant j$ and $\mathbf{Y}_{[i,j]} = \mathbf{Y}_{[j,i]}$ for $i < j$ [Dong et al., 2022, Lemma 16]. Substituting $\mathbf{z} = \frac{4}{\varepsilon n}\mathbf{y}$ and $\mathbf{Z} = \frac{2d\sqrt{2}}{\varepsilon n}\mathbf{Y}$ into Lemma 6.3.1 yields the result. □

In addition to its computational simplicity, there are two other aspects of Algorithm 10 that are important for understanding the utility of its output: (1) it adds the same amount of noise

[1] Unlike Dong et al. [2022] we square the Frobenius norm for the purposes of learning predictions later; in the single-instance setting this is immaterial. Whether one is more interested in one or the other is application-dependent.

Algorithm 11: MWEM with predictions

Input: dataset $\mathbf{x} \in \mathbb{Z}_{\geqslant 0}^d$ with n entries, query set $Q \subset [-1,1]^d$, prediction $\mathbf{w} \in \triangle_d$, number of iterations $m > 0$, privacy parameter $\varepsilon > 0$

$\mathbf{w}_1 \leftarrow \mathbf{w}$

for $i = 1, \ldots, m$ **do**

sample $\mathbf{q}_i \in Q$ w.p. $\propto \exp\left(\frac{\varepsilon}{8m}\left|\left\langle \mathbf{q}, \mathbf{x} - \frac{n\mathbf{w}_i}{\|\mathbf{w}_i\|_1}\right\rangle\right|\right)$ `// exponential mechanism`

$\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i \odot \exp\left(\frac{\left\langle \mathbf{q}_i, \mathbf{x} - \frac{n\mathbf{w}_i}{\|\mathbf{w}_i\|_1}\right\rangle + \mathrm{Lap}\left(\frac{4m}{\varepsilon}\right)}{2n}\mathbf{q}_i\right)$ `// mult. weights update`

Output: $\hat{\mathbf{x}} = \frac{n}{m}\sum_{i=1}^m \mathbf{w}_i$ `// release average iterate`

as the original SeparateCov method [Dong et al., 2022, Algorithm 1], despite our two-sided sensitivity analysis, and (2) it is invariant to perturbations of the prediction matrix by any scalar multiple of the identity, i.e. $\hat{\mathbf{C}}$ is the same when $\mathbf{W}$ is replaced by $\mathbf{W} + c\mathbf{I}_d$ for any $c \in \mathbb{R}$. Crucially, this means we can obtain a tighter bound for free by replacing the trace difference in the upper bound (6.7) by $\min_{c\in\mathbb{R}} \|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W} + c\mathbf{I}_d\|_{\mathrm{Tr}}$. Substituting $\mathbf{W} = \mathbf{0}_{d\times d}$ then yields the following corollary, which is a strict improvement upon the main pure-DP guarantee of Dong et al. [2022, Lemma 19] for prediction-free SeparateCov:

Corollary 6.3.2. If $\mathbf{X}$ has columns bounded by 1 in ℓ_2-norm then Algorithm 10 with $\mathbf{W} = \mathbf{0}_{d\times d}$ returns w.p. $\geqslant 1-\beta$ an estimate $\hat{\mathbf{C}} \in \mathbb{R}^{d\times d}$ satisfying

$$\|\hat{\mathbf{C}} - \mathbf{X}\mathbf{X}^\top/n\|_F^2 \leqslant \frac{144d + \mathcal{O}(\log^2\frac{1}{\beta}\log^2 d)}{\varepsilon^2 n^2} + \frac{48d\sqrt{2d} + \mathcal{O}(d\log\frac{1}{\beta}\log d)}{\varepsilon n}\min_{c\in\mathbb{R}}\|\mathbf{X}\mathbf{X}^\top/n - c\mathbf{I}_d\|_{\mathrm{Tr}} \tag{6.8}$$

While this improvement is for a prediction-free method, it is the direct result of the two-sided analysis we needed to incorporate predictions; as with our unbounded quantile release result, this is another example of how learning-augmented analysis is useful even in the prediction-free setting.

Lastly, we point the interested reader to several supplementary results that highlight the broad applicability of our framework. First, while we focus on pure DP (except for learning), the main analysis of Dong et al. [2022] is in the zCDP setting; in Appendix 6.B.2 we show that similar guarantees hold there. Note that a prediction-free improvement similar to that of Corollary 6.3.2 can also be shown for SeparateCov under zCDP (c.f. Corollary 6.B.1, which improves upon Dong et al. [2022, Theorem 1]). Lastly, we show that prediction-dependent guarantee also holds for the older approach of Amin et al. [2019], albeit with a modified algorithm and a more involved sensitivity analysis (c.f. Appendix 6.B.3).

6.3.3 Initializing synthetic dataset construction with a predicted dataset

Our final application is to private data release, in which the goal is to privately respond to queries of a dataset, with the latter being defined via counts of items from some finite universe. For simplicity we will assume an indexing that allows us to specify datasets as vectors $\mathbf{x} \in \mathbb{Z}_{\geqslant 0}^d$, and

we will consider a finite set Q of *linear* queries, i.e. ones that can be defined as an inner product of $\mathbf{x}$ with a vector $\mathbf{q} \in [-1,1]^d$. Here again we will incorporate a prediction into an existing algorithm, specifically the MWEM method of Hardt et al. [2012], which uses multiplicative weights to iteratively update a distribution over the data domain and to construct a synthetic dataset $\hat{\mathbf{x}} \in \mathbb{R}^d_{\geqslant 0}$ such that the maximum error $\max_{\mathbf{q}\in\mathcal{Q}} |\langle \mathbf{q}, \mathbf{x} - \hat{\mathbf{x}} \rangle|$ of all queries is small. The natural approach here is to assume the prediction can be written as a distribution $\mathbf{w} \in \triangle_d$ and use it instead of the uniform initialization used by Hardt et al. [2012]. Indeed this observation has been made in both the original work and by Liu et al. [2021a], who adapt the method to only operate over the support of a source dataset. A prediction-dependent guarantee also follows in a straightforward manner from the original analysis:[2]

Lemma 6.3.2. Algorithm 11 is ε-DP and produces $\hat{\mathbf{x}} \in \mathbb{R}^d_{\geqslant 0}$ s.t. w.p. $\geqslant 1 - \beta$

$$\max_{\mathbf{q}\in Q} \frac{|\langle \mathbf{q}, \mathbf{x} - \hat{\mathbf{x}} \rangle|^2}{n} \leqslant \frac{8n}{m} D_{\mathrm{KL}}\left(\frac{\mathbf{x}}{n} \middle\| \mathbf{w}\right) + \frac{16m^2}{\varepsilon^2 n}\left(3\log\frac{2m}{\beta} + 2\log^2|Q|\right)^2 \tag{6.9}$$

Our main purpose with this application is thus to discuss interesting issues arising in its robustness and especially in learning the prediction. We also conclude by noting the similarity of deriving prediction-based guarantees for all four methods—finding algorithms that implicitly use a default prediction such as a uniform distribution or zero matrix—even while the actual algorithms and uses of the predictions are quite different.

6.4 Robustness-consistency tradeoffs

While prediction-dependent guarantees work well if the prediction is accurate, without safeguards they may perform catastrophically poorly if the prediction is incorrect. In this section we provide robust alternatives to the methods we derived in the previous section, demonstrating the usefulness of the algorithms with predictions framework for understanding robustness when incorporating external information into DP algorithms.

6.4.1 Quantile estimation

While prediction-dependent guarantees work well if the prediction is accurate, without safeguards they may perform catastrophically poorly if the prediction is incorrect. Quantiles provide a prime demonstration of the importance of robustness, as using priors allows for approaches that may assign very little probability to the interval containing the quantile. For example, if one is confident that it has a specific value $x \in (a,b)$ one can specify a more concentrated prior, e.g. the Laplace distribution around x. Alternatively, if one believes the data is drawn i.i.d. from some a known distribution then μ can be constructed via its CDF using order statistics [David and Nagaraja, 2003, Equation 2.1.5]. These reasonable approaches can result in distributions with exponential or high-order-polynomial tails, using which directly may work poorly if the prediction is incorrect.

[2]Similar to covariance estimation, we consider the mean squared error for the purposes of learning the prediction.

Luckily, for our negative log-inner-product error metric it is straightforward to show a parameterized robustness-consistency tradeoff by simply mixing the prediction prior μ with a robust prior ρ:

Corollary 6.4.1. For any prior $\mu : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$, robust prior $\rho : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$, and robustness parameter $\lambda \in [0, 1]$, releasing $o \in \mathbb{R}$ w.p. $\propto \exp(-\varepsilon \operatorname{Gap}_q(\mathbf{x}, o)/2)\mu^{(\lambda)}(o)$ for $\mu^{(\lambda)} = (1 - \lambda)\mu + \lambda\rho$ is $\left(\frac{2}{\varepsilon} \log \frac{1/\beta}{\lambda \Psi_{\mathbf{x}}^{(q,\varepsilon)}(\rho)}\right)$-robust and $\left(\frac{2}{\varepsilon} \log \frac{1/\beta}{1-\lambda}\right)$-consistent w.p. $\geqslant 1 - \beta$.

Proof. Apply Lemma 6.A.1 and linearity of $\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu^{(\lambda)}) = (1 - \lambda)\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu) + \lambda\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\rho)$. □

Thus if the interval is finite and we set ρ to be the uniform prior, using $\mu^{(\lambda)}$ in the algorithm will have a high probability guarantee at most $\frac{2}{\varepsilon} \log \frac{1}{\lambda}$-worse than the prediction-free guarantee of Kaplan et al. [2022, Lemma A.1], no matter how poor μ is for the data, while also guaranteeing w.p. $\geqslant 1 - \beta$ that the error will be at most $\frac{2}{\varepsilon} \log \frac{1/\beta}{1-\lambda}$ if μ is perfect. A similar result holds for the case of an infinite interval if we instead use a Cauchy prior. Corollary 6.4.1 demonstrates the usefulness of the algorithms with predictions framework for not only quantifying improvement in utility using external information but also for making the resulting DP algorithms robust to prediction noise.

The above argument for single-quantiles is straightforward to extend to the negative log of the harmonic means of the inner products. In-fact for the binary case with uniform quantiles we can tradeoff between $\operatorname{polylog}(m)$-guarantees similar to those of Kaplan et al. [2022] and our prediction-dependent bounds:

Corollary 6.4.2. Consider priors $\mu_1, \ldots, \mu_m : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$, Cauchy prior $\rho : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$ with location $\frac{a+b}{2}$ and scale $\frac{b-a}{2}$, and robustness parameter $\lambda \in [0, 1]$. Then running Algorithm 13 on quantiles that are uniform negative powers of two with $K = 2$, edge-based prior adaptation, $\varepsilon_i = \bar{\varepsilon} = \varepsilon/\lceil \log_2 m \rceil \ \forall \ i$, and priors $\mu_i^{(\lambda)} = \lambda\rho + (1 - \lambda)\mu_i \ \forall \ i$ is $\left(\frac{2}{\varepsilon}\lceil \log_2 m \rceil^2 \log\left(\pi m \frac{b-a+\frac{4R^2}{b-a}}{2\lambda\beta\psi_{\mathbf{x}}}\right)\right)$-robust and $\left(\frac{2}{\varepsilon}\phi^{\log_2 m}\lceil \log_2 m \rceil \log \frac{m/\beta}{1-\lambda}\right)$-consistent w.p. $\geqslant 1 - \beta$.

Proof. Apply Lemma 6.A.2, Theorem 6.A.1, and linearity of the inner products in $\hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}$ and $\Psi_{\mathbf{x}}^{(\varepsilon)}$. □

6.4.2 Covariance estimation

We take a different approach to making our prediction-based covariance estimation method robust to matrices $\mathbf{W}$ with large trace distance to $\mathbf{X}\mathbf{X}^\top/n$. Instead of combining the prediction with a robust default, we simply spend some privacy to check whether $\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}$ is larger than $\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}$ and if so run Algorithm 10 with the zero matrix instead. This has the following guarantee:

Corollary 6.4.3. Pick $\lambda \in (0, 1)$ and run Algorithm 10 with privacy $(1 - \lambda)\varepsilon$ and symmetric prediction matrix $\mathbf{W}$ if $\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}} + z \leqslant \|\mathbf{X}\mathbf{X}^\top\|_{\mathrm{Tr}}/n$ and $\mathbf{0}_{d\times d}$ otherwise, where $z \sim \operatorname{Lap}(\frac{4}{\lambda\varepsilon n})$. This procedure is ε-DP, $\tilde{\mathcal{O}}\left(\frac{d\sqrt{d}}{\varepsilon n}\left(\frac{1}{\varepsilon n} + \|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}\right)\right)$-robust, and $\tilde{\mathcal{O}}\left(\frac{d\sqrt{d}}{\varepsilon^2 n^2}\right)$-consistent w.h.p.

Proof. By Lemma 6.B.1 the difference $\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}} - \|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}$ has sensitivity $4/n$, so the comparison of $\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}} + z$ and $\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}$ is equivalent to using the Laplace mechanism with $\lambda\varepsilon$-DP to estimate this difference and then taking the sign. Composing this with the privacy guarantee of Theorem 6.3.4 yields ε-DP. Since $\Pr(|z| \geqslant \frac{4}{\lambda\varepsilon n}\log\frac{2}{\beta}) \leqslant \beta/2$, the matrix $\mathbf{W}_z \in \{\mathbf{W}, \mathbf{0}_{d\times d}\}$ passed to Algorithm 10 satisfies $\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}_z\|_{\mathrm{Tr}} \leqslant \min\{\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}, \|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}\} + \frac{4}{\lambda\varepsilon n}\log\frac{2}{\beta}$ w.p. $\geqslant 1-\beta/2$. Applying the utility guarantee of Theorem 6.3.4 w.p. $1-\beta/2$ for constant $\lambda \in (0,1)$ yields the result. □

Adding this check for robustness make the data-independent term worse by a factor of $\sqrt{d}$; note that the data-dependent term can still be up to $\tilde{\mathcal{O}}(\varepsilon n\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}})$ times larger, so this does not remove the usefulness of the prediction guarantee. The additional cost results from the large dependence on d of this latter term in the original bound, which is itself be caused by a mismatch between the ℓ_1-sensitivity measure and the ℓ_2-bound on the columns. Specifically, if instead the ℓ_1-norms of the columns are assumed bounded by one then the ℓ_1-sensitivity of $\mathbf{X}\mathbf{X}^\top/n$ is $2/n$, making the numerator of the second term in Theorem 6.3.4 be $\tilde{\mathcal{O}}(\sqrt{d})$ and thus causing no (asymptotic) cost due to robustness.[3] Similarly, under the original assumption the corresponding term in the ℓ_2-sensitivity-based zCDP guarantee is also $\tilde{\mathcal{O}}(\sqrt{d})$ (c.f. Theorem 6.B.2) and leads to a term that is $\mathcal{O}(\sqrt{n/d})$ worse (multiplicatively) due to robustness (c.f. Corollary 6.B.2); while worse in some regimes, in sufficiently high dimensions ($d = \Omega(n)$) this means no (asymptotic) cost of robustness.

6.4.3 Data release

As with quantiles, a natural approach to making data release robust is to mix the initialization with the default uniform distribution, achieving a tunable tradeoff. In the following result we specify the number of steps based on the the worst-case guarantees for a prediction-free algorithm and obtain a favorable tradeoff that allows for very small values of λ for high consistency while still maintaining robustness due the latter's $\log\frac{d}{\lambda}$ dependence.

Corollary 6.4.4. For $d \geqslant 2$ and any $\mathbf{w} \in \triangle_d$, running Algorithm 11 with $m = \sqrt[3]{\frac{\varepsilon^2 n^2 \log d}{2\log^4|Q|}}$ and initialization $\mathbf{w}^{(\lambda)} = (1-\lambda)\mathbf{w} + \lambda\mathbf{1}_d/d$ is ε-DP, $\tilde{\mathcal{O}}\left((1+\log^{4/3}|Q|)\sqrt[3]{\frac{n}{\varepsilon^2\log d}}\log\frac{d}{\lambda}\right)$-robust, and $\tilde{\mathcal{O}}\left(\lambda(1+\log^{4/3}|Q|)\sqrt[3]{\frac{n\log^2 d}{\varepsilon^2}}\right)$-consistent w.h.p., where $\tilde{\mathcal{O}}$ hides poly-log terms in $\frac{1}{\varepsilon}, n, \log d$, and $\log|Q|$.

Proof. If $\mathbf{w} = \frac{\mathbf{x}}{n}$ then we have $D_{\mathrm{KL}}(\frac{\mathbf{x}}{n}||\mathbf{w}^{(\lambda)}) \leqslant (1-\lambda)D_{\mathrm{KL}}(\frac{\mathbf{x}}{n}||\mathbf{w}) + \lambda D_{\mathrm{KL}}(\frac{\mathbf{x}}{n}||\mathbf{w} \leqslant \lambda\log d$ by joint convexity of D_{KL}. On the other hand $D_{\mathrm{KL}}(\frac{\mathbf{x}}{n}||\mathbf{w}^{(\lambda)}) \leqslant \langle\frac{\mathbf{x}}{n}, \log\frac{d\mathbf{x}}{\lambda n}\rangle \leqslant \log\frac{d}{\lambda}$. Substituting into Lemma 6.3.2 and simplifying yields the result. □

[3] It is not as clear that the ℓ_1-sensitivity of the eigenvalues would be as affected by the different assumption.

Algorithm 12: Non-Euclidean DP-FTRL. For the `InitializeTree`, `AddToTree`, and `GetSum` subroutines see Kairouz et al. [2021a, Section B.1].

Input: Datasets $\mathbf{x}_1, \dots, \mathbf{x}_T$ arriving in a stream in arbitrary order, domain $\Theta \subset \mathbb{R}^p$, step-size $\eta > 0$, noise scale $\sigma > 0$, ℓ_2-sensitivity $\Delta_2 > 0$, regularizer $\phi : \Theta \mapsto \mathbb{R}$

$\mathbf{g}_1 \leftarrow \mathbf{0}_p$

$\mathcal{T} \leftarrow$`InitializeTree`(T, σ^2, Δ_2) `// start tree aggregation`

for $t = 1, \dots, T$ **do**

  $\boldsymbol{\theta}_t \leftarrow \arg\min_{\boldsymbol{\theta} \in \Theta} \phi(\boldsymbol{\theta}) + \eta \langle \mathbf{g}_t, \boldsymbol{\theta} \rangle$

  suffer $\ell_{\mathbf{x}_t}(\theta_t)$

  $\mathcal{T} \leftarrow$`AddToTree`$(\mathcal{T}, t, \nabla_\theta \ell_{\mathbf{x}_t}(\boldsymbol{\theta}_t))$ `// add gradient to tree`

  $\mathbf{g}_{t+1} \leftarrow$`GetSum`$(\mathcal{T}, t)$ `// estimate` $\sum_{s=1}^{t} \nabla_{\boldsymbol{\theta}} \ell_{\mathbf{x}_s}(\theta_s)$

6.5 Learning predictions, privately

Our last objective will be to *learn* predictions that do well according to the quality metrics we have defined, which themselves control the utility loss of running the DP algorithms. Past work, e.g. the public-private framework [Liu et al., 2021a, Bassily et al., 2022, Bie et al., 2022], has often focused on domain adaptation-type learning where we adapt a public source to private target. We avoid assuming access to large quantities of i.i.d. public data and instead assume numerous tasks that can have sensitive data and may be adversarially generated. As discussed before, this is the online setting where we see loss functions defined by a sequence of datasets $\mathbf{x}_1, \dots, \mathbf{x}_T$ and aim to compete with best fixed prediction in-hindsight. As in the previous chapter, such a a guarantee can also be converted into excess risk bounds (c.f. Lemma B.4.1).

6.5.1 Non-Euclidean DP-FTRL

Because the optimization domain is not well-described by the ℓ_2-ball, we are able to obtain significant savings in dependence on the dimension and in some cases even in the number of instances T by extending the DP-FTRL algorithm of Kairouz et al. [2021a] to use non-Euclidean regularizers, as in Algorithm 12. For this we prove the following regret guarantee:

Theorem 6.5.1. Let $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T$ be the outputs of Algorithm 12 using a regularizer $\phi : \Theta \mapsto \mathbb{R}$ that is strongly-convex w.r.t. $\|\cdot\|$. Suppose $\forall\, t \in [T]$ that $\ell_{\mathbf{x}_t}(\cdot)$ is L-Lipschitz w.r.t. $\|\cdot\|$ and its gradient has ℓ_2-sensitivity Δ_2. Then w.p. $\geqslant 1 - \beta'$ we have $\forall\, \boldsymbol{\theta}^* \in \Theta$ that

$$\sum_{t=1}^{T} \ell(\boldsymbol{\theta}_t; \mathbf{x}_t) - \ell(\boldsymbol{\theta}^*; \mathbf{x}_t) \leqslant \frac{\phi(\boldsymbol{\theta}^*) - \phi(\boldsymbol{\theta}_1)}{\eta} + \eta L \left(L + \left(G + C\sqrt{2 \log \frac{T}{\beta'}} \right) \sigma \Delta_2 \sqrt{\lceil \log_2 T \rceil} \right) T \tag{6.10}$$

where $G = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(\mathbf{0}_p, \mathbf{I}_p)} \sup_{\|\mathbf{y}\| \leqslant 1} \langle \mathbf{z}, \mathbf{y} \rangle = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(\mathbf{0}_p, 1)} \|\mathbf{z}\|_*$ is the Gaussian width of the unit $\|\cdot\|$-ball and C is the Lipschitz constant of $\|\cdot\|_*$ w.r.t. $\|\cdot\|_2$. Furthermore, for any $\varepsilon' \leqslant 2 \log \frac{1}{\delta'}$, setting $\sigma = \frac{1}{\varepsilon'} \sqrt{2 \lceil \log_2 T \rceil \log \frac{1}{\delta'}}$ makes the algorithm (ε', δ')-DP.

Proof. The privacy guarantee follows from past results for tree aggregation [Smith and Thakurta, 2013, Kairouz et al., 2021a]. For all $t \in [T]$ we use the shorthand $\nabla_t = \nabla_{\boldsymbol{\theta}} \ell_{\mathbf{x}_t}(\theta_t)$; we can then define $\tilde{\boldsymbol{\theta}}_t = \arg\min_{\boldsymbol{\theta} \in \Theta} \phi(\boldsymbol{\theta}) + \eta \sum_{s=1}^{t} \langle \nabla_s, \boldsymbol{\theta} \rangle$ and $\mathbf{b}_t = \mathbf{g}_t - \sum_{s=1}^{t} \nabla_s$. Then

$$\begin{aligned}
\sum_{t=1}^{T} \ell_{\mathbf{x}_t}(\boldsymbol{\theta}_t) - \ell_{\mathbf{x}_t}(\boldsymbol{\theta}^*) &\leqslant \sum_{t=1}^{T} \langle \nabla_t, \boldsymbol{\theta}_t - \boldsymbol{\theta}^* \rangle \\
&= \sum_{t=1}^{T} \langle \nabla_t, \tilde{\boldsymbol{\theta}}_t - \boldsymbol{\theta}^* \rangle + \sum_{t=1}^{T} \langle \nabla_t, \boldsymbol{\theta}_t - \tilde{\boldsymbol{\theta}}_t \rangle \\
&\leqslant \frac{\phi(\boldsymbol{\theta}^*) - \phi(\boldsymbol{\theta}_1)}{\eta} + \eta \sum_{t=1}^{T} \|\nabla_t\|_*^2 + \sum_{t=1}^{T} \|\nabla_t\|_* \|\tilde{\boldsymbol{\theta}}_t - \boldsymbol{\theta}_t\| \\
&\leqslant \frac{\phi(\boldsymbol{\theta}^*) - \phi(\boldsymbol{\theta}_1)}{\eta} + \eta L \left(LT + \sum_{t=1}^{T} \|\mathbf{b}_t\|_* \right)
\end{aligned} \tag{6.11}$$

where the first inequality follows from the standard linear approximation in online convex optimization [Zinkevich, 2003], the second by the regret guarantee for online mirror descent [Shalev-Shwartz, 2011, Theorem 2.15], and the last by applying McMahan [2017, Lemma 7] with $\phi_1(\cdot) = \phi(\cdot) + \eta \sum_{s=1}^{t} \langle \nabla_s, \cdot \rangle$, $\psi(\cdot) = \eta \langle \mathbf{b}_t, \cdot \rangle$, and $\phi_2(\cdot) = \phi(\cdot) + \eta \langle \mathbf{g}_t, \cdot \rangle$, yielding $\|\tilde{\boldsymbol{\theta}}_t - \boldsymbol{\theta}_t\| \leqslant \eta \|\mathbf{b}_t\|_* \; \forall \, t \in [T]$. The final guarantee follows by observing that the tree aggregation protocol adds noise $\mathbf{b}_t \sim \mathcal{N}(\mathbf{0}_p, \sigma^2 \Delta_2^2 \lceil \log_2 t \rceil)$ to each prefix sum and applying the Gaussian concentration of Lipschitz functions [Boucheron et al., 2012, Theorem 5.6]. □

The above proof of this result follows that of the Euclidean case, which can be recovered by setting $G = \mathcal{O}(\sqrt{d})$, $C = 1$, and $\Delta_2 = \mathcal{O}(L)$.[4] In addition to the Lipschitz constants L, a key term that can lead to improvement is the Gaussian width G of the unit $\|\cdot\|$-ball, which for the Euclidean case is $\mathcal{O}(\sqrt{d})$ but e.g. for $\|\cdot\| = \|\cdot\|_1$ is $\mathcal{O}(\sqrt{\log d})$. Note that a related dependence on the Laplace width of Θ appears in Agarwal and Singh [2017, Theorem 3.1], although their guarantee only holds for linear losses and is not obviously extendable. Thus Theorem 6.5.1 may be of independent interest for DP online learning.

6.5.2 Learning priors for one or more quantiles

We now turn to learning vectors $\boldsymbol{\mu}_t = \left(\boldsymbol{\mu}_{t[1]}, \cdots, \boldsymbol{\mu}_{t[m]} \right)$ or priors $\boldsymbol{\mu}_{[i]} : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$ to privately estimate m quantiles $q_1, \ldots, q_m$ on each of a sequence of T datasets $\mathbf{x}_t$. We will aim to set $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_T$ s.t. if at each time t we run Algorithm 13 with privacy $\varepsilon > 0$ then the guarantees given by Lemmas 6.A.1 and 6.A.2 will be asymptotically at least as good as those of the best set of measures in $\mathcal{F}^m$, where $\mathcal{F}$ is some class of measures on the finite interval (a, b). The latter we will assume to be known and bounded. Note that in this section almost all single-quantile results follow from setting $m = 1$, so we study it jointly with learning for multiple quantiles.

[4] As of this writing, the most recent arXiv version of Kairouz et al. [2021a, Theorem C.1] has a typo leading to missing a Lipschitz constant in the bound, confirmed via correspondence with the authors.

Ignoring constants, the loss functions implied by our prediction-dependent upper bounds for multiple-quantiles are the following negative log-harmonic sums of prior-EM inner-products:

$$U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}) = \log \sum_{i=1}^{m} \frac{1}{\Psi_{\mathbf{x}_t}^{(q_i,\varepsilon_i)}(\boldsymbol{\mu}_{[i]})} = \log \sum_{i=1}^{m} \frac{1}{\int_a^b \exp(-\varepsilon_i \operatorname{Gap}_{q_i}(\mathbf{x}_t, o)/2)\boldsymbol{\mu}_{[i]}(o)do} \tag{6.12}$$

We focus on minimizing regret $\max_{\boldsymbol{\mu}\in\mathcal{F}^m} \sum_{t=1}^{T} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_t) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu})$ over these losses for priors $\boldsymbol{\mu}_{[i]}$ in a class $\mathcal{F}_{V,d}$ of probability measures that are piecewise V-Lipschitz over each of d intervals uniformly partitioning $[a, b)$. This is chosen because it covers the class $\mathcal{F}_{V,1}$ of V-Lipschitz measures and the class of $\mathcal{F}_{0,d}$ of discrete measures that are constant on each of the d intervals. The latter can be parameterized by $\mathbf{W} \in \triangle_d^m$, so that the losses have the form $U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{W}}) = \log \sum_{i=1}^{m} \langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]} \rangle^{-1}$ for $\mathbf{s}_{t,i} \in \mathbb{R}_{\geqslant 0}^d$. This can be seen by setting

$$\mathbf{s}_{t,i[j]} = \frac{d}{b-a} \int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j} \exp(-\varepsilon_i \operatorname{Gap}_{q_i}(\mathbf{x}_t, o)/2)do \tag{6.13}$$

and $\boldsymbol{\mu}_{\mathbf{W}[i]}(o) = \frac{d}{b-a}\mathbf{W}_{[i,j]}$ over the interval $\left[a + \frac{b-a}{d}(j-1), a + \frac{b-a}{d}j\right)$. Finally, for $\lambda \in [0, 1]$ we also let $\mathcal{F}^{(\lambda)} = \{(1-\lambda)\mu + \frac{\lambda}{b-a} : \mu \in \mathcal{F}\}$ denote the class of mixtures of measures $\mu \in \mathcal{F}$ with the uniform measure.

As detailed in Appendix 6.D.1, losses of the form $-\log\langle \mathbf{s}_t, \cdot \rangle$, i.e. those above when $m = 1$, have been studied in (non-private) online learning [Hazan et al., 2007], including in this thesis (c.f. Section 2.3). However, specialized approaches, e.g. those taking advantage exp-concavity, are not obviously implementable via prefix sums of gradients, the standard approach to private online learning [Smith and Thakurta, 2013, Agarwal and Singh, 2017, Kairouz et al., 2021a]. Still, we can at least use the fact that we are optimizing over a product of simplices to improve the dimension-dependence by applying Non-Euclidean DP-FTRL with entropic regularizer $\phi(\mathbf{W}) = m\langle \mathbf{W}, \log \mathbf{W} \rangle$, which yields an m-way exponentiated gradient (EG) update [Kivinen and Warmuth, 1997]. To apply its guarantee for the problem of learning priors for quantile estimation, we need to bound the sensitivity of the gradients $\nabla_{\mathbf{W}} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{W}})$ to changes in the underlying datasets $\mathbf{x}_t$. This is often done via a bound on the gradient norm, which in our case is unbounded near the boundary of the simplex. We thus restrict to γ-robust priors for some $\gamma \in (0, 1]$ by constraining $\mathbf{W} \in \triangle_d^m$ to have entries lower bounded by γ/d—a domain where $\|\nabla_{\mathbf{W}} U_{\mathbf{x}_t}^{(\varepsilon)}(\mu_{\mathbf{W}})\|_1 \leqslant d/\gamma$ (c.f. Lemma 6.D.1)—and bounding the resulting approximation error; we are not aware of even a non-private approach that avoids this except by taking advantage of exp-concavity [Hazan et al., 2007].

We thus have a bound of $2d/\gamma$ on the ℓ_2-sensitivity. However, this may be too loose since it allows for changing the entire dataset $\mathbf{x}_t$, whereas we are only interested in changing one entry. Indeed, for small ε we can obtain a tighter bound:

Lemma 6.5.1. The ℓ_2-sensitivity of $\nabla_{\mathbf{w}} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{w}})$ is $\frac{d}{\gamma}\min\{2, e^{\tilde{\varepsilon}_m} - 1\}$, where $\tilde{\varepsilon}_m = (1 + 1_{m>1}) \max_i \varepsilon_i$.

Proof for $m = 1$; c.f. Appendix 6.D.1. Let $\tilde{\mathbf{x}}_t$ be a neighboring dataset of $\mathbf{x}_t$ and let $U^{(\varepsilon)}_{\tilde{\mathbf{x}}_t}(\boldsymbol{\mu}_{\mathbf{W}}) = -\log\langle\tilde{\mathbf{s}}_t, \mathbf{w}\rangle$ be the corresponding loss. Note that $\max_{o\in[a,b]} |\operatorname{Gap}_q(\mathbf{x}_t, o) - \operatorname{Gap}_q(\tilde{\mathbf{x}}_t, o)| \leqslant 1$ so

$$\begin{aligned}\tilde{\mathbf{s}}_{t[j]} &= \int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j} \exp\left(-\frac{\varepsilon}{2}\operatorname{Gap}_q(\tilde{\mathbf{x}}_t, o)\right) do \in e^{\pm\frac{\varepsilon}{2}} \int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j} \exp\left(-\frac{\varepsilon}{2}\operatorname{Gap}_q(\mathbf{x}_t, o)\right) do \\ &= e^{\pm\frac{\varepsilon}{2}}\mathbf{s}_{t[j]}\end{aligned} \tag{6.14}$$

Therefore since $m = 1$ we denote $\mathbf{w} = \mathbf{W}_{[1]}$, $\mathbf{s}_t = \mathbf{s}_{t,1}$, and $\tilde{\mathbf{s}}_t = \tilde{\mathbf{s}}_{t,1}$ and have

$$\begin{aligned}\|\nabla_{\mathbf{w}} U^{(\varepsilon)}_{\mathbf{x}_t}(\boldsymbol{\mu}_{\mathbf{w}}) - \nabla_{\mathbf{w}} U^{(\varepsilon)}_{\tilde{\mathbf{x}}_t}(\boldsymbol{\mu}_{\mathbf{w}})\|_2 &= \sqrt{\sum_{j=1}^{d}\left(\frac{\mathbf{s}_{t[j]}}{\langle\mathbf{s}_t, \mathbf{w}\rangle} - \frac{\tilde{\mathbf{s}}_{t[j]}}{\langle\tilde{\mathbf{s}}_t, \mathbf{w}\rangle}\right)^2} \\ &= \sqrt{\sum_{j=1}^{d}\frac{\mathbf{s}_{t[j]}^2}{\langle\mathbf{s}_t, \mathbf{w}\rangle^2}\left(1 - \frac{\tilde{\mathbf{s}}_{t[j]}\langle\mathbf{s}_t, \mathbf{w}\rangle}{\mathbf{s}_{t[j]}\langle\tilde{\mathbf{s}}_t, \mathbf{w}\rangle}\right)^2} \\ &\leqslant \|\nabla_{\mathbf{w}} U^{(\varepsilon)}_{\mathbf{x}_t}(\boldsymbol{\mu}_{\mathbf{w}})\|_1 \max_j |1 - \kappa_j|\end{aligned} \tag{6.15}$$

where $\kappa_j = \frac{\tilde{\mathbf{s}}_{t[j]}\langle\mathbf{s}_t,\mathbf{w}\rangle}{\mathbf{s}_{t[j]}\langle\tilde{\mathbf{s}}_t,\mathbf{w}\rangle} \in \frac{\mathbf{s}_{t[j]}\exp(\pm\frac{\varepsilon}{2})\langle\mathbf{s}_t,\mathbf{w}\rangle}{\mathbf{s}_{t[j]}\langle\mathbf{s}_t,\mathbf{w}\rangle\exp(\pm\frac{\varepsilon}{2})} \in \exp(\pm\varepsilon)$ by Equation 6.14. The result follows by taking the minimum with the bound on the Euclidean norm of the gradient (Lemma 6.D.1). □

Since $e^{\varepsilon} - 1 \leqslant 2\varepsilon$ for $\varepsilon \in (0, 1.25]$, for small ε this allows us to add less noise in DP-FTRL. With this sensitivity bound, we apply Algorithm 12 using the entropic regularizer to obtain the following result (c.f. Appendix 6.D.1):

Theorem 6.5.2. For $d \geqslant 2, \gamma \in (0, \frac{1}{2}]$ if we run Algorithm 12 on $U^{(\varepsilon)}_{\mathbf{x}_t}(\boldsymbol{\mu}_{\mathbf{W}}) = \log\sum_{i=1}^{m}\frac{1}{\Psi^{(q_i,\varepsilon_i)}_{\mathbf{x}_t}(\boldsymbol{\mu}_{\mathbf{W}})}$ over γ-robust priors with step-size $\eta = \frac{\gamma m}{d}\sqrt{\frac{\log(d)/T}{1+\left(2\sqrt{\log(md)}+\sqrt{2\log\frac{T}{\beta'}}\right)\sigma\sqrt{\log\lceil\log_2 T\rceil}\min\{1,\tilde{\varepsilon}_m\}}}$ and regularizer $\phi(\mathbf{W}) = m\langle\mathbf{W}, \log\mathbf{W}\rangle$ then for any $V \geqslant 0$, $\lambda \in [0, 1]$, and $\beta' \in (0, 1]$ we will have regret

$$\begin{aligned}&\max_{\boldsymbol{\mu}_{[i]}\in\mathcal{F}^{(\lambda)}_{V,d}} \sum_{t=1}^{T} U^{(\varepsilon)}_{\mathbf{x}_t}(\boldsymbol{\mu}_{\mathbf{W}_t}) - U^{(\varepsilon)}_{\mathbf{x}_t}(\boldsymbol{\mu}) \\ &\qquad\leqslant \frac{VmT}{\gamma d\bar{\psi}}(b-a)^3 + 2\max\{\gamma - \lambda, 0\}T\log 2 \\ &\qquad\quad + \frac{2md}{\gamma}\sqrt{\left(1 + \left(4\sqrt{\log(md)} + 2\sqrt{2\log\frac{T}{\beta'}}\right)\sigma\sqrt{\lceil\log_2 T\rceil}\min\{1, \tilde{\varepsilon}_m\}\right)T\log d}\end{aligned} \tag{6.16}$$

w.p. $\geqslant 1 - \beta'$, where $\bar{\psi}$ is the harmonic mean of $\psi_{\mathbf{x}_t} = \min_k \mathbf{x}_{t[k+1]} - \mathbf{x}_{t[k]}$ and $\tilde{\varepsilon}_m = (1 + 1_{m>1})\max_i \varepsilon_i$. For any $\varepsilon' \leqslant 2\log\frac{1}{\delta'}$ setting $\sigma = \frac{1}{\varepsilon'}\sqrt{2\lceil\log_2 T\rceil\log\frac{1}{\delta'}}$ makes this procedure (ε', δ')-DP.

Note that in the case of $V > 0$ or $\lambda = 0$ we will need to set $d = \omega_T(1)$ or $\gamma = o_T(1)$ in order to obtain sublinear regret. Thus for these more difficult classes our extension of DP-FTRL to non-Euclidean regularizers yields improved rates, as in the Euclidean case the first term has an extra $\sqrt[4]{d}$-factor. The following provides some specific upper bounds derived from Theorem 6.5.2:

Corollary 6.5.1. For each of the following classes of priors there exist settings of d (where needed) and $\gamma > 0$ in Theorem 6.5.2 that guarantee obtain the following regret w.p. $\geqslant 1 - \beta'$:

1. λ-robust and discrete $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{0,d}^{(\lambda)}$: $\tilde{\mathcal{O}}\left(\frac{dm}{\lambda}\sqrt{\left(1 + \frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right)T}\right)$
2. λ-robust and V-Lipschitz $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{V,1}^{(\lambda)}$: $\tilde{\mathcal{O}}\left(\frac{m}{\lambda}\sqrt{\frac{V}{\psi}}\sqrt[4]{\left(1 + \frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right)T^3}\right)$
3. discrete $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{0,d}$: $\tilde{\mathcal{O}}\left(\sqrt{dm}\sqrt[4]{\left(1 + \frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right)T^3}\right)$
4. V-Lipschitz $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{V,1}$: $\tilde{\mathcal{O}}\left(\sqrt{m}\sqrt[4]{\frac{V}{\psi}}\sqrt[8]{\left(1 + \frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right)T^7}\right)$

Thus competing with λ-robust priors with discrete PDFs enjoys the fastest regret rate of $\tilde{\mathcal{O}}(\sqrt{T})$, while either removing robustness or competing with any V-Lipschitz prior has regret $\tilde{\mathcal{O}}(T^{3/4})$, and doing both has regret $\tilde{\mathcal{O}}(T^{7/8})$. When comparing to Lipschitz priors we also incur a dependence on the inverse of minimum datapoint separation, which may be small. A notable aspect of all the bounds is that the regret *improves* with small ε due to the sensitivity analysis in Lemma 6.5.1; indeed for $\varepsilon = \mathcal{O}(\varepsilon')$ the regret bound only has a $\mathcal{O}(\log \frac{1}{\delta'})$-dependence on the privacy guarantee. Finally, for λ-robust priors we can also apply the $\log \frac{b-a}{\lambda\psi}$-boundedness of $-\log \Psi_{\mathbf{x}}^{(q,\varepsilon)}(\boldsymbol{\mu})$ and standard online-to-batch conversion (c.f. Appendix B.4) to obtain the following sample complexity guarantee:

Corollary 6.5.2. For any $\alpha > 0$ and distribution $\mathcal{D}$ over finite datasets $\mathbf{x}$ of ψ-separated points from (a, b), if we run the algorithm in Theorem 6.5.2 on

$$T = \Omega\left(\frac{\log \frac{1}{\beta'}}{\alpha^2}\left(\frac{d^2m^2}{\lambda^2}\left(1 + \frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right) + \log^2\frac{1}{\lambda\psi}\right)\right) \tag{6.17}$$

i.i.d. samples from $\mathcal{D}$ then w.p. $\geqslant 1 - \beta'$ the average $\hat{\mathbf{W}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{W}_t$ of the resulting iterates satisfies

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \log \sum_{i=1}^{m} \frac{1}{\Psi_{\mathbf{x}}^{(q_i,\varepsilon_i)}(\boldsymbol{\mu}_{\hat{\mathbf{W}}_{[i]}})} \leqslant \min_{\boldsymbol{\mu}_{[i]}\in\mathcal{F}_{0,d}^{(\lambda)}} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \log \sum_{i=1}^{m} \frac{1}{\Psi_{\mathbf{x}}^{(q_i,\varepsilon_i)}(\boldsymbol{\mu}_{[i]})} + \alpha \tag{6.18}$$

For α-suboptimality w.r.t. $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{V,1}^{(\lambda)}$ the sample complexity is

$$T = \Omega\left(\frac{\log \frac{1}{\beta'}}{\alpha^2}\left(\frac{V^2m^2}{\lambda^4\psi^2\alpha^2}\left(1 + \frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right) + \log^2\frac{1}{\lambda\psi}\right)\right) \tag{6.19}$$

6.5.3 Learning to estimate covariance matrices

We next study how to learn prediction matrices for DP covariance estimation by targeting the trace distance between them and the ground truth. This is a more straightforward learning task,

with Lipschitz losses over a finite-dimensional domain. Indeed, we could apply standard DP-FTRL and obtain regret $\tilde{\mathcal{O}}(\sqrt{(1+d/\varepsilon')dT})$ w.r.t. any symmetric matrix $\mathbf{W}$ because the losses $U_{\mathbf{X}_t}(\mathbf{W}) = \|\mathbf{X}_t\mathbf{X}_t^\top/|\mathbf{X}_t| - \mathbf{W}\|_{\mathrm{Tr}}$ are $\sqrt{d}$-Lipschitz w.r.t. the Frobenius norm. However, we can reduce the dependence on the dimension by a $\sqrt{d}$-factor by combining our non-Euclidean DP-FTRL algorithm with the well-known matrix-learning technique of using Schatten p-norm regularization [Duchi et al., 2010]:

Theorem 6.5.3. Let $\mathbf{X}_1, \dots, \mathbf{X}_T$ be a sequence of datasets with d-dimensional columns bounded by 1 in the ℓ_2-norm. If we run Algorithm 12 on losses $U_{\mathbf{X}_t}(\mathbf{W}) = \|\mathbf{X}_t\mathbf{X}_t^\top/|\mathbf{X}_t| - \mathbf{W}\|_{\mathrm{Tr}}$ with step-size $\eta = \sqrt{\frac{6\log(d)/T}{1+\left(\sqrt{d}+\sqrt{2\log\frac{T}{\beta'}}\right)\sigma\sqrt{d\lceil\log_2 T\rceil}}}$ and regularizer $\phi(\cdot) = \frac{3}{2}\log d\|\cdot\|_p^2$ then we will have regret

$$\max_{\mathbf{W}\in\mathbb{R}^{d\times d}} \sum_{t=1}^{T} U_{\mathbf{X}_t}(\mathbf{W}_t) - U_{\mathbf{X}_t}(\mathbf{W}) \leqslant \mathcal{O}\left(\sqrt{\left(1+\left(\sqrt{d}+\sqrt{\log\frac{T}{\beta'}}\right)\sigma\sqrt{d\lceil\log_2 T\rceil}\right)T\log d}\right) \tag{6.20}$$

w.p. $\geqslant 1-\beta'$. For any $\varepsilon' \leqslant 2\log\frac{1}{\delta'}$ setting $\sigma = \frac{1}{\varepsilon'}\sqrt{2\lceil\log_2 T\rceil\log\frac{1}{\delta'}}$ makes this procedure (ε', δ')-DP. Furthermore, suppose the datasets are drawn i.i.d. from some distribution $\mathcal{D}$. If we run the same algorithm and return the average prediction $\hat{\mathbf{W}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{W}_t$ then $T = \tilde{\Omega}\left(\frac{1+d/\varepsilon'}{\alpha^2}\log\frac{1}{\beta'}\right)$ samples suffice to guarantee that w.p. $1-\beta'$

$$\mathbb{E}_{\mathbf{X}\sim\mathcal{D}}\|\mathbf{X}\mathbf{X}^\top/|\mathbf{X}| - \hat{\mathbf{W}}\|_{\mathrm{Tr}} \leqslant \min_{\mathbf{W}} \mathbb{E}_{\mathbf{X}\sim\mathcal{D}}\|\mathbf{X}\mathbf{X}^\top/|\mathbf{X}| - \mathbf{W}\|_{\mathrm{Tr}} + \alpha \tag{6.21}$$

Proof. The loss functions $\|\mathbf{X}_t\mathbf{X}_t^\top/|\mathbf{X}_t| - \mathbf{W}\|_{\mathrm{Tr}}$ have gradients $-\mathbf{U}_t\mathbf{S}_t\mathbf{U}_t^\top$, where $\mathbf{U}_t$ is the matrix of eigenvectors of $\mathbf{X}_t\mathbf{X}_t^\top/|\mathbf{X}_t| - \mathbf{W}$ and $\mathbf{S}_t$ is the diagonal matrix of the signs of its eigenvalues; the losses are thus $\sqrt{d}$-Lipschitz w.r.t. the Frobenius norm and 1-Lipschitz w.r.t. the trace norm. Note that these gradients can be computed in polynomial time via eigendecomposition and used in DP-FTRL with the Schatten-p norm regularizer $\frac{3}{2}\log d\|\cdot\|_p^2$ for $p = 1 + 1/\log d$, which is strongly-convex w.r.t. the trace norm $\|\cdot\|_{\mathrm{Tr}}$ [Duchi et al., 2010]. Since the Gaussian width of the (symmetric) trace ball is $\mathcal{O}(\sqrt{d})$ [Latała et al., 2018] and the spectral norm is 1-Lipschitz w.r.t. the Frobenius norm, applying Theorem 6.5.1 yields the bound

$$\frac{3\log d\|\mathbf{W}\|_p^2}{2\eta} + \eta\left(1+\left(\mathcal{O}(\sqrt{d}) + \sqrt{2\log\frac{T}{\beta'}}\right)\sigma\sqrt{d\lceil\log_2 T\rceil}\right)T \tag{6.22}$$

For any optimal $\mathbf{W}$ we have

$$\begin{aligned}\|\mathbf{W}\|_p \leqslant \|\mathbf{W}\|_{\mathrm{Tr}} &\leqslant \frac{1}{T}\sum_{t=1}^{T}\|\mathbf{W} - \mathbf{X}_t\mathbf{X}_t^\top/|\mathbf{X}_t|\|_{\mathrm{Tr}} + \|\mathbf{X}_t\mathbf{X}_t^\top/|\mathbf{X}_t|\|_{\mathrm{Tr}} \\ &\leqslant \frac{2}{T}\sum_{t=1}^{T}\mathrm{Tr}(\mathbf{X}_t\mathbf{X}_t^\top)/|\mathbf{X}_t| \leqslant 2\end{aligned} \tag{6.23}$$

so the regret follows by substituting η. The sample complexity result follows from online-to-batch conversion (c.f. Lemma B.4.1). □

Thus prediction matrices for covariance estimation are efficiently and privately learnable, in both the online and distributional settings. Moreover, for both our extension to non-Euclidean DP-FTRL is critical for obtaining a weaker dependence on the dimension. One limitation of the analysis is that, unlike for quantiles, we did not conduct a refined analysis by studying how swapping single columns of $\mathbf{X}_t$ rather than the entire dataset affects the gradient of $U_{\mathbf{X}_t}$. It is not immediately clear that an improvement is possible, with the difficulty being the gradient's dependence on the signs of the eigenvalues.

6.5.4 Learning the initialization and number of iterations for data release

Finally, we learn to initialize MWEM-based data release. Here we are faced with optimizing

$$U_{\mathbf{x}_t}(\mathbf{w}, m) = \frac{8n_t}{m} D_{\mathrm{KL}}\left(\frac{\mathbf{x}_t}{n_t}||\mathbf{w}\right) + \frac{16m^2}{\varepsilon^2 n + t}\left(3\log\frac{2m}{\beta} + 2\log^2|Q|\right)^2 \tag{6.24}$$

Notably, unlike the past learning settings, this function is parameterized by both a prediction $\mathbf{w}$ and the number of steps m, which we will also set online. The reason for this is that the optimal step-size depends on the similarity between instances: if for the optimal $\mathbf{w}$ the measure $D_{\mathrm{KL}}(\mathbf{x}_t/n_t||\mathbf{w})$ is small for most datasets $\mathbf{x}_t$ then it is better to set a small m above, whereas if it is usually large it should be counter-acted with a larger m. Our goal will thus be to set $\mathbf{w}_t$ and m_t together in an online fashion so as to *simultaneously* compete with the optimal λ-robust $\mathbf{w} \in \triangle_d$ for some $\lambda > 0$ and the optimal number of steps $m > 0$. To do so we will run DP-FTRL with the entropic regularizer, i.e. private exponentiated gradient (EG), to set both the initialization from the simplex $\triangle_d$ and to set the number of iterations m_t at step t by sampling from a categorical distribution. This has the following regret guarantee (c.f. Appendix 6.D.3):

Theorem 6.5.4. Let $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{Z}^d_{\geqslant 0}$ be a sequence of datasets with $n_t = \|\mathbf{x}_t\|_1$ entries each, let $N = \max_t n_t$, and consider any $\gamma \in (0, 1]$ and $\lambda \in [0, 1]$. Then there exists $M \in \mathbb{Z}_{>0}$ and $\eta_1, \sigma_1, \eta_2, \sigma_2 > 0$ s.t. running DP-FTRL with regularizer $\phi(\mathbf{w}) = \langle \mathbf{w}, \log \mathbf{w}\rangle$, step-size η_1, and noise σ_1 on the losses $n_t D_{\mathrm{KL}}(\frac{\mathbf{x}_t}{n_t}||\mathbf{w})$ over the domain $\mathbf{w}_{[i]} \geqslant \gamma/d$ to set $\mathbf{w}_t$ and simultaneously running DP-FTRL with regularizer $\phi(\boldsymbol{\theta}) = \langle \boldsymbol{\theta}, \log \boldsymbol{\theta}\rangle$, step-size η_2, and noise σ_2 on the losses $\mathbb{E}_{m\sim\boldsymbol{\theta}} U_{\mathbf{x}_t}(\mathbf{w}_t, m)$ over the domain $\triangle_M$ and setting m_t using the categorical distribution defined by $\boldsymbol{\theta}_t$ over the $\mathrm{Unif}\{1, \ldots, M\}$ such that the entire scheme is (ε', δ')-DP and w.h.p. has regret

$$\tilde{\mathcal{O}}\left(\left(\frac{N^{\frac{4}{3}}}{\min\{1, \varepsilon^2\}} + \frac{N^{\frac{2}{3}}/\sqrt{\varepsilon'}}{\min\{1, \varepsilon\}} + \frac{dN}{\gamma} + \frac{d}{\gamma}\sqrt{\frac{N}{\varepsilon'}}\right)\sqrt{T} + \max\{\gamma - \lambda, 0\}NT\right) \tag{6.25}$$

w.r.t. any $m > 0$ and $\mathbf{w} \in \triangle_d$ satisfying $\mathbf{w}_{[i]} \geqslant \frac{\lambda}{d}$. For $\lambda > 0$ setting $\gamma = \lambda$ yields regret $\tilde{\mathcal{O}}\left(\frac{d}{\lambda \min\{1,\varepsilon^2\}} N^{\frac{4}{3}}\sqrt{\frac{T}{\varepsilon'}}\right)$; for $\lambda = 0$ and $T \geqslant d^2$, setting $\gamma = \frac{\sqrt{d}}{\sqrt[4]{T}}$ has regret $\tilde{\mathcal{O}}\left(\frac{N^{\frac{4}{3}}}{\min\{1,\varepsilon^2\}} T^{\frac{3}{4}}\sqrt{\frac{d}{\varepsilon'}}\right)$.

As in quantile learning, we suffer a strong dependence on the dimension here, and the rate is worse if we try to compete the non-robust initializations ($\lambda = 0$). It thus remains an open question whether either a better learning result or upper bound is possible. Nevertheless, to interpret this guarantee, note that for $H_\lambda = \min_{\mathbf{w}_{[i]} \geqslant \lambda/d}\left(\sum_{t=1}^T n_t D_{\mathrm{KL}}(\mathbf{x}_t/n_t||\mathbf{w})\right) / \left(\sum_{t=1}^T n_t\right)$ and if

$n_t = n \; \forall \; t$ then we have that the optimum-in-hindsight for the average upper bound is

$$\min_{m>0, \mathbf{w}_{[i]} \geqslant \lambda/d} \frac{1}{T} \sum_{t=1}^{T} U_{\mathbf{x}_t}(\mathbf{w}, m) = \tilde{\mathcal{O}} \left(\frac{\log^{\frac{4}{3}} |Q|}{\varepsilon^{\frac{2}{3}}} \frac{\left(\frac{1}{T} \sum_{t=1}^{T} n_t \right)^{\frac{2}{3}}}{\left(T / \sum_{t=1}^{T} \frac{1}{n_t} \right)^{\frac{1}{3}}} H_\lambda^{\frac{2}{3}} \right) = \tilde{\mathcal{O}} \left(\sqrt[3]{\frac{n \log^4 |Q|}{\varepsilon^2}} H_\lambda^2 \right) \tag{6.26}$$

Since H_λ approximates the entropy of the aggregate distribution across instances $\mathbf{x}_1, \ldots, \mathbf{x}_T$—indeed for $\lambda = 0$ it is exactly the entropy of the average distribution $\left(\sum_{t=1}^{T} \mathbf{x}_t \right) / \sum_{t=1}^{T} n_t$—the regret guarantee shows that we will do well asymptotically if the entropy is small. Note that being able to choose m in addition to $\mathbf{w}$ is crucial to adapting to this entropy, and is closely related to the problem of choosing the step-size in meta-learning, where similar aggregate measures appear as forms of task similarity (c.f. Sections 2.2 and 2.3).

6.6 Applications

Having derived prediction-dependent performance bounds for three DP tasks and analyzed their robustness and learnability, we now investigate how these algorithms might be deployed in practice. We focus on the problem of multiple quantile release and consider the two motivating settings from the introduction: public-private transfer and sequential release. While we make direct use of the robust mixing scheme devised in Section 6.4, our learnability analysis in Section 6.5 yielded unwieldy discretization-based algorithms due to the focus on approximating very general priors. This generality seems unnecessary, as we might reasonably expect simple, unimodal distributions to be good priors for quantiles.

We thus consider instead the problem of optimizing the performance bounds $U_{\mathbf{x}} = -\log \Psi_{\mathbf{x}}$ for multiple quantile release across classes of **location-scale** priors, which for some measure $f : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$ have the form $\mu_{\nu,\sigma}(x) = \frac{1}{\sigma} f\left(\frac{x-\nu}{\sigma}\right)$ for $\nu \in \mathbb{R}$ and $\sigma > 0$. Such families allows us to model both the location of a quantile using $\nu = \langle \mathbf{w}, \mathbf{f} \rangle$—where $\mathbf{w} \in \mathbb{R}^d$ is a linear model from public features $\mathbf{f} \in \mathbb{R}^d$ about the dataset—and our uncertainty about it using σ, all while staying in reasonable dimensions. Note that in this section we target only the ε-independent bound $U_{\mathbf{x}}$, as $U_{\mathbf{x}}^{(\varepsilon)}$ does not yield a convex objective; furthermore, while we mainly discuss the single-quantile bound $U_{\mathbf{x}}^{(q)}$ for simplicity, the general results (c.f. Section 6.E) extend naturally to the case of $m > 1$ because it is the log-sum-exp of the former.

6.6.1 Convexity vs. robustness of location-scale models

We must first determine which location-scale family to use, as this include Gaussians with mean ν and variance σ^2, Laplace with mean ν and scale σ, Cauchy with location ν and scale σ, and more. To make this decision, we consider two desiderata: (1) the prior should be robust in the way the Cauchy is robust, i.e. being wrong about the data location should not harm us too much, and (2) it should be easy to learn the parameters ν and σ, e.g. by optimizing $U_{\mathbf{x}}^{(q)}(\mu_{\nu,\sigma})$.

While not necessary, one way of ensuring (2) is convexity of $U_{\mathbf{x}}^{(q)}$, which we focus on as it enables efficient algorithms. We use a connection between our upper bounds and the likelihood

of **censored regression** [Pratt, 1981], which for noise $\xi_i \in \mathbb{R}$ models a relationship between features $\mathbf{f}_i \in \mathbb{R}^d$ and a variable $y_i = \langle \mathbf{w}, \mathbf{f}_i \rangle + \xi_i$ when information about y_i is only provided via an interval $[a_i, b_i)$ containing it (e.g. an individual's income bracket, not their exact income). If ξ_i is from a location-scale distribution with $\nu = 0$ the log-likelihood given datapoints $(a_i, b_i, \mathbf{f}_i)$ is

$$\mathcal{L}_{\{a_i,b_i,\mathbf{f}_i\}_{i=1}^n}(\mathbf{w}, \sigma) = \sum_{i=1}^{n} \log \int_{a_i}^{b_i} \frac{1}{\sigma} f\left(\frac{y - \langle \mathbf{w}, \mathbf{f}_i \rangle}{\sigma}\right) dy \tag{6.27}$$

Observe that for $a = \mathbf{x}_{[\lfloor qn \rfloor]}$ and $b = \mathbf{x}_{[\lfloor qn \rfloor]+1}$ we have

$$U_{\mathbf{x}}^{(q)}(\mu_{\langle \mathbf{w},\mathbf{f} \rangle,\sigma}) = -\log \mu_{\langle \mathbf{w},\mathbf{f} \rangle,\sigma}((a, b]) = -\log \int_a^b \frac{1}{\sigma} f\left(\frac{o - \langle \mathbf{w}, \mathbf{f} \rangle}{\sigma}\right) do \tag{6.28}$$

which is the negative of $\mathcal{L}_{a,b,\mathbf{f}}(\mathbf{w}, \sigma)$. We thus adopt the reparameterization of Burridge [1981], who showed that (6.27) is concave w.r.t. $(\mathbf{v}, \phi) = (\frac{\mathbf{w}}{\sigma}, \frac{1}{\sigma})$ whenever f is **log-concave**, a property satisfied by the Gaussian and Laplace families but not the Cauchy. Therefore, for such f we have that $\ell_{\mathbf{x}}^{(q)}(\langle \mathbf{v}, \mathbf{f} \rangle, \phi) = U_{\mathbf{x}}^{(q)}(\mu_{\frac{\langle \mathbf{v},\mathbf{f} \rangle}{\phi}, \frac{1}{\phi}})$ is convex w.r.t. $(\mathbf{v}, \phi)$.

Unfortunately, we show that no log-concave f is robust, in the sense that for any $R > 0$ there exists a dataset of points in the interval $(\theta \pm R)^n$ s.t. $U_{\mathbf{x}}^{(q)}(\mu_{\theta,1}) = \Omega(R)$ (rather than $\mathcal{O}(\log(1 + R^2))$ as shown for the Cauchy family in Corollary 6.3.1). On the other hand, log-concave location-scale families are the only ones for which $U_{\mathbf{x}}^{(q)}$ is convex, both for the original parameterization and that of Burridge [1981]. We record these facts in the following theorem:

Theorem 6.6.1 (c.f. Thm. 6.E.1)**.** Let $\mu_{\nu,\sigma}$ be a location-scale family associated with a continuous measure $f : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$.

1. If f is log-concave then $\exists\ a, b > 0$ s.t. for any $R > 0$, $\psi \in (0, \frac{R}{2n}]$, $q \geqslant \frac{1}{n}$, and $\theta \in \mathbb{R}$ there exists $\mathbf{x} \in (\theta \pm R)^n$ with $\min_i \mathbf{x}_{[i+1]} - \mathbf{x}_{[i]} = \psi$ s.t. $U_{\mathbf{x}}^{(q)}(\mu_{\theta,1}) = aR + \log \frac{b}{\psi}$.
2. If f is not log-concave then there exists $\mathbf{x} \in \mathbb{R}^n$ with $\min_i \mathbf{x}_{[i+1]} - \mathbf{x}_{[i]} > 0$ s.t. $U_{\mathbf{x}}^{(q)}(\mu_{\theta,1})$ is nonconvex in θ.

Note the latter dataset is not degenerate: for f strictly log-convex over $[a, b]$, any $\mathbf{x}$ whose optimal interval has length $< \frac{b-a}{2}$ has nonconvex $U_{\mathbf{x}}^{(q)}(\mu_{\theta,1}) = -\log \Psi_{\mathbf{x}}^{(q)}(\mu_{\theta,1})$. We must thus choose between having a robust location-scale family like the Cauchy or an easy-to-optimize log-concave one. As we can ensure robustness of the learned prior *post-hoc* using the approach of Section 6.4, we choose the latter. Specifically, we use the Laplace prior, as it is in some sense the most robust log-concave distribution (it has loss $\Theta(R)$ if $\mathbf{x} \in (\theta \pm R)^n$, whereas e.g. the Gaussian has loss $\Theta(R^2)$) and because it yields a numerically stable closed-form expression (6.103) for $\ell_{\mathbf{x}}^{(q)}(\theta, \phi)$ (unlike e.g. the Gaussian).

6.6.2 Augmenting quantile release using public data

We turn to two applications that depend on optimizing upper bounds $\ell_{\mathbf{x}}^{(q)}(\theta, \phi)$ on the performance of quantile release using the Laplace prior with scale $\frac{1}{\phi}$ and location $\frac{\theta}{\phi}$. While our final objective is small Gap_q, we will mainly discuss optimizing $\ell_{\mathbf{x}}^{(q)} = U_{\mathbf{x}}^{(q)}$, or its expectation if $\mathbf{x}$

is drawn from some distribution. In the former case this directly bounds (w.h.p.) the cost of multiple quantile release, while a bound on $\mathbb{E}_{\mathbf{x}} U_{\mathbf{x}}$ can bound $\mathbb{E}\,\mathrm{Gap}_{\max}$ by setting β appropriately. For example, using $\beta = \frac{2\pi^2}{\varepsilon n}\exp(2\sqrt{\log(2)\log(m+1)})$ in Theorem 6.3.3 implies $\mathrm{Gap}_{\max}$ has expectation at most

$$\mathcal{O}\left(\exp\left(2\sqrt{\log(2)\log(m+1)}\right)\frac{\log(\varepsilon mn)+\mathbb{E}_{\mathbf{x}} U_{\mathbf{x}}}{\varepsilon}\right) \tag{6.29}$$

Our first application is the popular setting where we have a large public dataset $\mathbf{x}' \in \mathbb{R}^N$ and want to use it to improve the release of statistics of a smaller private dataset $\mathbf{x} \in \mathbb{R}^n$. To apply our quantile release method, we must use $\mathbf{x}'$ to construct a prior $\boldsymbol{\mu}'_{[i]}$ for each quantile q_i that makes $U_{\mathbf{x}}^{(q_i)}(\boldsymbol{\mu}'_{[i]})$ small. If the entries of $\mathbf{x}$ and $\mathbf{x}'$ are sampled i.i.d. from similar distributions $\mathcal{D}$ and $\mathcal{D}'$, respectively, the convexity of $U_{\mathbf{x}}^{(q_i)}$ suggests using stochastic optimization find priors $\boldsymbol{\mu}$ that approximately minimize the expectation $\mathbb{E}_{\mathbf{z}\sim\mathcal{D}'^n} U_{\mathbf{z}}(\boldsymbol{\mu})$ using samples of size n drawn from $\mathbf{x}'$. We provide a guarantee for a variant of this generic approach that runs online gradient descent (OGD) with separate learning rates for θ and ϕ on samples drawn without replacement from $\mathbf{x}'$:

Theorem 6.6.2 (c.f. Thm. 6.E.2)**.** If $\mathcal{D}$ and $\mathcal{D}'$ have bounded densities with bounded support then there exists an algorithm optimizing $U_{\mathbf{x}'_t}$ over T datasets $\mathbf{x}'_t$ of size n drawn from $\mathbf{x}' \in \mathbb{R}^N$ without replacement that runs in time $\mathcal{O}(mN)$ and returns a set μ' of m Laplace priors s.t. w.h.p.

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}^n} U_{\mathbf{x}}(\boldsymbol{\mu}') \leqslant \min_{\boldsymbol{\mu}\in \mathrm{Lap}^m_{B,\sigma_{\min},\sigma_{\max}}} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}^n} U_{\mathbf{x}}(\boldsymbol{\mu}) + \tilde{\mathcal{O}}\left(\mathrm{TV}_q(\mathcal{D},\mathcal{D}') + \sqrt{\frac{mn}{N}}\right) \tag{6.30}$$

where $\mathrm{Lap}_{B,\sigma_{\min},\sigma_{\max}}$ is the set of Laplace priors with locations in $[\pm B]$ and scales in $[\sigma_{\min}, \sigma_{\max}]$ and $\mathrm{TV}_q(\mathcal{D},\mathcal{D}')$ is the total variation distance between the joint distributions of the order statistics $\left\{(\mathbf{x}_{[\lfloor q_i n\rfloor]}, \mathbf{x}_{[\lfloor q_i n\rfloor+1]})\right\}_{i=1}^m$ for $\mathbf{x}\sim\mathcal{D}^n$ and $\left\{(\mathbf{x}'_{[\lfloor q_i n\rfloor]}, \mathbf{x}'_{[\lfloor q_i n\rfloor+1]})\right\}_{i=1}^m$ for $\mathbf{x}'\sim\mathcal{D}'^n$.

For $N \gg mn$, the suboptimality of μ' for the upper bound $U_{\mathbf{x}}$ will depend on the statistical distance between the quantile intervals of $\mathcal{D}$ and $\mathcal{D}'$: even if $\mathcal{D}$ and $\mathcal{D}'$ are dissimilar, similar order statistic distributions will ensure good performance. Note, as in Section 6.4, we can hedge against large $\mathrm{TV}_q(\mathcal{D},\mathcal{D}')$ by mixing the output μ' with a robust prior.

We evaluate this approach, which we call *Public Fit* or `PubFit`, on Adult [Kohavi, 1996] and Goodreads [Wan and McAuley, 2018], both used previously for DP quantiles [Gillenwater et al., 2021, Kaplan et al., 2022]. Because our guarantees improve with different step-sizes for θ and ϕ, we use COCOB [Orabona and Tomassi, 2017]—an OGD variant that provably sets per-coordinate step-sizes without the need for tuning—as `PubFit`'s stochastic solver. We also test a robust version where its output is mixed with a half-Cauchy distribution, and three baselines: the uniform prior, just using the quantiles of the public data (`public quantiles`), and using the public quantiles to set the location parameters of m Cauchy priors (`public Cauchy`).

Adult tests the $\mathcal{D} = \mathcal{D}'$ case, with its "train" set the public dataset and a hundred samples from "test" as private. Figure 6.2 shows that `public quantiles` does best at small ε, as is expected with no distribution shift, but it cannot adapt to the empirical distribution of a small number of private points, and so is worse at $\varepsilon > 1$. Among the rest, `PubFit` is most similar to `public quantiles` at small ε but still does well at large ε.

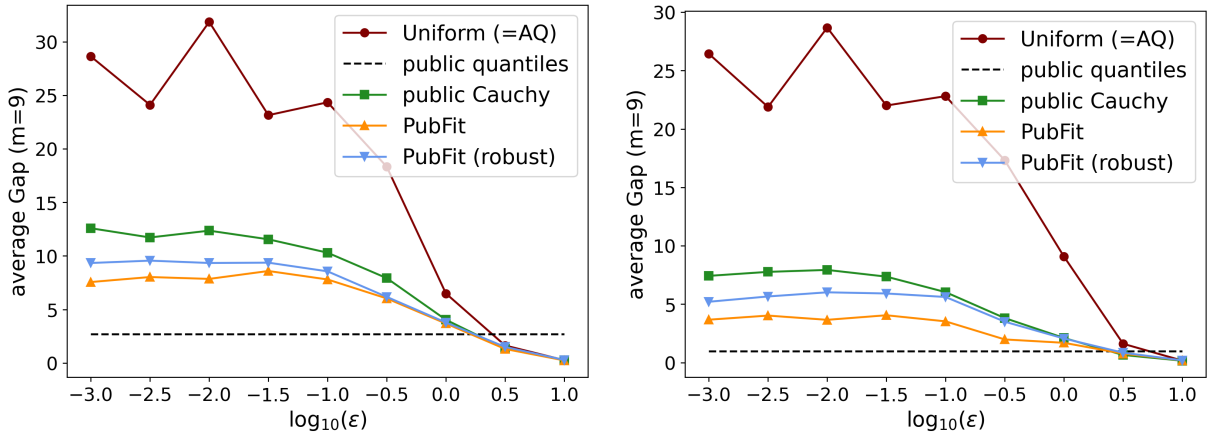

Figure 6.2: Public-private release of nine quantiles using 100 samples from the Adult age (left) and hours (right) datasets. The public data is the Adult training set while private data is test.

We use the Goodreads “History" and “Poetry" genres to evaluate under distribution shift by fitting on all but a small fraction of data from the former and releasing quantiles of samples from varying mixtures of the two datasets. As expected, the performance of `public quantiles` deteriorates with more samples from “Poetry." For book ratings, `PubFit` is best among the remaining methods, but without much change with distribution shift, possibly due to an incomplete fit of the data. For page counts, the `PubFit` methods and `public Cauchy` both do as well as `public-quantiles` when most data is from “History," but `PubFit (robust)` deteriorates least—and much less than regular `PubFit`—as the distribution shifts. This highlights the importance of robustness analysis, and suggest the former as a good method to start with, as it takes advantage of similar public and private distributions (Figure 6.2) while never doing much worse than the default method (uniform) when the the distributions are dissimilar (Figure 6.3).

6.6.3 Sequentially setting priors using past sensitive data

Our second application is sequential release, which we do not believe has been studied, but arises naturally if e.g. we wish to release daily statistics from a continuous stream of data. Here we have a sequence of datasets $\mathbf{x}_1, \dots, \mathbf{x}_T$, each with associated *public* features $\mathbf{f}_1, \dots, \mathbf{f}_T \in \mathbb{R}^d$ (e.g. day of the week), and we wish to minimize the average maximum gap $\frac{1}{T}\sum_{t=1}^{T} \max_i \mathrm{Gap}_{q_i}(\mathbf{x}_t, o_{t,i})$, whose expectation can be bounded (6.29) in terms of $\frac{1}{T}\sum_{t=1}^{T} U_{\mathbf{x}_t}$. For simplicity, we assume individuals do not occur in multiple datasets $\mathbf{x}_t$, e.g. we are releasing the median age of new users of a service. Note the natural way to avoid this assumption is to compose the privacy budgets at each time; empirically our methods are especially useful in the low privacy regime this entails.

Our analysis suggests that we can apply online learning here, e.g. doing the following at each t starting with a prior μ_1:

1. release o_t using the prior μ_t and suffer $\mathrm{Gap}_q(\mathbf{x}_t, o_t)$
2. update to μ_{t+1} using online learning on the loss $\ell_{\mathbf{x}_t}^{(q)}$

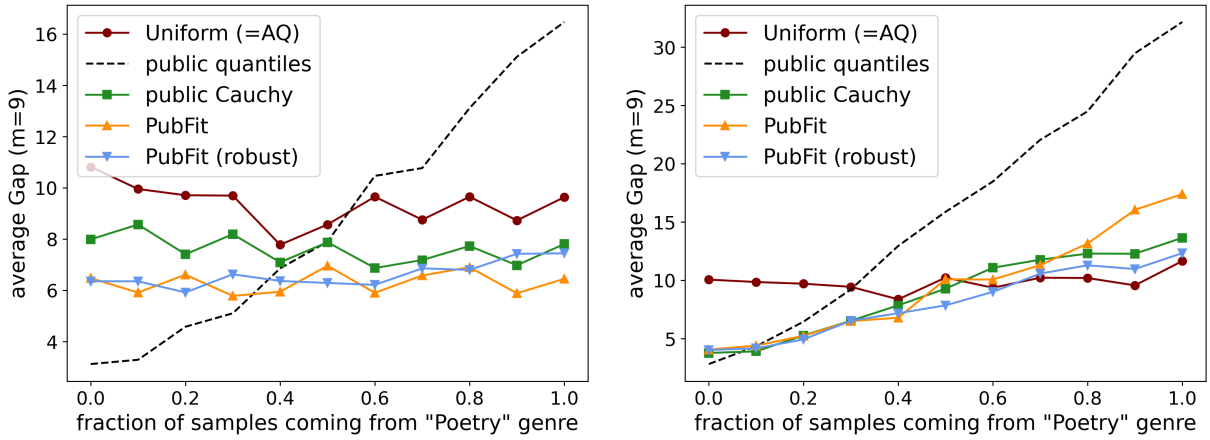

Figure 6.3: Public-private release of nine quantiles on one hundred samples from the Goodreads rating (left) and page count (right) datasets, with $\varepsilon = 1$. The public data is the "History" genre while private data is sampled from a mixture of it and "Poetry."

Because $\ell^{(q)}_{\mathbf{x}_t}(\theta, \phi) = U^{(q)}_{\mathbf{x}_t}(\mu_{\frac{\theta}{\phi}, \frac{1}{\phi}})$ is convex for Laplace priors, online convex optimization (OCO) lets us compete with the best prior in hindsight according to the upper bounds $U^{(q)}_{\mathbf{x}_t}(\mu_t)$, or with the best linear map $\mathbf{w}$ to locations $\langle \mathbf{w}, \mathbf{f}_t \rangle$. We can again hedge against poor predictions by mixing with a constant robust distribution.

However, we face the difficulty that online learning on losses $\ell^{(q)}_{\mathbf{x}_t}$ leaks information about $\mathbf{x}_t$. There are two natural solutions. One is to use part of the budget $\varepsilon' < \varepsilon$ on a DP online learner [Jain et al., 2012, Smith and Thakurta, 2013] and hope that the reduction in budget allocated to quantile release is made up for by the improved priors. Alternatively, we can replace ℓ with a *proxy* loss $\hat{\ell}$ that does not depend on the data and optimize it using regular OCO. The first can be done with provable guarantees by applying DP-FTRL [Kairouz et al., 2021a], again using two different step-sizes:

Theorem 6.6.3 (c.f. Thm. 6.E.3)**.** Consider a sequence of datasets $\mathbf{x}_t \in [\pm B]^{n_t}$ with bounded features $\mathbf{f}_t$ and suppose we set Laplace priors $\boldsymbol{\mu}_{t[i]} = \mu_{\frac{\langle \mathbf{v}_{t[i]}, \mathbf{f}_t \rangle}{\phi_{t[i]}}, \frac{1}{\phi_{t[i]}}}$ via two DP-FTRL algorithms applied separately to the variables $\mathbf{V}$ (an $m \times d$ matrix of m d-dimensional linear maps) and $\boldsymbol{\phi}$ (an m-vector of inverse scale parameters) passed to the losses $\ell_{\mathbf{x}_t}(\langle \mathbf{V}_{[i]}, \mathbf{f}_t \rangle, \boldsymbol{\phi}_{[i]})$ with budgets $\frac{\varepsilon'}{2}$, with respective step-sizes $\tilde{\Theta}\left(\sqrt{\frac{\varepsilon'}{\sigma^2_{\min} T}\sqrt{\frac{m}{d}}}\right)$ and $\tilde{\Theta}\left(\sqrt{\frac{\varepsilon'\sqrt{m}}{\sigma^2_{\min}\sigma^2_{\max} T}}\right)$. This is (ε', δ')-DP and w.h.p. has regret

$$\frac{1}{T}\sum_{t=1}^{T} U_{\mathbf{x}_t}(\boldsymbol{\mu}_t) - \min_{\substack{\mathbf{W} \in [\pm B]^{m \times d} \\ \boldsymbol{\sigma} \in [\sigma_{\min}, \sigma_{\max}]^m}} \frac{1}{T}\sum_{t=1}^{T} U_{\mathbf{x}_t}(\mu_{\langle \mathbf{W}_{[i]}, \mathbf{f}_t \rangle, \boldsymbol{\sigma}_{[i]}}) = \tilde{\mathcal{O}}\left(\frac{d^{\frac{3}{4}} + \sigma_{\max}}{\sigma_{\min}}\sqrt{\frac{m}{\varepsilon' T}\sqrt{m \log \frac{2}{\delta'}}}\right) \tag{6.31}$$

Thus we can do as well as any sequence of Laplace priors μ_t with locations determined by a fixed linear map from $\mathbf{f}_t$, up to a term that decreases at rate $\tilde{\mathcal{O}}(\frac{1}{\sqrt{T}})$. Furthermore, running quantile release with budget $\varepsilon - \varepsilon'$ ensures (ε, δ')-DP for each dataset $\mathbf{x}_t$. Note that using different

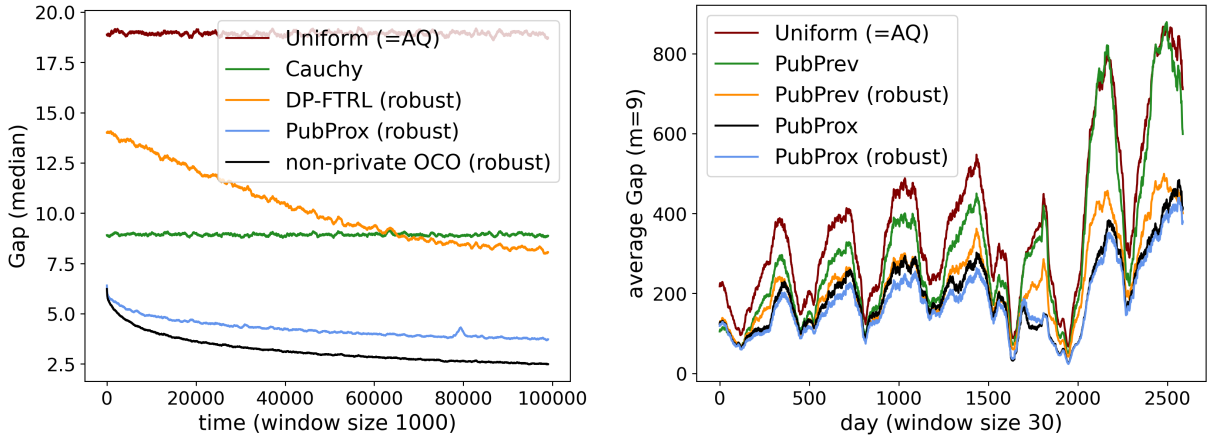

Figure 6.4: Comparison of sequential release over time on Synthetic (left, $\log_{10}\varepsilon = -1/2$) and CitiBike (right, $\log_{10}\varepsilon = -2$) tasks.

step-sizes allows us to separate the difficulty of learning a d-dimensional linear map from the difficulty of learning a scale parameter of magnitude at most $\sigma_{\max}$.

Unfortunately, DP-FTRL is too noisy to learn competitive priors, except with a lot of stationary data (c.f. Figure 6.4 (left)). One issue is that its DP guarantee is too strong, as it it allows swapping out the entire dataset $\mathbf{x}_t$ rather than a single entry. It is unclear if a better sensitivity is possible for $U_{\mathbf{x}_t}$, as changing an entry can flip the sign of the gradient while preserving magnitude. We show (c.f. Lemma 6.5.1) that it is possible for the ε-dependent bound $U^{(\varepsilon)}_{\mathbf{x}_t}$ over piecewise-constant priors—remarkably sensitivity *decreases* with ε—but that upper bound is nonconvex for location-scale families, which are preferable for model learning.

Our second solution involves recognizing that $U^{(q)}_{\mathbf{x}_t}$ depends only on the optimal interval $\left[\mathbf{x}_{t[\lfloor qn \rfloor]}, \mathbf{x}_{t[\lfloor qn \rfloor+1]}\right)$, whose location and size we have (public) estimates for: the former via the quantile estimate o_t and the size is lower-bounded by the underlying data discretization, which we have access to in practice (e.g. age is reported in years, bicycle trip length in seconds). We use this information to construct *proxy* losses $\hat{\ell}^{(q)}_{o_t}(\langle \mathbf{v}, \mathbf{f}_t \rangle, \phi)$, which do not depend on $\mathbf{x}_t$ and so be learned with (standard) OCO. As our DP-FTRL analysis again showed the importance of different step-sizes, we again use the COCOB optimizer here.

We evaluate sequential release on three online tasks, each consisting of a sequence of datasets needing quantiles:

1. Synthetic: each dataset is generated such that the quantiles are fixed linear functions of a random Gaussian feature vector, plus noise.
2. CitiBike: the data are the lengths of a day's bicycle trips, with the date and NYC weather information features.
3. BBC: the data are the Flesch readability scores of the comments on a headline posted to Reddit's `worldnews` forum, with date and headline text information features.

In addition to the proxy approach, which we call `PubProx`, we evaluate static priors—the uniform, Cauchy, and half-Cauchy (if nonnegative)—and an approach we call `PubPrev`, which uses a Laplace prior centered around the previous step's released quantile. Note that using the

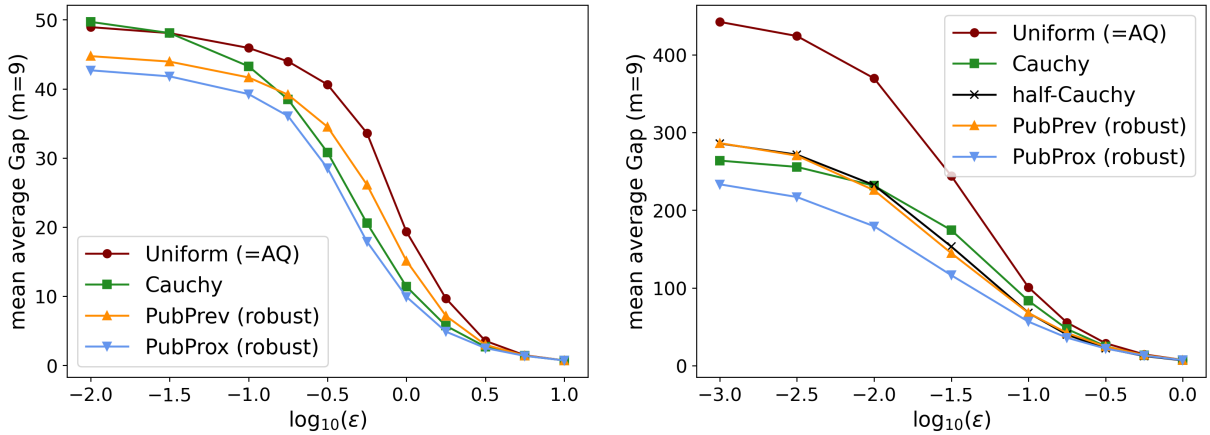

Figure 6.5: Time-averaged performance of the sequential release of nine quantiles on the Synthetic (left) and CitiBike (right) tasks.

uniform prior is equivalent to ApproximateQuantiles (AQ). For both `PubProx` and `PubPrev` we ensure robustness by mixing with a Cauchy (or half-Cauchy, if nonnegative) distribution with coefficient 0.1; this nearly always improves performance for these methods, likely by ensuring their training data is not too noisy. To see its effectiveness, note how in Figure 6.4 (right) both augmented methods are almost always better when made robust, especially `PubPrev`; in fact, non-robust `PubPrev` is unable to do better than AQ after around day 1600, when the start of the COVID-19 pandemic significantly affects bicycle trips.

Our main comparisons is time-aggregated performance as a function of ε (c.f. Figs. 6.5 and 6.6). All except perhaps Synthetic demonstrate significant improvement by our methods over the uniform (AQ) baseline, especially at small ε. On Synthetic and CitiBike, both tasks with features for which a linear model should provide some benefit, we see in Figure 6.5 that `PubProx` is indeed the best across all except perhaps the lowest privacy settings. For BBC, Figure 6.6 reveals a large difference between mean and median performance (note the difference in y-axis scales), with `PubProx` doing best for the typical headline but the Cauchy doing better on-average due to better performance on headlines with many comments. The result suggests that in highly noisy settings, the learning-based scheme should help, but it might not overcome the robustness of a static Cauchy prior in-expectation.

Overall, the results demonstrate the strength of the Cauchy and half-Cauchy priors, both as unbounded substitutes for the uniform and as a means of robustifying learning-augmented algorithms. They also demonstrate the utility of our upper bound in providing an objective for learning, albeit using proxy data rather the DP online learning: `PubProx` usually does better than `PubPrev` despite using the same information. Overall, `PubProx` performs the best at most privacy levels in all evaluation settings (Synthetic, CitiBike, and BBC) except when the mean is used as the metric for BBC (Figure 6.6, left), where it does almost as well as the best. Narrowing the performance gap with non-private OCO (c.f. Figure 6.4 (left), where we run COCOB directly on $\ell_{\mathbf{x}_t}^{(q)}$)—remains an important research direction.

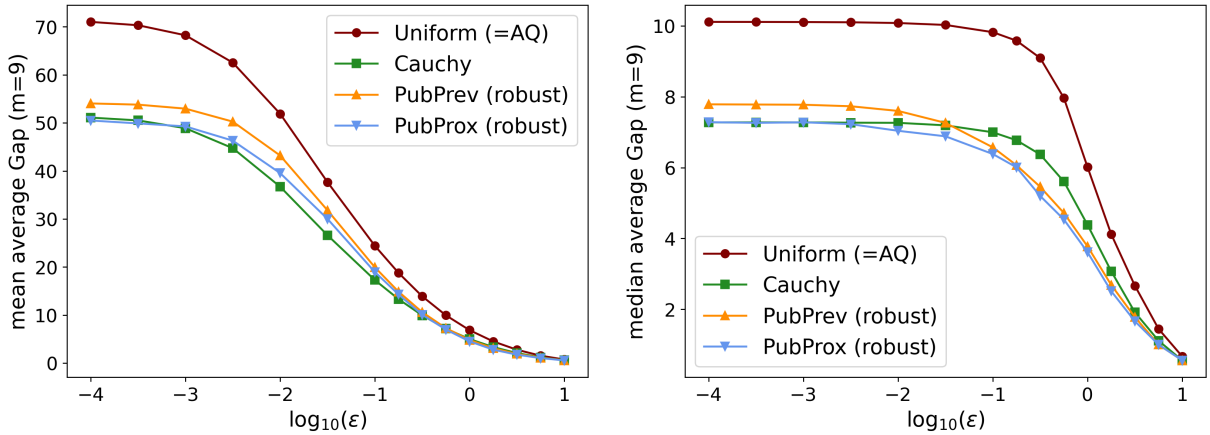

Figure 6.6: Time-aggregated mean (left) and median (right) performance of sequential release of nine quantiles on the BBC task.

6.7 Conclusion

This chapter extends the algorithms with predictions setup to DP methods and provides extensive evidence of its utility as a way of integrating external information into privacy-preserving algorithms. In particular, we show how it informs the design of methods that are robust to poor predictions and of learning algorithms for obtaining good predictions from data. Finally, we demonstrate how combining optimization with learning-augmented private algorithms can be used to significantly improve the quality of released statistics in practice. As a result, we believe these methods hold great promise for reducing error while preserving privacy on practical, real-world problems.

Beyond the current work, we believe this way of studying DP methods is highly applicable and will see a great deal of future work in finding new applications for incorporating predictions or improving the approaches described here. By conducting a fine-grained analysis of DP algorithms beyond their default parameterizations, it also is highly likely to lead to significant contributions even in the prediction-free setting, as exemplified by our guarantees for unbounded-domain quantile release and improved bounds for trace-sensitive covariance estimation. Some specific areas to explore include other forms of iterative data analysis beyond MWEM [Gupta et al., 2012, Hardt and Rothblum, 2010] and other important dataset statistics [Biswas et al., 2020]. Another important direction is that of learning: can DP online convex optimization be made useful for the purpose of learning predictions, or can guarantees be shown for our alternative of using public proxies?

6.A Quantile release

6.A.1 Section 6.3.1 details

The base measure μ of DP mechanisms such as the exponential is the starting point of many approaches to incorporating external information, especially ones focused on Bayesian posterior sampling [Dimitrakakis et al., 2017, Geumlek et al., 2017, Seeman et al., 2020]; while it is also our approach to single-quantile estimation with predictions, a key difference here is the focus on utility guarantees depending on both the prediction and instance, which is missing from this past work. In the quantile problem, given a quantile q and a sorted dataset $\mathbf{x} \in \mathbb{R}^n$ of n distinct points, the goal is to release a number o that upper bounds exactly $\lfloor qn \rfloor$ of the entries. A natural error metric, $\mathrm{Gap}_q(\mathbf{x}, o)$, is the number of entries between the released number o and $\lfloor qn \rfloor$, and we can show that prediction-dependent bound using astraightforward application of EM with utility $-\mathrm{Gap}_q$:

Lemma 6.A.1. Releasing $o \in \mathbb{R}$ w.p. $\propto \exp(-\varepsilon \mathrm{Gap}_q(\mathbf{x}, o)/2)\mu(o)$ is ε-DP, and w.p. $1 - \beta$

$$\mathrm{Gap}_q(\mathbf{x}, o) \leqslant \frac{2}{\varepsilon}\left(\log\frac{1}{\beta} - \log \Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu)\right) \leqslant \frac{2}{\varepsilon}\left(\log\frac{1}{\beta} - \log \Psi_{\mathbf{x}}^{(q)}(\mu)\right) \tag{6.32}$$

where $\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu) = \sum_{i=0}^{n} \exp(-\varepsilon \mathrm{Gap}_q(\mathbf{x}, I_i)/2)\mu(I_i) = \int \exp(-\varepsilon \mathrm{Gap}_q(\mathbf{x}, o)/2)\mu(o)do$ is the inner product between μ and the exponential score while $\Psi_{\mathbf{x}}^{(q)}(\mu) = \mu(I_{\lfloor qn \rfloor})$ is the measure of the optimal interval (note $\max_k u_q(\mathbf{x}, I_k) = -\mathrm{Gap}_q(\mathbf{x}, I_{\lfloor qn \rfloor}) = 0$ and so $\Psi_{\mathbf{x}}^{(q)}(\mu) \leqslant \Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu)\ \forall\ \varepsilon > 0$).

Proof. ε-DP follows from u_q having sensitivity one and the guarantee of EM with base measure μ [McSherry and Talwar, 2007, Theorem 6]. For the error, since we sample an interval I_k and then sample $o \in I_k$ we have

$$\begin{aligned}\Pr(\mathrm{Gap}_q(\mathbf{x}, o) \geqslant \gamma) = \Pr(u_q(\mathbf{x}, I_k) \leqslant -\gamma) &= \sum_{j=0}^{n} \Pr(k = j) 1_{u_q(\mathbf{x}, I_j) \leqslant -\gamma} \\ &\leqslant \sum_{j=0}^{n} \frac{\exp(-\frac{\varepsilon\gamma}{2})\mu(I_j)}{\sum_{i=0}^{n} \exp(\frac{\varepsilon}{2} u_q(\mathbf{x}, I_i))\mu(I_i)} \leqslant \frac{\exp(-\frac{\varepsilon\gamma}{2})}{\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu)}\end{aligned} \tag{6.33}$$

The result follows by substituting β for the failure probability and solving for γ. □

We can also analyze the error metrics in this bound for specific measures μ. In particular, if the points are in a bounded interval (a, b) and we use the uniform measure $\mu(o) = 1_{o \in (a,b)}/(b-a)$ then $\Psi_{\mathbf{x}}^{(q,\varepsilon)}(\mu) \geqslant \frac{\psi_{\mathbf{x}}}{b-a}$, where $\psi_{\mathbf{x}} = \min_k \mathbf{x}_{[k+1]} - \mathbf{x}_{[k]}$, and we exactly recover the standard bound of $\frac{2}{\varepsilon} \log \frac{b-a}{\beta\psi_{\mathbf{x}}}$, e.g. the one in Kaplan et al. [2022, Lemma A.1] (indeed their analysis implicitly uses this measure). However, our approach also allows us to remove the boundedness assumption, which itself can be viewed as a type of prediction, as one needs external information to assume that the data, or at least the quantile, lies within the interval (a, b). Taking this view,

we can use the prediction to set the location $\nu \in \mathbb{R}$ and scale $\sigma > 0$ of a Cauchy prior $\mu_{\nu,\sigma}(o) = \sigma/(\pi(\sigma^2 + (o-\nu)^2))$ without committing to (a, b) actually containing the data. Since we know that the optimal interval $(\mathbf{x}_{[\lfloor qn \rfloor]}, \mathbf{x}_{[\lfloor qn \rfloor+1]}]$ is a subset of $(\frac{a+b}{2} \pm R)$ for some $R > 0$, setting $\nu = \frac{a+b}{2}$ and $\sigma = \frac{b-a}{2}$ yields

$$\Psi_{\mathbf{x}}^{(q)}(\mu_{\nu,\sigma}) \geqslant \frac{\sigma}{\pi} \frac{\mathbf{x}_{[\lfloor qn \rfloor+1]} - \mathbf{x}_{[\lfloor qn \rfloor]}}{\sigma^2 + \max_{k \in \{\lfloor qn \rfloor, \lfloor qn \rfloor+1\}} (\nu - \mathbf{x}_{[k]})^2} \geqslant \frac{\sigma}{\pi} \min_k \frac{\mathbf{x}_{[k+1]} - \mathbf{x}_{[k]}}{\sigma^2 + R^2} \geqslant \frac{2(b-a)\psi_{\mathbf{x}}/\pi}{(b-a)^2 + 4R^2} \tag{6.34}$$

If $R = \frac{b-a}{2}$, i.e. we get the interval containing the data correct, then substituting the above into Lemma 6.A.1 recovers the guarantee of the uniform prior up to an additive factor $\frac{2}{\varepsilon} \log \pi$. However, whereas for the uniform prior we have no performance guarantees if the interval is incorrect, using the Cauchy prior the performance degrades gracefully as the error (R) grows. While this first result can be viewed as designing a better prediction-free algorithm, it can also be viewed as making more robust use of the external information about the interval containing the data.

Multiple quantile release using multiple priors

To estimate $m > 1$ quantiles $q_1, \ldots, q_m$ at once, we adapt the recursive approach of Kaplan et al. [2022], whose method ApproximateQuantiles implicitly constructs a binary tree with a quantile q_i at each node and uses the exponential mechanism to compute the quantile $\tilde{q}_i = (q_i - \underline{q}_i)/(\overline{q}_i - \underline{q}_i)$ of the dataset $\hat{\mathbf{x}}_i$ of points in the original dataset $\mathbf{x}$ restricted to the interval $(\hat{a}_i, \hat{b}_i)$; here $\underline{q}_i < q_i$ and $\overline{q}_i > q_i$ are quantiles appearing earlier in the tree whose respective estimates $\hat{a}_i$ and $\hat{b}_i$ determine the sub-interval (if there is no earlier quantile on the left and/or right of q_i we use $\underline{q}_i = 0, \hat{a}_i = a$ and/or $\overline{q}_i = 1, \hat{b}_i = b$). Because each datapoint only participates in $\mathcal{O}(\log_2 m)$ exponential mechanisms, the approach is able to run each mechanism with budget $\Omega(\varepsilon/\log_2 m)$ and thus only suffer error logarithmic in the number of quantiles m, a significant improvement upon running one EM with budget ε/m on the entire dataset for each quantile, which has error $\mathcal{O}(m)$ in the number of quantiles.

We can apply prior-dependent guarantees to ApproximateQuantiles—pseudocode for a generalized version of which is provided in Algorithm 13—by recognizing that implicitly the method assigns a uniform prior μ_i to each quantile q_i and then running EM with the *conditional* prior $\hat{\mu}_i$ restricted to the interval $[\hat{a}_i, \hat{b}_i]$ determined by earlier quantiles in the binary tree. An extension of the argument in Equation 6.33 (c.f. Lemma 6.A.2) then yields a bound on the error of the estimate o_i returned for quantile q_i in terms of the prior-EM inner-product computed with this conditional prior $\hat{\mu}_i$ over the subset $\hat{\mathbf{x}}_i$:

$$\Pr(\mathrm{Gap}_{q_i}(\mathbf{x}, o_i) \geqslant \gamma) \leqslant \frac{\exp\left(\frac{\varepsilon_i}{2}(\hat{\gamma}_i - \gamma)\right)}{\Psi_{\hat{\mathbf{x}}_i}^{(\tilde{q}_i, \varepsilon_i)}(\hat{\mu}_i)} \quad \text{for} \quad \hat{\gamma}_i = (1 - \tilde{q}_i)\,\mathrm{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \tilde{q}_i\,\mathrm{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i) \tag{6.35}$$

Note that the error is offset by a weighted combination $\hat{\gamma}_i$ of the errors of the estimates of quantiles earlier in the tree. Controlling this error allows us to bound the maximum error of any quantile via the harmonic mean of the inner products between the exponential scores and conditional priors:

Lemma 6.A.2. Algorithm 13 with $K = 2$ and $\varepsilon_i = \varepsilon/\lceil \log_2 m \rceil \ \forall\ i$ is ε-DP and w.p. $\geqslant 1 - \beta$ has

$$\max_i \operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2}{\varepsilon}\lceil \log_2 m \rceil^2 \log \frac{m}{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}} \qquad \text{for} \qquad \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)} = \left(\sum_{i=1}^{m} \frac{1/m}{\Psi_{\hat{\mathbf{x}}_i}^{(\tilde{q}_i, \varepsilon_i)}(\hat{\mu}_i)} \right)^{-1} \tag{6.36}$$

Proof. The privacy guarantee follows as in Kaplan et al. [2022, Lemma 3.1]. Setting the above probability bound (6.35) to $\frac{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}}{m \Psi_{\hat{\mathbf{x}}_i}^{(\tilde{q}_i, \varepsilon_i)}(\hat{\mu}_i)}$ for each i we have w.p. $\geqslant 1 - \beta$ that $\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2}{\bar{\varepsilon}} \log \frac{m}{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}} + \hat{\gamma}_i \ \forall\ i$. Now let k_i be the depth of quantile q_i in the tree. If $k_i = 1$ then i is the root node so $\hat{\gamma}_i = 0$ and we have $\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2}{\bar{\varepsilon}} \log \frac{m}{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}}$. To make an inductive argument, we assume $\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2k}{\bar{\varepsilon}} \log \frac{m}{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}} \ \forall\ i$ s.t. $k_i \leqslant k$, and so for any i s.t. $k_i = k + 1$ we have that

$$\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2}{\bar{\varepsilon}} \log \frac{m}{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}} + (1 - \tilde{q}_i) \operatorname{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \tilde{q}_i \operatorname{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i) \leqslant \frac{2(k+1)}{\bar{\varepsilon}} \log \frac{m}{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}} \tag{6.37}$$

Thus $\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2k_i}{\bar{\varepsilon}} \log \frac{m}{\beta \hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}} \ \forall\ i$, so using $k_i \leqslant \lceil \log_2 m \rceil$ and $\bar{\varepsilon} = \frac{\varepsilon}{\lceil \log_2 m \rceil}$ yields the result. □

Setting $\hat{\mu}_i$ to be uniform on $[\hat{a}_i, \hat{b}_i]$ exactly recovers both the algorithm and guarantee of Kaplan et al. [2022, Theorem 3.3]. As before, we can also extend the algorithm to the infinite interval:

Corollary 6.A.1. If all priors are Cauchy with location $\frac{a+b}{2}$ and scale $\frac{b-a}{2}$ and the data lies in the interval $(\frac{a+b}{2} \pm R)$ then w.p. $\geqslant 1 - \beta$ the maximum error is at most $\frac{2}{\varepsilon}\lceil \log_2 m \rceil^2 \log \left(\pi m \frac{b - a + \frac{4R^2}{b-a}}{2\beta \psi_{\mathbf{x}}} \right)$.

However, while this demonstrates the usefulness of Lemma 6.A.2 for obtaining robust priors on infinite intervals, the associated prediction measure $\hat{\Psi}_{\mathbf{x}}^{(\varepsilon)}$ is imperfect because it is non-deterministic: its value depends on the random execution of the algorithm, specifically on the data subsets $\hat{\mathbf{x}}_i$ and priors $\hat{\mu}_i$, which for i not at the root of the tree are affected by the DP mechanisms of i's ancestor nodes. In addition to not being given fully specified by the prediction and data, this makes $\hat{\Psi}^{(\varepsilon)}$ difficult to use as an objective for learning. A natural more desirable prediction metric is the harmonic mean of the inner products between the exponential scores and *original* priors μ_i over the *original* dataset $\mathbf{x}$, i.e. the direct generalization of our approach for single quantiles.

Unfortunately, the conditional restriction of μ_i to the interval $[\hat{a}_i, \hat{b}_i]$ removes the influence of probabilities assigned to intervals between points *not* in this interval. To solve this, we propose a different *edge*-restriction of μ_i that assigns probabilities $\mu_i((-\infty, \hat{a}_i))$ and $\mu_i((\hat{b}_i, \infty))$ of being outside the interval $[\hat{a}_i, \hat{b}_i]$ to atoms on its edges $\hat{a}_i$ and $\hat{b}_i$, respectively. Despite not using any information from points outside $\hat{\mathbf{x}}_i$, this approach puts probabilities assigned to intervals outside $[\hat{a}_i, \hat{b}_i]$ to the edge closest to them, allowing us to extend the previous probability bound (6.35) to depend on the original prior-EM inner-product (c.f. Lemma 6.A.5):

$$\Pr(\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \geqslant \gamma) \leqslant \exp(\varepsilon(\hat{\gamma}_i - \gamma/2)) / \Psi_{\mathbf{x}}^{(q_i, \varepsilon_i)}(\mu_i) \tag{6.38}$$

However, the stronger dependence of this bound on errors $\hat{\gamma}_i$ earlier in the tree lead to an $\tilde{\mathcal{O}}(\phi^{\log_2 m}) = \mathcal{O}(m^{0.7})$ dependence on m, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio:

Theorem 6.A.1. If the quantiles are uniform negative powers of two then Algorithm 13 with $K = 2$, edge-based prior adaptation, and $\varepsilon_i = \varepsilon/\lceil \log_2(m+1) \rceil \; \forall \; i$ is ε-DP and w.p. $\geqslant 1 - \beta$ has

$$\max_i \operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2}{\varepsilon} \phi^{\log_2(m+1)} \lceil \log_2(m+1) \rceil \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \quad \text{for} \quad \Psi_{\mathbf{x}}^{(\varepsilon)} = \left(\sum_{i=1}^{m} \frac{1/m}{\Psi_{\mathbf{x}}^{(q_i, \varepsilon_i)}(\mu_i)} \right)^{-1} \tag{6.39}$$

Proof. Since $\tilde{q}_i = 1/2 \; \forall \; i$, setting the new probability bound equal to $\frac{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}}{m \Psi_{\mathbf{x}}^{(q_i \varepsilon_i)}(\mu_i)}$ yields that w.p. $\geqslant 1 - \beta$

$$\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2}{\bar{\varepsilon}} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} + 2\hat{\gamma}_i = \frac{2}{\bar{\varepsilon}} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} + \operatorname{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \operatorname{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i) \; \forall \; i \tag{6.40}$$

If for each $k \leqslant \lceil \log_2 m \rceil$ we define E_k to be the maximum error of any quantile of at most depth k in the tree then since one of $\underline{q}_i$ and $\overline{q}_i$ is at depth at least one less than q_i and the other is at depth at least two less than q_i we have $E_k \leqslant \frac{2A_k}{\bar{\varepsilon}} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}}$ for recurrent relation $A_k = 1 + A_{k-1} + A_{k-2}$ with $A_0 = 0$ and $A_1 = 1$. Since $A_k = F_{k+1} - 1$ for Fibonacci sequence $F_j = \frac{\phi^j - (1-\phi)^j}{\sqrt{5}}$, we have

$$\begin{aligned} \max_i \operatorname{Gap}_{q_i}(\mathbf{x}, o_i) = \max_k E_k &\leqslant \frac{2\phi^{\lceil \log_2(m+1) \rceil + 1}}{\bar{\varepsilon}\sqrt{5}} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \\ &= \frac{2\phi^{\lceil \log_2(m+1) \rceil + 1}}{\varepsilon \sqrt{5}} \lceil \log_2(m+1) \rceil \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \end{aligned} \tag{6.41}$$

□

Thus while we have obtained a performance guarantee depending only on the prediction and the data via the harmonic mean $\Psi_{\mathbf{x}}^{(\varepsilon)}$ of the true prior-EM inner-products, the dependence on m is now polynomial. Note that it is still sublinear, which means it is better than the naive baseline of running m independent exponential mechanisms. Still, we can do much better—in-fact asymptotically better than any power of m—by recognizing that the main issue is the compounding error induced by successive errors to the boundaries of sub-intervals. We can reduce this by reducing the depth of the tree using a K-ary rather than binary tree and instead paying $K - 1$ times the privacy budget at each depth in order to naively release values for $K - 1$ quantiles. This can introduce out-of-order quantiles, but by Lemma 6.A.6 swapping any two out-of-order quantiles does not increase the maximum error and so this issue can be solved by sorting the $K - 1$ quantiles before using them to split the data. We thus have the following prediction-dependent performance bound for multiple quantiles:

Theorem 6.A.2. If we run Algorithm 13 with $K = \lceil \exp(\sqrt{\log 2 \log(m+1)}) \rceil$, edge-based adaptation, and $\varepsilon_i = \frac{\bar{\varepsilon}}{k_i^p}$ for some power $p > 1$, k_i the depth of q_i in the K-ary tree, and

$\bar{\varepsilon} = \frac{\varepsilon}{K-1}\left(\sum_{k=1}^{\lceil \log_K(m+1)\rceil} \frac{1}{k^p}\right)^{-1}$, then the result satisfies ε-DP and w.p. $\geqslant 1-\beta$ we have

$$\max_i \operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2\pi^2}{\varepsilon} \exp\left(2\sqrt{\log(2)\log(m+1)}\right) \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \tag{6.42}$$

if $p = 2$ and more generally

$$\max_i \operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{c_p}{\varepsilon} \exp\left(2\sqrt{\log(2)\log(m+1)}\right) \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \tag{6.43}$$

where c_p depends only on p.

Proof. The privacy guarantee follows as in [Kaplan et al., 2022, Lemma 3.1] except before each split we compute $K-1$ quantiles with $K-1$ times less budget. As in the previous proof, we have w.p. $\geqslant 1-\beta$ that

$$\operatorname{Gap}_{q_i}(\mathbf{x}, o_i) \leqslant \frac{2}{\varepsilon_i} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} + 2\hat{\gamma}_i = \frac{2k_i^2}{\bar{\varepsilon}} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} + 2(1-\tilde{q}_i) \operatorname{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + 2\tilde{q}_i \operatorname{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i) \; \forall \, i \tag{6.44}$$

If for each $k \leqslant \lceil \log_K(m+1) \rceil$ we define E_k to be the maximum error of any quantile of at most depth k in the tree then since both $\underline{q}_i$ and $\overline{q}_i$ are at depth at least one less than q_i we have $E_k \leqslant \frac{2A_k}{\bar{\varepsilon}} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}}$, where $A_k = k^p + 2A_{k-1}$ and $A_1 = 1$. For the case of $p = 2$, $A_k \leqslant 6 \cdot 2^k$ and $1/\bar{\varepsilon} = \frac{K-1}{\varepsilon} \sum_{k=1}^{\lceil \log_K(m+1) \rceil} \frac{1}{k^2} \leqslant \frac{\pi^2}{6\varepsilon}(K-1)$ so we have that

$$\max_i \operatorname{Gap}_{q_i}(\mathbf{x}, o_i) = \max_k E_k \leqslant \frac{12}{\bar{\varepsilon}} 2^{\lceil \log_K(m+1) \rceil} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \leqslant \frac{2\pi^2}{\varepsilon}(K-1) 2^{\lceil \log_K(m+1) \rceil} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \tag{6.45}$$

Substituting $K = \lceil \exp(\sqrt{\log 2 \log(m+1)}) \rceil$ and simplifying yields the result. For $p > 1$, $A_k \leqslant 2^{k-2}\left(2 + \Phi\left(\frac{1}{2}, -p, 2\right)\right)$, where Φ is the Lerch transcendent, and $1/\bar{\varepsilon} \leqslant \frac{K-1}{\varepsilon}\zeta(p)$, where ζ is the Riemann zeta function. Therefore

$$\begin{aligned} \max_i \operatorname{Gap}_{q_i}(\mathbf{x}, o_i) = \max_k E_k &\leqslant \frac{2^{\lceil \log_K(m+1) \rceil}}{2\bar{\varepsilon}} \left(2 + \Phi\left(\frac{1}{2}, -p, 2\right)\right) \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \\ &\leqslant \frac{c_p}{\varepsilon}(K-1) 2^{\lceil \log_K(m+1) \rceil} \log \frac{m}{\beta \Psi_{\mathbf{x}}^{(\varepsilon)}} \end{aligned} \tag{6.46}$$

for $c_p = \left(1 + \Phi\left(\frac{1}{2}, -p, 2\right)/2\right)\zeta(p)$. □

Similarly to Theorem 6.A.1, the proof establishes a recurrence relationship between the maximum errors at each depth. Note that in addition to the K-ary tree this bound uses depth-dependent budgeting to remove a $\mathcal{O}(\log_2 m)$-factor; the constant depending upon the parameter $p > 1$ of the latter has a minimum of roughly 8.42 at $p \approx 1.6$. As discussed before, the new dependence $\tilde{\mathcal{O}}\left(\exp\left(2\sqrt{\log(2)\log(m+1)}\right)\right)$ on m is sub-polynomial, i.e $o(m^\alpha) \; \forall \; \alpha > 0$. While it is also super-polylogarithmic, its shape for any practical value of m is roughly $\mathcal{O}(\log_2^2 m)$, making the result of interest as a justification for the negative log-inner-product performance metric.

Experimental details

For the experiments we report in Section 6.3.1, specifically Figure 6.3, we evaluate three variants of the algorithm on data drawn from a standard Gaussian distribution and from the Adult "age" dataset [Kohavi, 1996]. In both cases we use 1000 samples and run each experiment 40 times, reporting the average performance. As we do for all datasets, we use reasonable guesses of mean, scale, and bounds on each dataset to set priors. As in this section we report the uniform, we need to specify its range; for Gaussian we use $[-10, 10]$, while for "age" we use $[10, 120]$.

The original AQ algorithm of Kaplan et al. [2022] is now fully specified. We test two variants of our K-ary modification: one with edge-based adaptation, and the other using the original conditional adaptation. For both cases we set K as a function of m according to the formula in Theorem 6.3.3, and we set the power p of the depth-dependent budget discounting to 1.5, which is close to the theoretically optimal value of around 1.6 (c.f. Thm 6.A.2).

6.A.2 Additional proofs

Lemma 6.A.3. In Algorithm 13, for any $i \in [m]$ and $\hat{\gamma}_i = (1-\tilde{q}_i)\operatorname{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \tilde{q}_i \operatorname{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i)$ we have

1. $\operatorname{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, o) \leqslant \operatorname{Gap}_{q_i}(\mathbf{x}, o) + \hat{\gamma}_i \;\forall\, o \in \mathbb{R}$
2. $\operatorname{Gap}_{q_i}(\mathbf{x}, o) \leqslant \operatorname{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, o) + \hat{\gamma}_i \;\forall\, o \in [\hat{a}_i, \hat{b}_i]$

Proof. For $o \in [\hat{a}_i, \hat{b}_i]$ we apply the triangle inequality twice to get

$$\begin{aligned}
\operatorname{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, o) &= |\max_{\hat{\mathbf{x}}_{[j]}<o} j - \lfloor \tilde{q}_i \hat{n}_i \rfloor| \\
&= |\max_{\hat{\mathbf{x}}_{[j]}<o} j + \max_{\mathbf{x}_{[j]}<\hat{a}_i} j - \lfloor q_i n \rfloor + \lfloor q_i n \rfloor - \max_{\mathbf{x}_{[j]}<\hat{a}_i} j - \lfloor \tilde{q}_i \hat{n}_i \rfloor| \\
&\leqslant \operatorname{Gap}_{q_i}(\mathbf{x}, o) + \left| \lfloor \tilde{q}_i (\lfloor \overline{q}_i n \rfloor - \lfloor \underline{q}_i n \rfloor) \rfloor + \lfloor \underline{q}_i n \rfloor - \max_{\mathbf{x}_{[j]}<\hat{a}_i} j - \lfloor \tilde{q}_i (\max_{\mathbf{x}_{[j]}<\hat{b}_i} j - \max_{\mathbf{x}_{[j]}<\hat{a}_i} j) \rfloor \right| \\
&\leqslant \operatorname{Gap}_{q_i}(\mathbf{x}, o) + (1-\tilde{q}_i)\operatorname{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \tilde{q}_i \operatorname{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i)
\end{aligned} \tag{6.47}$$

and again to get

$$\begin{aligned}
\operatorname{Gap}_{q_i}(\mathbf{x}, o) &= |\max_{\mathbf{x}_{[j]}<o} j - \lfloor q_i n \rfloor| \\
&= |\max_{\hat{\mathbf{x}}_{[j]}<o} j + \max_{\mathbf{x}_{[j]}<\hat{a}_i} j - \lfloor \tilde{q}_i \hat{n}_i \rfloor + \lfloor \tilde{q}_i \hat{n}_i \rfloor - \lfloor q_i n \rfloor| \\
&\leqslant \operatorname{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, o) + \left| \max_{\mathbf{x}_{[j]}<\hat{a}_i} j - \lfloor \tilde{q}_i (\max_{\mathbf{x}_{[j]}<\hat{b}_i} j + \max_{\mathbf{x}_{[j]}<\hat{a}_i} j) \rfloor - \lfloor \tilde{q}_i (\lfloor \overline{q}_i n \rfloor - \lfloor \underline{q}_i n \rfloor) \rfloor - \lfloor \underline{q}_i n \rfloor \right| \\
&\leqslant \operatorname{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, o) + (1-\tilde{q}_i)\operatorname{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \tilde{q}_i \operatorname{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i)
\end{aligned} \tag{6.48}$$

For $o < \hat{a}_i$ we use the fact that $\max_{\mathbf{x}_{[j]}<o} j \leqslant \max_{\mathbf{x}_{[j]}<\hat{a}_i} j$ and the triangle inequality to get

$$\begin{aligned}
\mathrm{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, o) &= \lfloor \tilde{q}_i \hat{n}_i \rfloor \\
&= \lfloor \tilde{q}_i (\max_{\mathbf{x}_{[j]}<\hat{b}_i} j - \max_{\mathbf{x}_{[j]}<\hat{a}_i} j) \rfloor \\
&\leqslant \lfloor \tilde{q}_i \max_{\mathbf{x}_{[j]}<\hat{b}_i} j \rfloor + \lfloor (1-\tilde{q}_i) \max_{\mathbf{x}_{[j]}<\hat{a}_i} j \rfloor - \max_{\mathbf{x}_{[j]}<o} j \\
&= \lfloor \tilde{q}_i \max_{\mathbf{x}_{[j]}<\hat{b}_i} j \rfloor + \lfloor (1-\tilde{q}_i) \max_{\mathbf{x}_{[j]}<\hat{a}_i} j \rfloor - \max_{\mathbf{x}_{[j]}<o} j + \lfloor q_i n \rfloor | - \lfloor \tilde{q}_i (\lfloor \overline{q}_i n \rfloor - \lfloor \underline{q}_i n \rfloor) \rfloor - \lfloor \underline{q}_i n \rfloor \\
&\leqslant \mathrm{Gap}_{q_i}(\mathbf{x}, o) + (1-\tilde{q}_i)\,\mathrm{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \tilde{q}_i\,\mathrm{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i)
\end{aligned} \tag{6.49}$$

For $o > \hat{b}_i$ we use the fact that $\max_{\mathbf{x}_{[j]}<\hat{b}_i} j \leqslant \max_{\mathbf{x}_{[j]}<o} j$ and the triangle inequality to get

$$\begin{aligned}
\mathrm{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, o) &= \lfloor (1-\tilde{q}_i) \hat{n}_i \rfloor \\
&= \lfloor (1-\tilde{q}_i) (\max_{\mathbf{x}_{[j]}<\hat{b}_i} j - \max_{\mathbf{x}_{[j]}<\hat{a}_i} j) \rfloor \\
&\leqslant \max_{\mathbf{x}_{[j]}<o} j - \lfloor \tilde{q}_i \max_{\mathbf{x}_{[j]}<\hat{b}_i} j - \lfloor (1-\tilde{q}_i) \max_{\mathbf{x}_{[j]}<\hat{a}_i} j \\
&= \max_{\mathbf{x}_{[j]}<o} j - \lfloor \tilde{q}_i \max_{\mathbf{x}_{[j]}<\hat{b}_i} j - \lfloor (1-\tilde{q}_i) \max_{\mathbf{x}_{[j]}<\hat{a}_i} j - \lfloor q_i n \rfloor + \lfloor \tilde{q}_i (\lfloor \overline{q}_i n \rfloor - \lfloor \underline{q}_i n \rfloor) \rfloor + \lfloor \underline{q}_i n \rfloor \\
&\leqslant \mathrm{Gap}_{q_i}(\mathbf{x}, o) + (1-\tilde{q}_i)\,\mathrm{Gap}_{\underline{q}_i}(\mathbf{x}, \hat{a}_i) + \tilde{q}_i\,\mathrm{Gap}_{\overline{q}_i}(\mathbf{x}, \hat{b}_i)
\end{aligned} \tag{6.50}$$

□

Lemma 6.A.4. For any $\gamma > 0$ the estimate o_i of the quantile q_i by Algorithm 13 satisfies

$$Pr\{\mathrm{Gap}_{q_i}(\mathbf{x}, o_i) \geqslant \gamma\} \leqslant \frac{\exp\left(\varepsilon_i(\hat{\gamma}_i - \gamma)/2\right)}{\Psi_{\hat{\mathbf{x}}_i}^{(\tilde{q}_i, \varepsilon_i)}(\hat{\mu}_i)} \tag{6.51}$$

Proof. We use k_i to denote the interval $\hat{I}_k^{(j)}$ sampled at index i in the algorithm and note that o_i corresponds to the released number o at that index. Since $o_i \in [\hat{a}_i, \hat{b}_i]$, applying Lemma 6.A.3 yields

$$\begin{aligned}
\Pr(\mathrm{Gap}_{q_i}(\mathbf{x}, o_i) \geqslant \gamma) &= \sum_{j=0}^{\hat{n}_i} \Pr(k_i = j) 1_{\mathrm{Gap}_{q_i}(\mathbf{x}, \hat{I}_j^{(i)}) \geqslant \gamma} \\
&= \sum_{j=0}^{n_i} \frac{\exp(-\varepsilon\,\mathrm{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, \hat{I}_j^{(i)})/2) \hat{\mu}_i(\hat{I}_j^{(i)}) 1_{\mathrm{Gap}_{q_i}(\mathbf{x}, \hat{I}_j^{(i)}) \geqslant \gamma}}{\sum_{l=0}^{\hat{n}_i} \exp(\varepsilon u_{\tilde{q}_i}(\hat{\mathbf{x}}_i, \hat{I}_l^{(i)})/2) \hat{\mu}_i(\hat{I}_l)} \\
&\leqslant \frac{\exp(\varepsilon \hat{\gamma}_i/2)}{\Psi_{\hat{\mathbf{x}}_i}^{(\tilde{q}_i, \varepsilon_i)}(\hat{\mu}_i)} \sum_{j=0}^{n_i} \exp(-\varepsilon\,\mathrm{Gap}_{q_i}(\mathbf{x}, \hat{I}_j^{(i)})/2) \hat{\mu}_i(\hat{I}_j^{(i)}) 1_{\mathrm{Gap}_{q_i}(\mathbf{x}, \hat{I}_j^{(i)}) \geqslant \gamma} \\
&\leqslant \frac{\exp(\varepsilon(\hat{\gamma}_i - \gamma)/2)}{\Psi_{\hat{\mathbf{x}}_i}^{(\tilde{q}_i, \varepsilon_i)}(\hat{\mu}_i)}
\end{aligned} \tag{6.52}$$

□

Lemma 6.A.5. For any $\gamma > 0$ the estimate o_i of the quantile q_i by Algorithm 13 with edge-based prior adaptation satisfies

$$\Pr(\mathrm{Gap}_{q_i}(\mathbf{x}, o_i) \geqslant \gamma) \leqslant \frac{\exp(\varepsilon(\hat{\gamma}_i - \gamma/2))}{\Psi_{\mathbf{x}}^{(q_i,\varepsilon_i)}(\mu_i)} \tag{6.53}$$

Proof. Applying Lemma 6.A.3 yields the following lower bound on $\Psi_{\tilde{q}_i}^{(\varepsilon_i)}(\hat{\mathbf{x}}_i, \hat{\mu}_i)$:

$$\begin{aligned}
&\sum_{l=0}^{\hat{n}_i} \exp(\varepsilon u_{\tilde{q}_i}(\hat{\mathbf{x}}_i, \hat{I}_l^{(i)})/2)\hat{\mu}_i(\hat{I}_l^{(i)}) \\
&\quad = \exp(\varepsilon u_{\tilde{q}_i}(\hat{\mathbf{x}}_i, \hat{I}_0^{(i)})/2)\mu_i((-\infty, \hat{a}_i]) + \exp(\varepsilon u_{\tilde{q}_i}(\hat{\mathbf{x}}_i, \hat{I}_{\hat{n}_i}^{(i)})/2)\mu_i([\hat{b}_i, \infty)) \\
&\qquad + \sum_{l=0}^{\hat{n}_i} \exp(\varepsilon u_{\tilde{q}_i}(\hat{\mathbf{x}}_i, \hat{I}_l^{(i)})/2)\mu_i(\hat{I}_l) \\
&\quad = \sum_{l=0}^{\max_{\mathbf{x}_{[j]} < \hat{a}_i} j} \exp(-\varepsilon\, \mathrm{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, I_l \cap (-\infty, \hat{a}_i])/2)\mu_i(I_l \cap (-\infty, \hat{a}_i]) \\
&\qquad + \sum_{l=\max_{\mathbf{x}_{[j]} < \hat{b}_i} j}^{n} \exp(-\varepsilon\, \mathrm{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, I_l \cap [\hat{b}_i, \infty))/2)\mu_i(I_l \cap [\hat{b}_i, \infty)) \\
&\qquad + \sum_{l=\max_{\mathbf{x}_{[j]} < \hat{a}_i} j}^{\max_{\mathbf{x}_{[j]} < \hat{b}_i} j} \exp(-\varepsilon\, \mathrm{Gap}_{\tilde{q}_i}(\hat{\mathbf{x}}_i, I_l \cap [\hat{a}_i, \hat{b}_i])\mu_i(I_l \cap [\hat{a}_i, \hat{b}_i]) \\
&\quad \geqslant \Psi_{\mathbf{x}}^{(q_i,\varepsilon_i)}(\mu_i) \exp(-\varepsilon\hat{\gamma}_i/2)
\end{aligned} \tag{6.54}$$

Substituting into Lemma 6.A.2 yields the result. □

Lemma 6.A.6. Suppose $q_0 < q_1$ are two quantiles and $o_0 > o_1$. Then

$$\max_{i=0,1} \mathrm{Gap}_{q_i}(\mathbf{x}, o_i) \geqslant \max_{i=0,1} \mathrm{Gap}_{q_i}(\mathbf{x}, o_{1-i}) \tag{6.55}$$

Proof. We consider four cases. If $\lfloor q_0|\mathbf{x}| \rfloor \leqslant \max_{\mathbf{x}_{[j]} < o_1} j$ and $\lfloor q_1|X| \rfloor \leqslant \max_{\mathbf{x}_{[j]} < o_0} j$ then

$$\lfloor q_0|\mathbf{x}| \rfloor \leqslant \min\{\lfloor q_1|\mathbf{x}| \rfloor, \max_{\mathbf{x}_{[j]} < o_1} j\} \leqslant \max\{\lfloor q_1|\mathbf{x}| \rfloor, \max_{\mathbf{x}_{[j]} < o_1} j\} \leqslant \max_{\mathbf{x}_{[j]} < o_0} j \tag{6.56}$$

and so

$$\max_{i=0,1} \mathrm{Gap}_{q_i}(\mathbf{x}, o_i) = \max_{\mathbf{x}_{[j]} < o_0} j - \lfloor q_0|\mathbf{x}| \rfloor \geqslant \max_{i=0,1} \mathrm{Gap}_{q_i}(X, o_{i-1}) \tag{6.57}$$

If $\lfloor q_0|X| \rfloor \leqslant \max_{\mathbf{x}_{[j]} < o_1} j$ and $\lfloor q_1|\mathbf{x}| \rfloor > \max_{\mathbf{x}_{[j]} < o_0} j$ then

$$\lfloor q_0|\mathbf{x}| \rfloor \leqslant \max_{\mathbf{x}_{[j]} < o_1} j \leqslant \max_{\mathbf{x}_{[j]} < o_0} j < \lfloor q_1|\mathbf{x}| \rfloor \tag{6.58}$$

and so both improve after swapping. If $\lfloor q_0|\mathbf{x}|\rfloor > \max_{\mathbf{x}_{[j]}<o_1} j$ and $\lfloor q_1|\mathbf{x}|\rfloor > \max_{\mathbf{x}_{[j]}<o_0} j$ then

$$\max_{\mathbf{x}_{[j]}<o_1} j \leqslant \min\{\lfloor q_0|\mathbf{x}|\rfloor, \max_{\mathbf{x}_{[j]}<o_0} j\} \leqslant \max\{\lfloor q_0|\mathbf{x}|\rfloor, \max_{\mathbf{x}_{[j]}<o_0} j\} \leqslant \lfloor q_1|\mathbf{x}|\rfloor \tag{6.59}$$

and so

$$\max_{i=0,1} \mathrm{Gap}_{q_i}(\mathbf{x}, o_i) = \max_{\mathbf{x}_{[j]}<o_1} j - \lfloor q_1|\mathbf{x}|\rfloor \geqslant \max_{i=0,1} \mathrm{Gap}_{q_i}(\mathbf{x}, o_{i-1}) \tag{6.60}$$

Finally, if $\lfloor q_0|\mathbf{x}|\rfloor > \max_{\mathbf{x}_{[j]}<o_1} j$ and $\lfloor q_1|\mathbf{x}|\rfloor \leqslant \max_{\mathbf{x}_{[j]}<o_0} j$ then

$$\max_{\mathbf{x}_{[j]}<o_1} j < \lfloor q_0|\mathbf{x}|\rfloor \leqslant \lfloor q_1|\mathbf{x}|\rfloor \leqslant \max_{\mathbf{x}_{[j]}<o_0} j \tag{6.61}$$

so swapping will make the new largest error for each quantile at most as large as the other quantile's current error. □

6.B Covariance estimation

6.B.1 Section 6.3.2 details

Sensitivity results

Lemma 6.B.1. The eigenvalues $\mathbf{\Lambda}$ of $\mathbf{X}\mathbf{X}^\top/n - \mathbf{W} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ for $\mathbf{X} \in \mathbb{R}^{d\times n}$ with 1-bounded columns and symmetric $\mathbf{W} \in \mathbb{R}^{d\times d}$ has ℓ_1-sensitivity $2/n$, as does its trace norm $\|\mathbf{\Lambda}\|_1 = \|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}$.

Proof. Consider two datasets $\mathbf{X}, \tilde{\mathbf{X}} \in \mathbb{R}^{d\times n}$ that share the same first $n-1$ columns $\mathbf{Z} \in \mathbb{R}^{d\times n-1}$ but have different respective last columns $\mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^d$. For any vector $\mathbf{v} \in \mathbb{R}^d$ we have

$$\mathbf{v}^\top(\mathbf{X}\mathbf{X}^\top/n - \mathbf{W})\mathbf{v} = \mathbf{v}^\top\mathbf{Z}\mathbf{Z}^\top\mathbf{v}/n + \mathbf{v}^\top\mathbf{x}\mathbf{x}^\top\mathbf{v}/n - \mathbf{v}^\top\mathbf{W}\mathbf{v} \geqslant \mathbf{v}^\top(\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W})\mathbf{v} \tag{6.62}$$

so for $\hat{\mathbf{U}}\hat{\mathbf{\Lambda}}\hat{\mathbf{U}}^\top = \mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W}$ we have

$$\mathbf{\Lambda}_{[i]} \geqslant \hat{\mathbf{\Lambda}}_{[i]} \;\forall\, i \in [d] \tag{6.63}$$

Thus

$$\|\mathbf{\Lambda} - \hat{\mathbf{\Lambda}}\|_1 = \mathrm{Tr}(\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}) - \mathrm{Tr}(\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W}) = \mathrm{Tr}(\mathbf{x}\mathbf{x}^\top/n) \leqslant 1/n \tag{6.64}$$

The same argument holds when replacing $\mathbf{X}$ by $\tilde{\mathbf{X}}$, so the result for the eigenvalues follows by the triangle inequality.

For the trace norm we have that

$$\begin{aligned}\left|\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}} - \|\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}\right| = \left|\sum_{i=1}^d |\mathbf{\Lambda}_{[i]}| - |\hat{\mathbf{\Lambda}}_{[i]}|\right| &\leqslant \sum_{i=1}^d ||\mathbf{\Lambda}_{[i]}| - |\hat{\mathbf{\Lambda}}_{[i]}|| \\ &\leqslant \sum_{i=1}^d |\mathbf{\Lambda}_{[i]} - \hat{\mathbf{\Lambda}}_{[i]}| \leqslant 1/n\end{aligned} \tag{6.65}$$

where the second inequality holds trivially when $\mathbf{\Lambda}_{[i]}$ and $\hat{\mathbf{\Lambda}}_{[i]}$ have the same sign and otherwise the latter is negative (6.63) so we have $||\mathbf{\Lambda}_{[i]}| - |\tilde{\mathbf{\Lambda}}_{[i]}|| = |\mathbf{\Lambda}_{[i]} + \tilde{\mathbf{\Lambda}}_{[i]}| \leqslant |\mathbf{\Lambda}_{[i]} - \tilde{\mathbf{\Lambda}}_{[i]}|$, and the third is by Equation 6.64. This also holds when replacing $\mathbf{X}$ by $\tilde{\mathbf{X}}$, so the result follows by the triangle inequality. □

Claim 6.B.1. If $a_k, b_k \geqslant c_k \; \forall \; k \in [d]$ then $\sum_{k=1}^{d}(a_k - b_k)^2 \leqslant \left(\sum_{k=1}^{d} a_k - c_k\right)^2 + \left(\sum_{k=1}^{d} b_k - c_k\right)^2$.

Proof. Note that this result is an easy corollary of Dong et al. [2022, Fact 1], but for completeness:

$$\begin{aligned}
&\left(\sum_{k=1}^{d} a_k - c_k\right)^2 + \left(\sum_{k=1}^{d} b_k - c_k\right)^2 \\
&\qquad = \sum_{k=1}^{d}(a_k - c_k)^2 + (a_k - c_k)\sum_{j\neq k} a_j - c_j + \sum_{k=1}^{d}(b_k - c_k)^2 + (b_k - c_k)\sum_{j\neq k} b_j - c_j \\
&\qquad \geqslant \sum_{k=1}^{d} a_k^2 - 2a_kc_k + c_k^2 + b_k^2 - 2b_kc_k + c_k^2 \\
&\qquad = \sum_{k=1}^{d}(a_k - b_k)^2 + 2a_kb_k - 2a_kc_k - 2b_kc_k + 2c_k^2 \quad \geqslant \quad \sum_{k=1}^{d}(a_k - b_k)^2
\end{aligned} \tag{6.66}$$

where the first inequality follows because $a_k - c_k, b_k - c_k \geqslant 0 \; \forall \; k$ and the second because the convex function $a_kb_k - a_kc_k - 2b_kc_k + c_k^2$ attains its minimum over $c_k \in (-\infty, \min\{a_k, b_k\}]$ at $\min\{a_k, b_k\}$. □

Lemma 6.B.2. The ℓ_2-sensitivities of $\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$ and its eigenvalues $\mathbf{\Lambda}$ are both $\sqrt{2}/n$.

Proof. The first result follows directly by Biswas et al. [2020, Lemma 3.2] For the second, consider two datasets $\mathbf{X}, \tilde{\mathbf{X}} \in \mathbb{R}^{d\times n}$ that share the same first $n-1$ columns $\mathbf{Z} \in \mathbb{R}^{d\times n-1}$ but have different respective last columns $\mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^d$. Applying Claim 6.B.1, Equation 6.63, and Equation 6.64 yields

$$\|\mathbf{\Lambda} - \tilde{\mathbf{\Lambda}}\|_F^2 = \sum_{k=1}^{d}(\mathbf{\Lambda}_{[i]} - \tilde{\mathbf{\Lambda}}_{[i]})^2 \leqslant \left(\sum_{k=1}^{d} \mathbf{\Lambda}_{[i]} - \hat{\mathbf{\Lambda}}_{[i]}\right)^2 + \left(\sum_{k=1}^{d} \tilde{\mathbf{\Lambda}}_{[i]} - \hat{\mathbf{\Lambda}}_{[i]}\right)^2 \leqslant \frac{1}{n^2} + \frac{1}{n^2} = \frac{2}{n^2} \tag{6.67}$$

□

Proof of Lemma 6.3.1

Proof. Let $\mathbf{C} = \mathbf{X}\mathbf{X}^\top/n$. Then by triangle inequality and the orthonormality of $\tilde{\mathbf{U}}$ we have

$$\begin{aligned}
\|\mathbf{C} - \mathbf{W} - \tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top\|_F &= \|\mathbf{C} - \mathbf{W} - \tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top\|_F \\
&= \|\mathbf{C} - \mathbf{W} - \tilde{\mathbf{U}}\mathbf{\Lambda}\tilde{\mathbf{U}}^\top - \tilde{\mathbf{U}}(\hat{\mathbf{\Lambda}} - \mathbf{\Lambda})\tilde{\mathbf{U}}^\top\|_F \\
&\leqslant \|\mathbf{C} - \mathbf{W} - \tilde{\mathbf{U}}\mathbf{\Lambda}\tilde{\mathbf{U}}^\top\|_F + \|\tilde{\mathbf{U}}(\hat{\mathbf{\Lambda}} - \mathbf{\Lambda})\tilde{\mathbf{U}}^\top\|_F \\
&\leqslant \sqrt{\|\mathbf{C} - \mathbf{W}\|_F^2 - 2\operatorname{Tr}((\mathbf{C} - \mathbf{W})(\tilde{\mathbf{U}}\mathbf{\Lambda}\tilde{\mathbf{U}}^\top)) + \|\tilde{\mathbf{U}}\mathbf{\Lambda}\tilde{\mathbf{U}}^\top\|_F^2} + \|\mathbf{z}\|_2 \\
&= \sqrt{2\sum_{j=1}^d \mathbf{\Lambda}_{[j]}^2 - 2\operatorname{Tr}(\mathbf{\Lambda}\tilde{\mathbf{U}}^\top(\mathbf{C} - \mathbf{W})\tilde{\mathbf{U}})} + \|\mathbf{z}\|_2 \\
&= \sqrt{2\sum_{j=1}^d \mathbf{\Lambda}_{[j]}^2 - 2\sum_{j=1}^d \mathbf{\Lambda}_{[j]}\tilde{\mathbf{U}}_{[j]}^\top(\mathbf{C} - \mathbf{W})\tilde{\mathbf{U}}_{[j]}} + \|\mathbf{z}\|_2 \\
&= \sqrt{2\sum_{j=1}^d \mathbf{\Lambda}_{[j]}(\mathbf{\Lambda}_{[j]} - \tilde{\mathbf{U}}_{[j]}^\top(\mathbf{C} - \mathbf{W})\tilde{\mathbf{U}}_{[j]})} + \|\mathbf{z}\|_2
\end{aligned} \tag{6.68}$$

For $j \in [d]$ s.t. $\mathbf{\Lambda}_{[j,j]} > 0$ let $\mathbf{u}_j = \arg\max_{\|\mathbf{u}\|_2=1, \tilde{\mathbf{U}}_{[j:d]}\mathbf{u}=\mathbf{0}} \mathbf{u}(\mathbf{C} - \mathbf{W})\mathbf{u}$. By the Courant-Fischer-Weyl min-max principle we have that

$$\mathbf{\Lambda}_{[j]} = \min_{\mathbf{V}\in\mathbb{R}^{(d-j+1)\times d}} \max_{\|\mathbf{u}\|_2=1, \mathbf{V}\mathbf{u}=\mathbf{0}} \mathbf{u}^\top(\mathbf{C} - \mathbf{W})\mathbf{u} \leqslant \max_{\|\mathbf{u}\|_2=1, \tilde{\mathbf{U}}_{[j:d]}\mathbf{u}=\mathbf{0}} \mathbf{u}^\top(\mathbf{C} - \mathbf{W})\mathbf{u} = \mathbf{u}_j(\mathbf{C} - \mathbf{W})\mathbf{u}_j \tag{6.69}$$

Therefore

$$\begin{aligned}
\tilde{\mathbf{U}}_{[j]}^\top(\mathbf{X}\mathbf{X}^\top - \mathbf{W})\tilde{\mathbf{U}}_{[j]} &= \tilde{\mathbf{U}}_{[j]}^\top(\mathbf{X}\mathbf{X}^\top - \mathbf{W} + \mathbf{Z})\tilde{\mathbf{U}}_{[j]} - \tilde{\mathbf{U}}_{[j]}^\top\mathbf{Z}\tilde{\mathbf{U}}_{[j]} \\
&\geqslant \tilde{\mathbf{U}}_{[j]}^\top(\mathbf{X}\mathbf{X}^\top - \mathbf{W} + \mathbf{Z})\tilde{\mathbf{U}}_{[j]} - \|\mathbf{Z}\|_\infty \\
&\geqslant \mathbf{u}_j(\mathbf{C} - \mathbf{W} + \mathbf{Z})\mathbf{u}_j - \|\mathbf{Z}\|_\infty \\
&\geqslant \mathbf{u}_j(\mathbf{C} - \mathbf{W})\mathbf{u}_j - 2\|\mathbf{Z}\|_\infty \geqslant \mathbf{\Lambda}_{[j]} - 2\|\mathbf{Z}\|_\infty
\end{aligned} \tag{6.70}$$

Similarly, for $j \in [d]$ s.t. $\mathbf{\Lambda}_{[j,j]} < 0$ let $\mathbf{u}_j = \arg\min_{\|\mathbf{u}\|_2=1, \tilde{\mathbf{U}}_{[1:j]}\mathbf{u}=\mathbf{0}} \mathbf{u}(\mathbf{C} - \mathbf{W})\mathbf{u}$. By the Courant-Fischer-Weyl min-max principle we have that

$$\mathbf{\Lambda}_{[j]} = \max_{\mathbf{V}\in\mathbb{R}^{j\times d}} \min_{\|\mathbf{u}\|_2=1, \mathbf{V}\mathbf{u}=\mathbf{0}} \mathbf{u}^\top(\mathbf{C} - \mathbf{W})\mathbf{u} \geqslant \min_{\|\mathbf{u}\|_2=1, \tilde{\mathbf{U}}_{[1:j]}\mathbf{u}=\mathbf{0}} \mathbf{u}^\top(\mathbf{C} - \mathbf{W})\mathbf{u} = \mathbf{u}_j(\mathbf{C} - \mathbf{W})\mathbf{u}_j \tag{6.71}$$

Therefore

$$\begin{aligned}
\tilde{\mathbf{U}}_{[j]}^\top(\mathbf{C} - \mathbf{W})\tilde{\mathbf{U}}_{[j]} &= \tilde{\mathbf{U}}_{[j]}^\top(\mathbf{C} - \mathbf{W} + \mathbf{Z})\tilde{\mathbf{U}}_{[j]} - \tilde{\mathbf{U}}_{[j]}^\top\mathbf{Z}\tilde{\mathbf{U}}_{[j]} \\
&\leqslant \tilde{\mathbf{U}}_{[j]}^\top(\mathbf{C} - \mathbf{W} + \mathbf{Z})\tilde{\mathbf{U}}_{[j]} + \|\mathbf{Z}\|_\infty \\
&\leqslant \mathbf{u}_j(\mathbf{C} - \mathbf{W} + \mathbf{Z})\mathbf{u}_j + \|\mathbf{Z}\|_\infty \\
&\leqslant \mathbf{u}_j(\mathbf{C} - \mathbf{W})\mathbf{u}_j + 2\|\mathbf{Z}\|_\infty \leqslant \mathbf{\Lambda}_{[j]} + 2\|\mathbf{Z}\|_\infty
\end{aligned} \tag{6.72}$$

Substituting the bounds (6.70) and (6.72) in for the appropriate j terms in the summation in Equation 6.68 yields

$$\|\mathbf{C}-\mathbf{W}-\tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top\|_F \leqslant 2\|\mathbf{z}\|_2+2\sqrt{\|\mathbf{Z}\|_\infty \sum_{j=1}^{d}|\mathbf{\Lambda}_{[j]}|} = 2\left(\|\mathbf{z}\|_2+\sqrt{\|\mathbf{Z}\|_\infty\|\mathbf{C}-\mathbf{W}\|_{\mathrm{Tr}}}\right) \quad (6.73)$$

□

6.B.2 zCDP guarantees for SeparateCov with predictions

Definition 6.B.1 (Bun and Steinke [2016])**.** Algorithm $\mathcal{A}$ is ρ-zCDP if $D_\alpha(\mathcal{A}(\mathbf{X})||\mathcal{A}(\tilde{\mathbf{X}})) \leqslant \rho\alpha \;\forall\; \alpha > 1$ whenever $\mathbf{X}$ and $\tilde{\mathbf{X}}$ differ in a single element, where D_α is the α-Rényi divergence.

Theorem 6.B.1 (Bun and Steinke [2016])**.** If a query $q : \mathcal{X} \mapsto \mathbb{R}^d$ has an ℓ_2**-sensitivity** of $\max_{\mathbf{X}\sim\tilde{\mathbf{X}}}\|q(\mathbf{X})-q(\tilde{\mathbf{X}})\|_2^2 \leqslant \Delta$ then the **Gaussian mechanism**, which releases $q(\mathbf{X})+\mathbf{z}$ for $\mathbf{z}\sim\mathcal{N}(\mathbf{0}_d, \frac{\Delta^2}{2\rho})$, is ρ-zCDP.

Theorem 6.B.2. If $\mathbf{X}$ has columns bounded by 1 in ℓ_2-norm then Algorithm 14 is ρ-zCDP and w.p. $\geqslant 1-\beta$

$$\|\hat{\mathbf{C}}-\mathbf{X}\mathbf{X}^\top/n\|_F^2 \leqslant \tilde{\mathcal{O}}\left(\frac{d}{n^2\rho}+\frac{\|\mathbf{X}\mathbf{X}^\top/n-\mathbf{W}\|_{\mathrm{Tr}}}{n}\sqrt{\frac{d}{\rho}}\right) \quad (6.74)$$

Proof. The privacy guarantee follows from the composition of two Gaussian mechanisms with the sensitivities of Lemma 6.B.2. The utility guarantee is due to substituting Gaussian concentration from Dong et al. [2022, Lemmas 6 & 7] into Lemma 6.3.1. □

Corollary 6.B.1. If $\mathbf{X}$ has columns bounded by 1 in ℓ_2-norm then Algorithm 14 with $\mathbf{W}=\mathbf{0}_{d\times d}$ returns w.p. $\geqslant 1-\beta$ an estimate $\hat{\mathbf{C}}\in\mathbb{R}^{d\times d}$ satisfying

$$\|\hat{\mathbf{C}}-\mathbf{X}\mathbf{X}^\top/n\|_F^2 \leqslant \tilde{\mathcal{O}}\left(\frac{d}{n^2\rho}+\min_{c\in\mathbb{R}}\frac{\|\mathbf{X}\mathbf{X}^\top/n-c\mathbf{I}_d\|_{\mathrm{Tr}}}{n}\sqrt{\frac{d}{\rho}}\right) \quad (6.75)$$

Corollary 6.B.2. Pick $\lambda\in(0,1)$ and run Algorithm 14 with privacy $(1-\lambda)\rho$ and symmetric prediction matrix $\mathbf{W}$ if $\|\mathbf{X}\mathbf{X}^\top/n-\mathbf{W}\|_{\mathrm{Tr}}+z \leqslant \|\mathbf{X}\mathbf{X}^\top\|_{\mathrm{Tr}}/n$ and $\mathbf{0}_{d\times d}$, where $z\sim\mathcal{N}(0,\frac{1}{\lambda\rho n})$. This procedure is ρ-zCDP, $\tilde{\mathcal{O}}\left(\frac{\sqrt{d}}{n\sqrt{\rho}}\left(\frac{\sqrt{d}/n+1/\sqrt{n}}{\sqrt{\rho}}+\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}\right)\right)$-robust and $\tilde{\mathcal{O}}\left(\frac{\sqrt{d}}{n\sqrt{\rho}}\left(\frac{\sqrt{d}/n+1/\sqrt{n}}{\sqrt{\rho}}\right)\right)$-consistent.

Proof. By Lemma 6.B.2 the difference $\|\mathbf{X}\mathbf{X}^\top/n-\mathbf{W}\|_{\mathrm{Tr}}-\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}$ has sensitivity $2/\sqrt{n}$, so the comparison of $\|\mathbf{X}\mathbf{X}^\top-\mathbf{W}\|_{\mathrm{Tr}}+z$ and $\|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}$ is equivalent to using the Gaussian mechanism with $\lambda\rho$-zCDP to estimate this difference and then taking the sign. Composing this with the privacy guarantee of Theorem 6.B.2 yields ρ-zCDP. Since $\Pr(|z|\geqslant 2\sqrt{\frac{1}{\lambda\rho n}\log\frac{2}{\beta}}) \leqslant \beta/2$, the matrix $\mathbf{W}_z\in\{\mathbf{W},\mathbf{0}_{d\times d}\}$ passed to Algorithm 14 satisfies $\|\mathbf{X}\mathbf{X}^\top/n-\mathbf{W}_z\|_{\mathrm{Tr}} \leqslant \min\{\|\mathbf{X}\mathbf{X}^\top/n-\mathbf{W}\|_{\mathrm{Tr}}, \|\mathbf{X}\mathbf{X}^\top/n\|_{\mathrm{Tr}}+2\sqrt{\frac{1}{\lambda\rho n}\log\frac{2}{\beta}}$ w.p. $\geqslant 1-\beta/2$. Applying the utility guarantee of Theorem 6.B.2 w.p. $1-\beta/2$ for constant $\lambda\in(0,1)$ yields the result. □

6.B.3 IterativeEigenvectorSampling with predictions

Lemma 6.B.3. Given a dataset $\mathbf{X} \in \mathbb{R}^{d\times n}$ with 1-bounded columns, any orthonormal basis $\mathbf{P} \in \mathbb{R}^{k\times d}$, and a symmetric matrix $\mathbf{W} \in \mathbb{R}^{d\times d}$ the queries $\mathbf{u}^\top\mathbf{P}\{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+\mathbf{P}^\top\mathbf{u}$ and $\mathbf{u}^\top\mathbf{P}\{\mathbf{M} - \mathbf{X}\mathbf{X}^\top/n\}\mathbf{P}^\top\mathbf{u}$—where $\{\mathbf{A}\}_+$ denotes taking only the components of $\mathbf{A}$ with positive eigenvalues—have sensitivity $2/n$.

Proof. Consider two datasets $\mathbf{X}, \tilde{\mathbf{X}} \in \mathbb{R}^{d\times n}$ that share the same first $n-1$ columns $\mathbf{Z} \in \mathbb{R}^{d\times n-1}$ but have different respective last columns $\mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^d$. Let $\mathbf{P}_+$ and $\mathbf{Q}_+ \in \mathbb{R}^{d\times d}$ be projection matrices removing the negative components of $\mathbf{X}\mathbf{X}^\top - \mathbf{W}$ and $\mathbf{Z}\mathbf{Z}^\top - \mathbf{W}$, respectively. Then we have

$$\begin{aligned}
&\|\{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+ - \{\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W}\}_+\|_\infty \\
&\quad= \|\mathbf{P}_+(\mathbf{X}\mathbf{X}^\top/n - \mathbf{W})\mathbf{P}_+ - \mathbf{Q}_+(\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W})\mathbf{Q}_+\|_\infty \\
&\quad= \max_{\|\mathbf{v}\|_2=1} \mathbf{v}^\top\mathbf{P}_+(\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W})\mathbf{P}_+\mathbf{v} + \mathbf{v}^\top\mathbf{P}_+\mathbf{x}\mathbf{x}^\top\mathbf{P}_+\mathbf{v}/n - \mathbf{v}^\top\mathbf{Q}_+(\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W})\mathbf{Q}_+\mathbf{v} \\
&\quad\leqslant 1/n + \max_{\|\mathbf{v}\|_2=1} \mathbf{v}^\top\mathbf{Q}_+(\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W})\mathbf{Q}_+\mathbf{v} - \mathbf{v}^\top\mathbf{Q}_+(\mathbf{Z}\mathbf{Z}^\top/n - \mathbf{W})\mathbf{Q}_+\mathbf{v}/n = 1/n
\end{aligned} \tag{6.76}$$

where the second equality follows by (6.62) and the definition of the spectral norm. The same argument holds when replacing $\mathbf{X}$ by $\tilde{\mathbf{X}}$, so we can bound the sensitivity by the triangle inequality:

$$\begin{aligned}
|\mathbf{u}^\top\mathbf{P}\{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+\mathbf{P}^\top\mathbf{u} - \mathbf{u}^\top\mathbf{P}\{\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top/n - \mathbf{W}\}_+\mathbf{P}^\top\mathbf{u}| \\
\leqslant \|\mathbf{P}(\{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+ - \{\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top/n - \mathbf{W}\}_+)\mathbf{P}^\top\|_\infty \\
\leqslant \|\{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+ - \{\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top/n - \mathbf{W}\}_+\|_\infty \leqslant 2/n
\end{aligned} \tag{6.77}$$

Similarly, for $\mathbf{P}_-$ and $\mathbf{Q}_- \in \mathbb{R}^{d\times d}$ the projection matrices removing the negative components of $\mathbf{X}\mathbf{X}^\top - \mathbf{W}$ and $\mathbf{Z}\mathbf{Z}^\top - \mathbf{W}$, respectively, we have

$$\begin{aligned}
&\|\{\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n\}_+ - \{\mathbf{W} - \mathbf{Z}\mathbf{Z}^\top/n\}_+\|_\infty \\
&\quad= \|\mathbf{P}_-(\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n)\mathbf{P}_- - \mathbf{Q}_-(\mathbf{W} - \mathbf{Z}\mathbf{Z}^\top/n)\mathbf{Q}_-\|_\infty \\
&\quad= \max_{\|\mathbf{v}\|_2=1} \mathbf{v}^\top\mathbf{Q}_-(\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n)\mathbf{Q}_-\mathbf{v} + \mathbf{v}^\top\mathbf{Q}_-\mathbf{x}\mathbf{x}^\top\mathbf{Q}_-\mathbf{v}/n - \mathbf{v}^\top\mathbf{P}_-(\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n)\mathbf{P}_-\mathbf{v} \\
&\quad\leqslant 1/n + \max_{\|\mathbf{v}\|_2=1} \mathbf{v}^\top\mathbf{P}_-(\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n)\mathbf{P}_-\mathbf{v} - \mathbf{v}^\top\mathbf{P}_-(\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n - \mathbf{W})\mathbf{P}_-\mathbf{v} = 1/n
\end{aligned} \tag{6.78}$$

where the second equality follows by (6.62) and the definition of the spectral norm. The same argument holds when replacing $\mathbf{X}$ by $\tilde{\mathbf{X}}$, so as before we can obtain the sensitivity via the triangle inequality. □

Theorem 6.B.3. Algorithm 15 preserves $\left(\sum_{s\in\{\pm1\}}\sum_{i=0}^{d}\varepsilon_i^{(s)}\right)$-DP and the output $\hat{\mathbf{C}}$ satisfies w.p.

$\geqslant 1-\beta$

$$\|\mathbf{X}\mathbf{X}^\top/n - \hat{\mathbf{C}}\|_F^2 \leqslant \tilde{\mathcal{O}}\left(\frac{d}{n}\left(\sum_{s\in\{\pm1\}}\frac{1}{\varepsilon_0^{(s)^2}n} + \sum_{i=1}^{d}\frac{|\mathbf{\Lambda}_{[i]}|}{\varepsilon_i^{(\mathbf{S}_{[i]})}}\right)\right) \tag{6.79}$$

where $\mathbf{\Lambda}_{[i]}$ is the matrix of eigenvalues of $\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$, $\mathbf{S}_{[i]}$ is the matrix of its signs, and $\tilde{\mathcal{O}}$ hides logarithmic factors in d, $\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_\infty$, and β.

Proof. The privacy result follows from Lemma 6.B.3 applied to Algorithm 15's release of $\lambda_i^{(s)}$ and $\hat{\mathbf{u}}_i^{(s)}$ using the Laplace and Exponential mechanisms, respectively, followed by basic composition. For utility, since $\mathbf{X}\mathbf{X}^\top/n - \hat{\mathbf{C}} = \mathbf{X}\mathbf{X}^\top - \mathbf{W} - \hat{\mathbf{C}} + \mathbf{W} = \{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+ - \{\hat{\mathbf{C}} - \mathbf{W}\}_+ - \{\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n\}_+ + \{\mathbf{W} - \hat{\mathbf{C}}\}_+$ we have by the triangle inequality, the fact that $(a+b)^2 \leqslant 2(a^2+b^2)\ \forall\ a,b \in \mathbb{R}$, and the utility guarantee (squared and normalized by n) of IterativEigenvectorSampling [Amin et al., 2019, Theorem 1] applied to $\{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+$ and $\{\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n\}_+$ that

$$\begin{aligned}
&\|\mathbf{X}\mathbf{X}^\top/n - \hat{\mathbf{C}}\|_F^2 \\
&\quad\leqslant 2\|\{\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\}_+ - \{\hat{\mathbf{C}} - \mathbf{W}\}_+\|_F^2 + 2\|\{\mathbf{W} - \mathbf{X}\mathbf{X}^\top/n\}_+ - \{\mathbf{W} - \hat{\mathbf{C}}\}_+\|_F^2 \\
&\quad\leqslant \tilde{\mathcal{O}}\left(d\left(\frac{1}{(\varepsilon_0^{(1)}n)^2} + \sum_{i=1}^{d}\frac{\max\{\mathbf{\Lambda}_{[i]},0\}}{\varepsilon_i^{(1)}n} + \frac{1}{(\varepsilon_0^{(-1)}n)^2} + \sum_{i=1}^{d}\frac{\max\{-\mathbf{\Lambda}_{[i]},0\}}{\varepsilon_i^{(-1)}n}\right)\right) \\
&\quad= \tilde{\mathcal{O}}\left(d\left(\sum_{s\in\{\pm1\}}\frac{1}{(\varepsilon_0^{(s)}n)^2} + \sum_{i=1}^{d}\frac{|\mathbf{\Lambda}_{[i]}|}{\varepsilon_i^{(S_{[i]})}n}\right)\right)
\end{aligned} \tag{6.80}$$

□

Corollary 6.B.3. Suppose for $s \in \{\pm1\}$ we set $\varepsilon_0^{(s)} = \varepsilon/4$ and $\varepsilon_i^{(s)} = \frac{\varepsilon}{4d}\ \forall\ i \in [d]$, where $\varepsilon > 0$ is the overall privacy budget. Then w.p. $1-\beta$ the output $\hat{\mathbf{C}}$ of Algorithm 15 satisfies

$$\|\mathbf{X}\mathbf{X}^\top/n - \hat{\mathbf{C}}\|_F^2 \leqslant \tilde{\mathcal{O}}\left(\frac{d}{\varepsilon n}\left(\frac{1}{\varepsilon n} + d\|\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}\|_{\mathrm{Tr}}\right)\right) \tag{6.81}$$

6.C Data release

Proof of Lemma 6.3.2. Follow the proof of the original [Hardt et al., 2012] but replace Fact A.3 by $\Psi_0 \leqslant D_{\mathrm{KL}}(\frac{\mathbf{x}}{n}||\mathbf{w})$, upper bound the square of the result by twice the sum of the squares of the two terms, and obtain guarantees w.p. $\geqslant 1-\beta$ by solving $\frac{2m}{|Q|^c} = \beta$ for c and substituting the solution into the bound. □

6.D Online learning

6.D.1 Negative log-inner-product losses

For functions of the form $f_t(\mu) = -\log\int_a^b s_t(o)\mu(o)do$, we showed $\tilde{\mathcal{O}}(T^{3/4})$ regret for the case $s_t(o) \in \{0,1\}\ \forall\ o \in [a,b]$ using a variant of exponentiated gradient with a dynamic dis-

cretization (c.f. Section 2.3). Notably their algorithm can be extended to (non-privately) learn $-\log \Psi^{(q)}_{\mathbf{x}_t}(\mu)$, since s_t in this case is one on the optimal interval and zero elsewhere. However, the changing discretization and dependence of the analysis on the range of s_t suggests it may be difficult to privatize their approach. The discretized form $-\log\langle \mathbf{s}_t, \mathbf{w}\rangle$ is more heavily studied, arising in portfolio management [Cover, 1991]. It enjoys the exp-concavity property, leading to $\mathcal{O}(d \log T)$ regret using the EWOO method [Hazan et al., 2007]. However, EWOO requires maintaining and sampling from a distribution defined by a product of inner products, which is inefficient and similarly difficult to privatize. Other algorithms, e.g. adaptive FTAL [Hazan et al., 2007], also attain logarithmic regret for exp-concave functions, but the only private variant we know of is non-adaptive and only guarantees $\mathcal{O}(\sqrt{T})$-regret for non-strongly-convex losses [Smith and Thakurta, 2013]. The adaptivity, which is itself data-dependent, seems critical for taking advantage of exp-concavity.

Lemma 6.D.1. If $f_t(\boldsymbol{\mu}_{\mathbf{W}}) = -\log \sum_{i=1}^m \frac{1/m}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle}$ for some $\mathbf{s}_{t,i} \in \mathbb{R}^d_{\geqslant 0}$ then we have the bound $\|\nabla_{\mathbf{W}} f_t(\boldsymbol{\mu}_{\mathbf{W}})\|_1 \leqslant d/\gamma \;\forall\, \mathbf{W} \in \triangle_d^m$ s.t. $\mathbf{W}_{[i,j]} \geqslant \gamma/d \;\forall\, i, j$ for some $\gamma \in (0, 1]$.

Proof.

$$\begin{aligned}\|\nabla_{\mathbf{W}} f_t(\boldsymbol{\mu}_{\mathbf{W}})\|_1 = \sum_{i=1}^m \|\nabla_{\mathbf{W}_{[i]}} f_t(\mu_{\mathbf{W}})\|_1 &= \left(\sum_{i=1}^m \frac{1}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle}\right)^{-1} \sum_{i=1}^m \sum_{j=1}^d \frac{\mathbf{s}_{t,i[j]}}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^2} \\ &\leqslant \left(\sum_{i=1}^m \frac{1}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle}\right)^{-1} \sum_{i=1}^m \frac{1}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]} \odot \mathbf{W}_{[i]}\rangle} \leqslant d/\gamma\end{aligned} \tag{6.82}$$

where the first inequality follows by Sedrakyan's inequality and the second by $\mathbf{W}_{[i,j]} \geqslant \gamma/d$. □

Proof of Lemma 6.5.1 for $m > 1$

Proof. Let $\tilde{\mathbf{x}}_t$ be a neighboring dataset of $\mathbf{x}_t$ constructed by adding or removing a single element, and let $\tilde{U}_t$ be the corresponding loss function. We note that changing from $\mathbf{x}_t$ to $\tilde{\mathbf{x}}_t$ changes the value of $\mathrm{Gap}_{q_i}(\mathbf{x}_t, o)$ at any point $o \in [a, b]$ by at most ± 1 and so the value of the exponential score at any point $o \in [a, b]$ is changed by at most a multiplicative factor $\exp(-\varepsilon_i/2)$ in either direction. Therefore

$$\begin{aligned}\tilde{\mathbf{s}}_{t,i[j]} &= \int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j} \exp(-\varepsilon_i \,\mathrm{Gap}_{q_i}(\tilde{\mathbf{x}}_t, o)/2) do \\ &\in \exp(\pm\varepsilon_i/2) \int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j} \exp(-\varepsilon_i \,\mathrm{Gap}_{q_i}(\mathbf{x}_t, o)/2) do = \exp(\pm\varepsilon_i/2)\mathbf{s}_{t,i[j]}\end{aligned} \tag{6.83}$$

where $\pm$ indicates the interval between values.

$$
\begin{aligned}
&\|\nabla_{\mathbf{W}} U_t(\mathbf{W}) - \nabla_{\mathbf{W}} \tilde{U}_t(\mathbf{W})\|_F \\
&= \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{d}\left(\left(\sum_{i'=1}^{m}\frac{1}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}\right)^{-1}\frac{\mathbf{s}_{t,i[j]}}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^2} - \left(\sum_{i'=1}^{m}\frac{1}{\langle \tilde{\mathbf{s}}_{t,i'}, \mathbf{W}_{[i']}\rangle}\right)^{-1}\frac{\tilde{\mathbf{s}}_{t,i[j]}}{\langle \tilde{\mathbf{s}}_{t,i}, \mathbf{W}_{[i]}\rangle^2}\right)^2} \\
&= \left(\sum_{i'=1}^{m}\frac{1}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}\right)^{-1}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{d}\left(\frac{\mathbf{s}_{t,i[j]}}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^2} - \frac{\tilde{\mathbf{s}}_{t,i[j]}}{\langle \tilde{\mathbf{s}}_{t,i}, \mathbf{W}_{[i]}\rangle^2}\frac{\sum_{i'=1}^{m}\frac{1}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}}{\sum_{i'=1}^{m}\frac{1}{\langle \tilde{\mathbf{s}}_{t,i'}, \mathbf{W}_{[i']}\rangle}}\right)^2} \\
&= \left(\sum_{i'=1}^{m}\frac{1}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}\right)^{-1}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{d}\frac{\mathbf{s}_{t,i[j]}^2}{\langle \mathbf{W}_{t,i}, \mathbf{W}_{[i]}\rangle^4}\left(1 - \frac{\langle \mathbf{g}_{t,i}, \mathbf{x}_{[i]}\rangle^2}{\langle \tilde{\mathbf{s}}_{t,i}, \mathbf{W}_{[i]}\rangle^2}\frac{\sum_{i'=1}^{m}\frac{\tilde{\mathbf{s}}_{t,i[j]}}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}}{\sum_{i'=1}^{m}\frac{\mathbf{s}_{t,i[j]}}{\langle \tilde{\mathbf{s}}_{t,i'}, \mathbf{W}_{[i']}\rangle}}\right)^2} \\
&\leqslant \left(\sum_{i'=1}^{m}\frac{1}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}\right)^{-1}\sum_{i=1}^{m}\sum_{j=1}^{d}\frac{\mathbf{s}_{t,i[j]}}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^2}|1-\kappa_{i,j}| \leqslant \frac{d}{\gamma}\max_{i,j}|1-\kappa_{i,j}|
\end{aligned}
\tag{6.84}
$$

where we have

$$
\begin{aligned}
\kappa_{i,j} = \frac{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^2}{\langle \tilde{\mathbf{s}}_{t,i}, \mathbf{x}_{[i]}\rangle^2}\frac{\sum_{i'=1}^{m}\frac{\tilde{\mathbf{s}}_{t,i[j]}}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}}{\sum_{i'=1}^{m}\frac{\mathbf{s}_{t,i[j]}}{\langle \tilde{\mathbf{s}}_{t,i'}, \mathbf{W}_{[i']}\rangle}} &\in \frac{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^2}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^2\exp(\pm\varepsilon_i)}\frac{\sum_{i'=1}^{m}\frac{\mathbf{s}_{t,i[j]}\exp(\pm\frac{\varepsilon_{i'}}{2})}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle}}{\sum_{i'=1}^{m}\frac{\mathbf{s}_{t,i[j]}}{\langle \mathbf{s}_{t,i'}, \mathbf{W}_{[i']}\rangle\exp(\pm\frac{\varepsilon_{i'}}{2})}} \\
&= \exp(\pm 2\max_i \varepsilon_i)
\end{aligned}
\tag{6.85}
$$

Substituting into the previous inequality and taking the minimum with the ℓ_1 bound on the gradient of the losses from Lemma 6.D.1 yields the result. □

Proof of Theorem 6.5.2

Proof. For set of γ-robust priors $\boldsymbol{\rho}$ s.t. $\boldsymbol{\rho}_{[i]} = \min\{1-\gamma+\lambda, 1\}\boldsymbol{\mu}_{[i]} + \frac{\max\{\gamma-\lambda,0\}}{b-a}$ and $\mathbf{W} \in \triangle_d^m$ s.t. $\mathbf{W}_{[i,j]} = \frac{b-a}{d}\int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j}\boldsymbol{\rho}_{[i]}(o)do$ we can divide the regret into three components:

$$
\begin{aligned}
&\sum_{t=1}^{T} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{W}_t}) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}) \\
&\quad= \sum_{t=1}^{T} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{W}_t}) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{W}}) + \sum_{t=1}^{T} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{W}}) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\rho}) + \sum_{t=1}^{T} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\rho}) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu})
\end{aligned}
\tag{6.86}
$$

The first summation is the regret of DP-FTRL with regularizer ϕ, which is strongly convex w.r.t. $\|\cdot\|_1$. The Gaussian width of its unit ball is $2\sqrt{\log(md)}$, by Lemma 6.D.1 the losses are $\frac{d}{\gamma}$-Lipschitz w.r.t. $\|\cdot\|_1$, and by Lemma 6.5.1 the ℓ_2-sensitivity is $\Delta_2 = \frac{d}{\gamma}\min\{2, e^{\tilde{\varepsilon}_m} - 1\} \leqslant$

$\frac{2d}{\gamma}\min\{1,\tilde{\varepsilon}_m\}$, so applying Theorem 6.5.1 yields the bound

$$\frac{m^2\log d}{\eta}+\frac{\eta d^2 T}{\gamma^2}\left(1+\left(4\sqrt{\log d}+2\sqrt{2\log\frac{T}{\beta'}}\right)\sigma\sqrt{\lceil\log_2 T\rceil}\min\{1,\varepsilon\}\right) \tag{6.87}$$

The second summation is a sum over the errors due to discretization, where we have

$$\begin{aligned}
&\sum_{t=1}^{T} U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}_{\mathbf{W}}) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\rho}) \\
&\quad= \sum_{t=1}^{T}\log\sum_{i=1}^{m}\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle^{-1} - \log\sum_{i=1}^{m}\frac{1}{\int_a^b \exp(-\varepsilon_i \operatorname{Gap}_{q_i}(\mathbf{x}_t, o)/2)\boldsymbol{\rho}_{[i]}(o)do} \\
&\quad\leqslant \sum_{t=1}^{T}\sum_{i=1}^{m}\frac{\int_a^b \exp(-\frac{\varepsilon_i}{2}\operatorname{Gap}_{q_i}(\mathbf{x}_t, o))\boldsymbol{\rho}_{[i]}(o)do - \langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle}{\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle} \\
&\quad\leqslant \sum_{t=1}^{T}\sum_{i=1}^{m}\frac{\sum_{j=1}^{d}\int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j}\exp(-\frac{\varepsilon_i}{2}\operatorname{Gap}_{q_i}(\mathbf{x}_t, o))(\boldsymbol{\rho}_{[i]}(o) - \boldsymbol{\mu}_{\mathbf{W}_{[i]}}(o))do}{\gamma\psi_{\mathbf{x}_t}/(b-a)} \\
&\quad\leqslant \sum_{t=1}^{T}\sum_{i=1}^{m}\frac{\sum_{j=1}^{d}\int_{a+\frac{b-a}{d}(j-1)}^{a+\frac{b-a}{d}j}|\boldsymbol{\rho}_{[i]}(o) - \boldsymbol{\rho}_{[i]}(o_{i,j})|do}{\gamma\psi_{\mathbf{x}_t}/(b-a)} \leqslant \frac{VmT}{\gamma d\bar{\psi}}(b-a)^3
\end{aligned} \tag{6.88}$$

where the first inequality follows by concavity, the second by using the definition of $\mathbf{W}$ to see that $\langle \mathbf{s}_{t,i}, \mathbf{W}_{[i]}\rangle = \int_a^b \exp(-\frac{\varepsilon_i}{2}\operatorname{Gap}_{q_i}(\mathbf{x}_t, o))\mu_{\mathbf{W}_{[i]}}(o)do \geqslant \frac{\gamma\psi_{\mathbf{x}_t}}{b-a}$, the third by Hölder's inequality and the mean value theorem for some $o_{i,j} \in (a+\frac{b-a}{d}(j-1), a+\frac{b-a}{d}j)$, and the fourth by the Lipschitzness of $\boldsymbol{\rho}_{[i]} \in \mathcal{F}_{V,d}^{(\gamma)}$. The third summation is a sum over the errors due to γ-robustness, with the result following by $U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\rho}) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}) \leqslant U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}) - \log(1-\max\{\gamma-\lambda,0\}) - U_{\mathbf{x}_t}^{(\varepsilon)}(\boldsymbol{\mu}) \leqslant 2\max\{\gamma-\lambda,0\}\log 2$. □

Settings of γ and d for Corollary 6.5.1

1. λ-robust and discrete $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{0,d}^{(\lambda)}$: $\gamma = \lambda$
2. λ-robust and V-Lipschitz $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{V,1}^{(\lambda)}$: $\gamma = \lambda$ and $d = \left\lceil \sqrt{\frac{V(b-a)^3}{\bar{\psi}}\sqrt{\left(1+\frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right)T}} \right\rceil$
3. discrete $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{0,d}$: $\gamma = \sqrt{md}\sqrt[4]{\frac{1+\min\{1,\tilde{\varepsilon}_m\}/\varepsilon'}{T}}$
4. V-Lipschitz $\boldsymbol{\mu}_{[i]} \in \mathcal{F}_{V,1}$: $\gamma = \sqrt{m}\sqrt[4]{\frac{V(b-a)^3}{\bar{\psi}}}\sqrt[8]{\frac{1+\min\{1,\tilde{\varepsilon}_m\}/\varepsilon'}{T}}$ and $d = \left\lceil \sqrt{\frac{V(b-a)^3}{\bar{\psi}}\sqrt{\left(1+\frac{\min\{1,\tilde{\varepsilon}_m\}}{\varepsilon'}\right)T}} \right\rceil$

6.D.2 Data release

Lemma 6.D.2. For $\mathbf{w} \in \triangle_d$ s.t. $\mathbf{w}_{[i]} \geqslant \gamma/d \ \forall \ i \in [n]$ the gradient $\nabla_{\mathbf{w}} D_{\mathrm{KL}}(\frac{\mathbf{x}}{n}||\mathbf{w})$ of the KL divergence w.r.t. its second argument is bounded in ℓ_∞-norm by d/γ and has ℓ_2-sensitivity $\frac{d\sqrt{2}}{\gamma n}$.

Proof. We have $\nabla_{\mathbf{w}} D_{\mathrm{KL}}(\mathbf{x}/n||\mathbf{w}) = -\nabla_{\mathbf{w}}\langle \mathbf{x}/n, \log \mathbf{w}\rangle = \frac{\mathbf{x}}{n\mathbf{w}}$ so since $\mathbf{w}_{[i]} \geqslant \gamma/d$ we have that $\|\nabla_{\mathbf{w}} D_{\mathrm{KL}}(\mathbf{x}/n||\mathbf{w})\|_\infty = \left\|\frac{\mathbf{x}}{n\mathbf{w}}\right\|_\infty \leqslant \frac{d \max_i \mathbf{x}_{[i]}}{\gamma n} \leqslant d/\gamma$ Furthermore, for neighboring datasets $\mathbf{x}$ and $\tilde{\mathbf{x}}$ we have

$$\|\nabla_{\mathbf{w}} D_{\mathrm{KL}}(\mathbf{x}/n||\mathbf{w}) - \nabla_{\mathbf{w}} D_{\mathrm{KL}}(\tilde{\mathbf{x}}/n||\mathbf{w})\|_2 = \left\|\frac{\mathbf{x}}{n\mathbf{w}} - \frac{\tilde{\mathbf{x}}}{n\mathbf{w}}\right\|_2 \leqslant \frac{d\sqrt{2}}{\gamma n} \tag{6.89}$$

□

Lemma 6.D.3. For $\mathbf{w} \in \triangle_d$ s.t. $\mathbf{w}_{[i]} \geqslant \gamma/d \ \forall \ i \in [n]$ the gradient $\nabla_\theta \mathbb{E}_{m\sim\theta} U_t(\mathbf{w}, m)$ has ℓ_2-sensitivity at most $7\pi \log \frac{d}{\gamma}$.

Proof. For any $\mathbf{x}_t$ and neighboring $\tilde{\mathbf{x}}_t$ that replaces one element we have

$$|D_{\mathrm{KL}}(\mathbf{x}_t/n_t||\mathbf{w}) - D_{\mathrm{KL}}(\tilde{\mathbf{x}}_t/n_t||\mathbf{w})| = |\langle \mathbf{x}_t - \tilde{\mathbf{x}}_t, \log \mathbf{w}\rangle|/n_t \leqslant \frac{2}{n_t} \log \frac{d}{\gamma} \tag{6.90}$$

Therefore

$$\begin{aligned}\|\nabla_\theta \mathbb{E}_{m\sim\theta} U_t(\mathbf{w}, m) - \nabla_\theta \mathbb{E}_{m\sim\theta} \tilde{U}_t(\mathbf{w}, m)\|_2 &\leqslant 8n_t \sqrt{\sum_{m=1}^{\infty} \left(\frac{D_{\mathrm{KL}}(\mathbf{x}_t/n_t||\mathbf{w}) - D_{\mathrm{KL}}(\tilde{\mathbf{x}}_t/n_t||\mathbf{w})}{m}\right)^2} \\ &\leqslant 7\pi \log \frac{d}{\gamma}\end{aligned} \tag{6.91}$$

□

6.D.3 Proof of Theorem 6.5.4

Proof. Let $M = \left\lceil \sqrt[3]{\frac{\varepsilon^2 N^2 \log \frac{d}{\gamma}}{16 \log^2 \frac{2|Q|}{\beta}}} \right\rceil$ and note that the m minimizing $\sum_{t=1}^T U_t(\mathbf{w}_t, m)$ is in $[M]$ and also

$$\begin{aligned}\max_{t,m} U_t(\mathbf{w}_t, m) &\leqslant 8N \log \frac{d}{\gamma} + 54 \log^{\frac{2}{3}} \frac{d}{\gamma} ((N^2/\varepsilon)^{\frac{2}{3}} + 1/\varepsilon^2) \left(\log^2 \frac{\varepsilon N \log \frac{d}{\gamma}}{\beta} + \log^4 |Q|\right) \\ &= \tilde{\mathcal{O}}\left(\frac{N^{\frac{4}{3}} \log^4 \frac{|Q|}{\beta}}{\min\{1, \varepsilon^2\}} \log \frac{d}{\gamma}\right)\end{aligned} \tag{6.92}$$

Letting $A = \mathcal{O}\left(\log \frac{d}{\gamma}\right)$ and $B = \tilde{\mathcal{O}}\left(\frac{N^{\frac{4}{3}} \log^4 \frac{|Q|}{\beta}}{\min\{1,\varepsilon^2\}} \log \frac{d}{\gamma}\right)$ we have

$$
\begin{aligned}
&\sum_{t=1}^{T} U_t(\mathbf{w}_t, m_t) \\
&\quad\leqslant \frac{\log M}{\eta_2} + \eta_2 \left(B \left(B + \left(2\sqrt{\log M} + \sqrt{2 \log \frac{3T}{\beta'}} \right) \right) \sigma_2 A \sqrt{\lceil \log_2 T \rceil} \right) T \\
&\qquad + B \sqrt{\frac{T}{2} \log \frac{3}{\beta'}} + \min_{\theta \in \Theta} \sum_{t=1}^{T} U_t(\mathbf{w}_t, \theta) \\
&\quad\leqslant \frac{\log M}{\eta_2} + \eta_2 \left(B \left(B + \left(2\sqrt{\log M} + \sqrt{2 \log \frac{3T}{\beta'}} \right) \right) \sigma_2 A \sqrt{\lceil \log_2 T \rceil} \right) T \\
&\qquad + B \sqrt{\frac{T}{2} \log \frac{3}{\beta'}} + \min_{m \in [M]} \sum_{t=1}^{T} \frac{8 n_t}{m} D_{\mathrm{KL}} \left(\frac{\mathbf{x}_t}{n_t} \middle\| \mathbf{w}_t \right) + \frac{16 m^2}{\varepsilon^2 n_t} \left(3 \log \frac{2m}{\beta} + 2 \log^2 |Q| \right)^2 \\
&\quad\leqslant \frac{\log M}{\eta_2} + \eta_2 \left(B \left(B + \left(2\sqrt{\log M} + \sqrt{2 \log \frac{3T}{\beta'}} \right) \right) \sigma_2 A \sqrt{\lceil \log_2 T \rceil} \right) T \\
&\qquad + 8 \left(\frac{\log d}{\eta_1} + \eta_1 \frac{Nd}{\gamma} \left(\frac{Nd}{\gamma} + \left(2\sqrt{\log d} + \sqrt{2 \log \frac{3T}{\beta'}} \right) \sigma_1 \frac{d\sqrt{2}}{\gamma} \sqrt{\lceil \log_2 T \rceil} \right) T \right) \\
&\qquad + B \sqrt{\frac{T}{2} \log \frac{3}{\beta'}} + \min_{m > 0, \mathbf{w}_{[i]} \geqslant \gamma/d} \sum_{t=1}^{T} U_t(\mathbf{w}, m) \\
&\quad\leqslant \tilde{\mathcal{O}}\left(\sqrt{(B + A\sigma_2) BT} \right) + \tilde{\mathcal{O}}\left(\frac{d}{\gamma} \sqrt{(N + \sigma_1) NT} \right) + B \sqrt{\frac{T}{2} \log \frac{3}{\beta'}} \\
&\qquad + \tilde{\mathcal{O}}\left(\max\{\gamma - \lambda, 0\} NT \right) + \min_{m > 0, \mathbf{w}_{[i]} \geqslant \lambda/d} \sum_{t=1}^{T} U_t(\mathbf{w}, m) \\
&\quad= \tilde{\mathcal{O}}\left(\left(\frac{N^{\frac{4}{3}}}{\min\{1, \varepsilon^2\}} + \frac{N^{\frac{2}{3}}/\sqrt{\varepsilon'}}{\min\{1, \varepsilon\}} + \frac{dN}{\gamma} + \frac{d}{\gamma} \sqrt{\frac{N}{\varepsilon'}} \right) \sqrt{T} + \max\{\gamma - \lambda, 0\} NT \right) \\
&\qquad + \min_{m > 0, \mathbf{w}_{[i]} \geqslant \lambda/d} \sum_{t=1}^{T} U_t(\mathbf{w}, m)
\end{aligned}
\tag{6.93}
$$

where the first inequality follows by the regret of DP-FTRL w.r.t. θ together with Cesa-Bianchi and Lugosi [2006, Lemma 4.1], the second by noting the definition of U_t and restricting to integer m, the third by the guarantee of DP-FTRL w.r.t. $\mathbf{w}$, and the fourth by joint-convexity of D_{KL} and simplifying terms. □

6.E Section 6.6 details

Code to reproduce the experimental results in Section 6.6 is available here: `https://github.com/mkhodak/private-quantiles`.

6.E.1 Location-scale families

A location-scale model is a distribution parameterized by a location $\nu \in \mathbb{R}$ and scale $\sigma \in \mathbb{R}_{\geqslant 0}$ with density of form $\mu_{\nu,\sigma}(x) = \frac{1}{\sigma} f\left(\frac{x-\nu}{\sigma}\right)$ for some centered probability measure $f : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$.

Impossibility of simultaneous robustness and convexity

Theorem 6.E.1. Let $f : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$ be a centered probability measure and for each $\theta \in \Theta$ define $\mu_\theta(x) = f(x - \theta)$.

1. If f is continuous then $U_{\mathbf{x}}(\mu_\theta)$ is convex in θ for all sorted dataset $\mathbf{x} \in \mathbb{R}^n$ if and only if f is log-concave.
2. There exist constants $a, b > 0$ s.t. for any $r > 0$, $\psi \in (0, \frac{R}{2n}]$, $q \geqslant \frac{1}{n}$, and $\theta \in \mathbb{R}$ there exists a sorted dataset $\mathbf{x} \in (\theta \pm R)^n$ with $\min_{i \in [n-1]} \mathbf{x}_{[i+1]} - \mathbf{x}_{[i]} = \psi$ s.t. $U^{(q)}_{\mathbf{x}}(\mu_\theta) = aR + \log \frac{b}{\psi}$.

Proof. For the first direction of the first result, consider any $\theta, \theta' \in \mathbb{R}$ and $\lambda \in [0, 1]$. We have that

$$U^{(q)}_{\mathbf{x}}(\mu_{\lambda\theta+(1-\lambda)\theta'}) - \left(\lambda U^{(q)}_{\mathbf{x}}(\mu_\theta) - (1-\lambda) \log U^{(q)}_{\mathbf{x}}(\mu_{\theta'}\right) = \log \frac{\Psi^{(q)}_{\mathbf{x}}(\mu_\theta)^\lambda \Psi^{(q)}_{\mathbf{x}}(\mu_{\theta'})^{1-\lambda}}{\Psi^{(q)}_{\mathbf{x}} \mu_{\lambda\theta+(1-\lambda)\theta'}\big)} \tag{6.94}$$

so it suffices to show that $\Psi^{(q)}_{\mathbf{x}}(\mu_{\lambda\theta+(1-\lambda)\theta'}) \geqslant \Psi^{(q)}_{\mathbf{x}}(\mu_\theta)^\lambda \Psi^{(q)}_{\mathbf{x}}(\mu_{\theta'})^{1-\lambda}$. By the log-concavity of f we have

$$\begin{aligned} \mu_{\lambda\theta+(1-\lambda)\theta'}(\lambda x + (1-\lambda)y) = f(\lambda(x-\theta) + (1-\lambda)(y-\theta')) &\geqslant f(x-\theta)^\lambda f(y-\theta')^{1-\lambda} \\ &= \mu_\theta(x)^\lambda \mu_{\theta'}(y)^{1-\lambda} \end{aligned} \tag{6.95}$$

for all $x, y \in \mathbb{R}$. Therefore by the Prékopa-Leindler inequality we have that

$$\begin{aligned} \Psi^{(q)}_{\mathbf{x}}(\mu_{\lambda\theta+(1-\lambda)\theta'}) &= \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \mu_{\lambda\theta+(1-\lambda)\theta'}(x)dx \\ &\geqslant \left(\int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \mu_\theta(x)dx\right)^\lambda \left(\int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \mu_{\theta'}(x)dx\right)^{1-\lambda} \\ &= \Psi^{(q)}_{\mathbf{x}}(\mu_\theta)^\lambda \Psi^{(q)}_{\mathbf{x}}(\mu_{\theta'})^{1-\lambda} \end{aligned} \tag{6.96}$$

For the second direction, by assumption $\exists\ a < c, b > c$ s.t. $\sqrt{f(x)f(y)} > f(\frac{x+y}{2})\ \forall\ x, y \in [a, b]$, i.e. f is strictly log-convex on $[a, b]$. Let $\mathbf{x} \in \mathbb{R}^n$ be any dataset s.t. $\mathbf{x}_{[\lfloor qn \rfloor+1]} - \mathbf{x}_{[\lfloor qn \rfloor]} \leqslant \frac{b-a}{2}$ and

set $\theta = \mathbf{x}_{[\lfloor qn \rfloor]} - a$, $\theta' = \mathbf{x}_{[\lfloor qn \rfloor]} - \frac{a+b}{2}$. Then we have

$$\begin{aligned}
\sqrt{\int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \mu_\theta(x)dx \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \mu_{\theta'}(x)dx} &= \sqrt{\int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \sqrt{\mu_\theta(x)}^2 dx \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \sqrt{\mu_{\theta'}(x)}^2 dx} \\
&\geqslant \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \sqrt{\mu_\theta(x)\mu_{\theta'}(x)}dx \\
&= \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \sqrt{f(x-\theta)f(x-\theta')}dx \\
&> \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} f\left(x - \frac{\theta+\theta'}{2}\right)dx = \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \mu_{\frac{\theta+\theta'}{2}}(x)dx
\end{aligned} \tag{6.97}$$

where the first inequality is Hölder's and the second is due to the strict log-convexity of f on $[a, b]$. Taking the logarithm of both sides followed by their negatives completes the proof.

Finally, for the second result, since f is centered and log-concave, by Cule and Samworth [2010, Lemma 1] there exist constants $C, c > 0$ s.t. $\mu_\theta(x) \leqslant C\exp(-c|x-\theta|)\ \forall\ \theta \in \mathbb{R}$. Let $\mathbf{x} = \begin{pmatrix}\theta + R - n\psi & \theta + R - (n-1)\psi & \cdots & \theta + R - 2\psi & \theta + R - \psi\end{pmatrix}$, so that $|\mathbf{x}_{[\lfloor qn \rfloor]} - \theta| \geqslant |\mathbf{x}_{[1]} - \theta| = R - n\psi \geqslant \frac{R}{2}$. Then

$$\Psi_{\mathbf{x}}^{(q)}(\mu_\theta) = \int_{\mathbf{x}_{[\lfloor qn \rfloor]}}^{\mathbf{x}_{[\lfloor qn \rfloor+1]}} \mu_\theta(x)dx \leqslant C\psi\exp(-c|\mathbf{x}_{[\lfloor qn \rfloor]} - \theta|) \leqslant C\psi\exp(-cR/2) \tag{6.98}$$

so $U_{\mathbf{x}}^{(q)}(\mu) = -\log\Psi_{\mathbf{x}}^{(q)}(\mu_\theta) \geqslant \log\frac{1}{C\psi} + \frac{cR}{2}$. □

Variants of the first result have been previously shown in the censored regression literature [Burridge, 1981, Pratt, 1981]. In fact, Burridge [1981] shows convexity of $U_{\mathbf{x}}^{(q)}(\mu_{\frac{\langle \mathbf{v}, \mathbf{f} \rangle}{\phi}, \frac{1}{\phi}})$ w.r.t. $(\mathbf{v}, \phi) \in \mathbb{R}^d \times \mathbb{R}_{>0}$, i.e. simultaneous learning of a feature map and inverse scale. Convexity of $U_{\mathbf{x}} = -\log\Psi_{\mathbf{x}} = \log\sum_{i=1}^m \frac{1}{\Psi_{\mathbf{x}}^{(q_i)}} = \log\sum_{i=1}^m \exp(-\log\Psi_{\mathbf{x}}^{(q_i)})$ follows because the log-sum-exp $\mathrm{LSE}(\boldsymbol{\ell}) = \log\sum_{i=1}^m e^{\boldsymbol{\ell}_{[i]}}$ is convex and non-decreasing in each argument. Note that for the converse direction, the dataset $\mathbf{x}$ is not a degenerate case; in-fact if f is strictly log-convex over an interval $[a, b]$ then any dataset whose optimal interval has length smaller than $\frac{b-a}{2}$ will yield a nonconvex $U_{\mathbf{x}}^{(q)}(\mu_\theta)$.

The case of the Laplacian

For the Laplace prior with $a = \mathbf{x}_{[\lfloor qn \rfloor]}$ and $b = \mathbf{x}_{[\lfloor qn \rfloor+1]}$ we have

$$\begin{aligned}
&-\log\Psi_{\mathbf{x}}^{(q)}(\mu_{\frac{\theta}{\phi}, \frac{1}{\phi}}) \\
&\quad = \log 2 - \log\left(\mathrm{sign}\left(b - \frac{\theta}{\phi}\right)\left(1 - e^{-\left|b - \frac{\theta}{\phi}\right|\phi}\right) - \mathrm{sign}\left(a - \frac{\theta}{\phi}\right)\left(1 - e^{-\left|a - \frac{\theta}{\phi}\right|\phi}\right)\right)
\end{aligned} \tag{6.99}$$

For $\theta < a\phi$ this simplifies to

$$\begin{aligned}\log 2 - \log\left(e^{\theta-a\phi} - e^{\theta-b\phi}\right) &= \log 2 - \log\left((e^{\frac{b-a}{2}\phi} - e^{\frac{a-b}{2}\phi})e^{\theta-\frac{a+b}{2}\phi}\right) \\ &= \left|\theta - \frac{a+b}{2}\phi\right| - \log\left(\sinh\left(\frac{b-a}{2}\phi\right)\right)\end{aligned} \tag{6.100}$$

and similarly for $\theta > b\phi$ it becomes

$$\begin{aligned}\log 2 - \log\left(e^{b\phi-\theta} - e^{a\phi-\theta}\right) &= \log 2 - \log\left((e^{\frac{b-a}{2}\phi} - e^{\frac{a-b}{2}\phi})e^{\frac{a+b}{2}\phi-\theta}\right) \\ &= \left|\frac{a+b}{2}\phi - \theta\right| - \log\left(\sinh\left(\frac{b-a}{2}\phi\right)\right)\end{aligned} \tag{6.101}$$

On the other hand for $\theta \in [a\phi, b\phi]$ it is

$$\begin{aligned}\log 2 - \log\left(2 - e^{-|b\phi-\theta|} - e^{-|a\phi-\theta|}\right) &= \log 2 - \log\left(2 - e^{\theta-b\phi} - e^{a\phi-\theta}\right) \\ &= \log 2 - \log\left(e^{-\frac{b-a}{2}\phi}\left(2e^{\frac{b-a}{2}\phi} - e^{\theta-\frac{a+b}{2}\phi} - e^{\frac{a+b}{2}\phi-\theta}\right)\right) \\ &= \frac{b-a}{2}\phi + \log 2 - \log\left(2e^{\frac{b-a}{2}\phi} - e^{\theta-\frac{a+b}{2}\phi} - e^{\frac{a+b}{2}\phi-\theta}\right) \\ &= \frac{b-a}{2}\phi - \log\left(e^{\frac{b-a}{2}\phi} - \cosh\left(\theta - \frac{a+b}{2}\phi\right)\right)\end{aligned} \tag{6.102}$$

Thus we have

$$U_{\mathbf{x}}^{(q)}(\mu_{\frac{\theta}{\phi},\frac{1}{\phi}}) = \begin{cases} \frac{b-a}{2}\phi - \log\left(\exp\left(\frac{b-a}{2}\phi\right) - \cosh\left(\theta - \frac{a+b}{2}\phi\right)\right) & \text{if} \quad \theta \in [a\phi, b\phi] \\ \left|\theta - \frac{a+b}{2}\phi\right| - \log\left(\sinh\left(\frac{b-a}{2}\phi\right)\right) & \text{else} \end{cases} \tag{6.103}$$

Suppose $\mathbf{x} \in [\pm B]^n$ and has the optimal interval has separation $\psi > 0$, $\frac{\theta}{\phi} \in [\pm B]$, and $\frac{1}{\phi} \in [\sigma_{\min}, \sigma_{\max}]$. Then $\phi \in [1/\sigma_{\max}, 1/\sigma_{\min}]$ and $\theta \in [\pm B/\sigma_{\min}]$, and so

$$U_{\mathbf{x}}^{(q)}(\mu_{\frac{\theta}{\phi},\frac{1}{\phi}}) \leqslant \frac{2B}{\sigma_{\min}} + \log\frac{2\sigma_{\max}}{\psi} \tag{6.104}$$

For $\theta \notin [a\phi, b\phi]$, the derivative w.r.t. θ always has magnitude 1. Within the interval, the derivative w.r.t. θ is $-\frac{\sinh(\frac{a+b}{2}\phi-\theta)}{\exp(\frac{b-a}{2}\phi)-\cosh(\theta-\frac{a+b}{2}\phi)}$, which attains its extrema at the endpoints $a\phi$ and $b\phi$, where its magnitude is also 1. Outside the interval, the derivative w.r.t. ϕ has magnitude

$$\begin{aligned}\left|\frac{a+b}{2}\operatorname{sign}\left(\frac{a+b}{2}\phi - \theta\right) - \frac{b-a}{2}\coth\left(\frac{b-a}{2}\phi\right)\right| &\leqslant \frac{|a+b|}{2} + \frac{b-a}{2}\coth\left(\frac{b-a}{2}\phi\right) \\ &\leqslant \frac{|a+b|}{2} + \frac{b-a}{2}\left(\frac{2/\phi}{(b-a)} + 1\right) \\ &= \frac{|a+b|}{2} + \frac{b-a}{2} + \frac{1}{\phi}\end{aligned} \tag{6.105}$$

while inside the interval the derivative w.r.t. ϕ is $\frac{b-a}{2} - \frac{(b-a)\exp(\frac{b-a}{2}\phi)-(a+b)\sinh(\frac{a+b}{2}\phi-\theta)}{2\left(\exp(\frac{b-a}{2}\phi)-\cosh(\frac{a+b}{2}\phi-\theta)\right)}$, which again attains its extrema at the endpoints $a\phi$ and $b\phi$, yielding magnitudes

$$\begin{aligned}\frac{b-a}{2} + \frac{b-a}{2}\left(\coth\left(\frac{b-a}{2}\phi\right)+1\right) + \frac{|a+b|}{2} &\leqslant \frac{b-a}{2}\left(\frac{2/\phi}{(b-a)}+3\right) + \frac{|a+b|}{2} \\ &\leqslant \frac{1}{\phi} + \frac{3}{2}(b-a) + \frac{|a+b|}{2}\end{aligned} \tag{6.106}$$

Thus we have

$$|\partial_\theta U_{\mathbf{x}}^{(q)}(\mu_{\frac{\theta}{\phi},\frac{1}{\phi}})| \leqslant 1 \qquad \text{and} \qquad |\partial_\phi U_{\mathbf{x}}^{(q)}(\mu_{\frac{\theta}{\phi},\frac{1}{\phi}})| \leqslant 4B + \sigma_{\max} \tag{6.107}$$

6.E.2 Public-private release

In this subsection we use $\boldsymbol{\ell}_{\mathbf{x}}(\boldsymbol{\theta},\boldsymbol{\phi}) = \left(\ell_{\mathbf{x}}^{(q_1)}(\boldsymbol{\theta}_{[1]},\boldsymbol{\phi}_{[1]})\cdots\ell_{\mathbf{x}}^{(q_m)}(\boldsymbol{\theta}_{[m]},\boldsymbol{\phi}_{[m]})\right)$ to refer to a vector whose ith entry is the loss $\ell_{\mathbf{x}}^{(q_i)}(\theta,\phi) = U_{\mathbf{x}}^{(q_i)}(\mu_{\frac{\theta}{\phi},\frac{1}{\phi}})$ associated with the ith of m quantiles $q_1,\ldots,q_m$. Furthermore, we use $\ell_{\mathbf{x}} = \mathrm{LSE}(\boldsymbol{\ell}_{\mathbf{x}})$ to refer to their log-sum-exp.

Guarantees

Theorem 6.E.2. Suppose for $N \geqslant n$ we have a private dataset $\mathbf{x} \sim \mathcal{D}^n$ and a public dataset $\mathbf{x}' \sim \mathcal{D}'^N$, both drawn from κ-bounded distributions over $[\pm B]$. Use i.i.d. draws from the public dataset to construct $T = \lfloor N/n \rfloor$ datasets $\mathbf{x}'_t \sim \mathcal{D}'^n$ and run online gradient descent on the resulting losses $\ell_{\mathbf{x}'_t}(\boldsymbol{\theta},\boldsymbol{\phi}) = \mathrm{LSE}(\boldsymbol{\ell}_{\mathbf{x}_t}(\boldsymbol{\theta},\boldsymbol{\phi}))$ over the parameter space $\boldsymbol{\theta} \in [\pm B/\sigma_{\min}]^m$ starting at $\boldsymbol{\theta} = \mathbf{0}_m$ and $\boldsymbol{\phi} \in [1/\sigma_{\max}, 1/\sigma_{\min}]^m$ starting at the midpoint, with stepsize $B\sqrt{\frac{m}{T}}$ for $\boldsymbol{\theta}$ and $\frac{\sigma_{\max}-\sigma_{\min}}{4B+\sigma_{\max}}\sqrt{\frac{m}{T}}$ for $\boldsymbol{\phi}$, obtaining iterates $(\boldsymbol{\theta}_1,\boldsymbol{\phi}_1),\ldots,(\boldsymbol{\theta}_T,\boldsymbol{\phi}_T)$. Return the priors $\mu_i = \mu_{\frac{\bar{\boldsymbol{\theta}}_{[i]}}{\bar{\boldsymbol{\phi}}_{[i]}},\frac{1}{\bar{\boldsymbol{\phi}}_{[i]}}}$ for $\bar{\boldsymbol{\theta}} = \frac{1}{T}\sum_{t=1}^T \boldsymbol{\theta}_t$ and $\bar{\boldsymbol{\phi}} = \frac{1}{T}\sum_{t=1}^T \boldsymbol{\phi}_t$ the average of these iterates. Then $\boldsymbol{\mu}' = \begin{pmatrix}\mu_1 & \cdots & \mu_m\end{pmatrix}$ satisfies

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}^n} U_{\mathbf{x}}(\boldsymbol{\mu}') \leqslant \min_{\boldsymbol{\mu}\in\mathrm{Lap}^m_{B,\sigma_{\min},\sigma_{\max}}} \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}^n} U_{\mathbf{x}}(\boldsymbol{\mu}) &+ 2\left(\frac{2B}{\sigma_{\min}} + \log\frac{4\kappa m(n+1)N\sigma_{\max}}{\beta'}\right)\mathrm{TV}_q(\mathcal{D},\mathcal{D}') \\ &+ (B + 4B\sigma_{\max} + \sigma_{\max}^2)\sqrt{\frac{m(n+1)}{N}} \\ &+ 2\left(\frac{4B}{\sigma_{\min}} + \log\frac{4\kappa m(n+1)N\sigma_{\max}}{\beta'}\right)\sqrt{\frac{2(n+1)}{N}\log\frac{4}{\beta'}} \\ &+ \frac{(n+1)\beta'}{N}\left(3 + \frac{4B}{\sigma_{\min}} + 4\log\frac{2\kappa(n+1)N\sqrt{2m\sigma_{\max}}}{\beta'}\right)\end{aligned} \tag{6.108}$$

where $\mathrm{Lap}_{B,\sigma_{\min},\sigma_{\max}}$ is the set of Laplace priors with locations in $[\pm B]$ and scales in $[\sigma_{\min},\sigma_{\max}]$.

Proof. Define ${\mathcal{D}'_\psi}^n$ to be the conditional distribution over $\mathbf{z} \sim {\mathcal{D}'}^n$ s.t. $\psi_\mathbf{z} \geqslant \psi$, with associated density $\rho'_\psi(\mathbf{z}) = \frac{\rho'(\mathbf{z})1_{\psi_\mathbf{z} \geqslant \psi}}{1-p'_\psi}$, where $p'_\psi = \int_{\psi_\mathbf{z} < \psi} \rho'(\mathbf{z}) \leqslant \kappa n^2 \psi$. Then we have for any $\boldsymbol{\mu}^* \in \mathrm{Lap}^m_{B,\sigma_{\min},\sigma_{\max}}$ that

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathbf{z}\sim\mathcal{D}^n} U_\mathbf{x}(\boldsymbol{\mu}') &= \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathcal{D}^n} U_\mathbf{z}(\boldsymbol{\mu}') - \mathop{\mathbb{E}}_{\mathbf{z}\sim{\mathcal{D}'}^n} U_\mathbf{z}(\boldsymbol{\mu}') + \mathop{\mathbb{E}}_{\mathbf{z}\sim{\mathcal{D}'}^n} U_\mathbf{z}(\boldsymbol{\mu}') - \mathop{\mathbb{E}}_{\mathbf{z}\sim{\mathcal{D}'_\psi}^n} U_\mathbf{z}(\boldsymbol{\mu}') + \mathop{\mathbb{E}}_{\mathbf{z}\sim{\mathcal{D}'_\psi}^n} U_\mathbf{z}(\boldsymbol{\mu}') \\
&\leqslant \int U_\mathbf{z}(\boldsymbol{\mu}')(\rho(\mathbf{z}) - \rho'(\mathbf{z})) + \int U_\mathbf{z}(\boldsymbol{\mu}')(\rho'(\mathbf{z}) - \rho'_\psi(\mathbf{x})) + \mathop{\mathbb{E}}_{\mathbf{z}\sim{\mathcal{D}'_\psi}^n} U_\mathbf{z}(\boldsymbol{\mu}^*) + \mathcal{E}_\psi \\
&\leqslant \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathcal{D}^n} U_\mathbf{x}(\boldsymbol{\mu}^*) + \int (U_\mathbf{z}(\boldsymbol{\mu}') + U_\mathbf{x}(\boldsymbol{\mu}^*))|\rho(\mathbf{z}) - \rho'(\mathbf{x})| \\
&\quad + \int (U_\mathbf{z}(\boldsymbol{\mu}') + U_\mathbf{z}(\boldsymbol{\mu}^*))|\rho'(\mathbf{z}) - \rho'_\psi(\mathbf{z})| + \mathcal{E}_\psi
\end{aligned} \tag{6.109}$$

where $\mathcal{E}_\psi$ is the error of running online gradient descent with the specified step-sizes on samples $\mathbf{z}'_t \sim {\mathcal{D}'_\psi}^n$ for $t = 1, \ldots, T$. Now if $\mathbf{z}$ has entries drawn i.i.d. from a κ-bounded distribution $\mathcal{D}^n$ (or ${\mathcal{D}'}^n$), then we have that

$$\int_0^\psi \rho_{\psi_\mathbf{z}}(y)dy = \Pr(\psi_\mathbf{z} \leqslant \psi : \mathbf{z} \sim \mathcal{D}^n) \leqslant n(n-1) \max_{z\in\mathbb{R}} \Pr(|z - z'| \leqslant \psi : z' \sim \mathcal{D}) \leqslant \kappa n^2 \psi \tag{6.110}$$

where $\rho_{\psi_\mathbf{z}}$ is the density of $\psi_\mathbf{z}$ for $\mathbf{z} \sim \mathcal{D}^n$ (not to be confused with the conditional density ρ_ψ over $\mathbf{z}$); the same holds for the analog $\rho'_{\psi_\mathbf{z}}$ for ${\mathcal{D}'}^n$. Since this holds for all $\psi \geqslant 0$ and $\log \frac{1}{y}$ is monotonically decreasing on $y > 0$, this means the worst-case measure that $\rho_{\psi_\mathbf{z}}$ can be is constant over $[0, \psi]$ and thus $\int_0^\psi \rho_{\psi_\mathbf{z}}(y) \log \frac{1}{y} dy \leqslant \kappa n^2 \int_0^\psi \log \frac{1}{y} dy = \kappa n^2 \psi(1 + \log \frac{1}{\psi})$, and similarly for $\rho'_{\psi_\mathbf{z}}$.

To bound the first integral, note that $U_\mathbf{z} = \mathrm{LSE}(\boldsymbol{\ell}_\mathbf{z}) \leqslant \max_i U_\mathbf{z}^{(q_i)} + \log m \leqslant \frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi_\mathbf{z}}$ and that the r.v. $\psi_\mathbf{z}$ depends only on the joint distribution over the order statistics of $\mathcal{D}^n$ and ${\mathcal{D}'}^m$. Then we have

$$\begin{aligned}
&\int (U_\mathbf{z}(\boldsymbol{\mu}') + U_\mathbf{z}(\boldsymbol{\mu}^*))|\rho(\mathbf{z}) - \rho'(\mathbf{z})| \\
&\qquad \leqslant \int \left(\frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi_\mathbf{z}}\right) |\rho(\mathbf{z}) - \rho'(\mathbf{z})| \\
&\qquad \leqslant 2\left(\frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi}\right) \mathrm{TV}_q(\mathcal{D}, \mathcal{D}') + \int_{\psi_\mathbf{z} < \psi} |\rho(\mathbf{z}) - \rho'(\mathbf{z})| \log \frac{1}{\psi_\mathbf{z}} \\
&\qquad \leqslant 2\left(\frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi}\right) \mathrm{TV}_q(\mathcal{D}, \mathcal{D}') + \int_0^\psi (\rho_{\psi_\mathbf{z}}(y) + \rho'_{\psi_\mathbf{z}}(y)) \log \frac{1}{y} dy \\
&\qquad \leqslant 2\left(\frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi}\right) \mathrm{TV}_q(\mathcal{D}, \mathcal{D}') + 2\kappa n^2 \psi \left(1 + \log \frac{1}{\psi}\right)
\end{aligned} \tag{6.111}$$

For the second integral we have for $p'_\psi = \int_{\psi_{\mathbf{z}}<\psi} \rho'(\mathbf{z}) \leqslant \kappa n^2 \psi$ that

$$\begin{aligned}
&\int (U_{\mathbf{z}}(\boldsymbol{\mu}') + U_{\mathbf{z}}(\boldsymbol{\mu}^*))|\rho'(\mathbf{z}) - \rho'_\psi(\mathbf{z})| \\
&\qquad = \int_{\psi_{\mathbf{z}} \geqslant \psi} (U_{\mathbf{x}}(\boldsymbol{\mu}') + U_{\mathbf{z}}(\boldsymbol{\mu}^*)) \left| \rho'(\mathbf{z}) - \frac{\rho'(\mathbf{z})}{1 - p'_\psi} \right| + \int_{\psi_{\mathbf{z}}<\psi} (U_{\mathbf{z}}(\boldsymbol{\mu}') + U_{\mathbf{z}}(\boldsymbol{\mu}^*))\rho'(\mathbf{z}) \\
&\qquad = \frac{2p'_\psi}{1 - p'_\psi} \int_{\psi_{\mathbf{z}} \geqslant \psi} \left(\frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi} \right) \rho'(\mathbf{z}) + \int_{\psi_{\mathbf{z}}<\psi} \left(\frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi_{\mathbf{z}}} \right) \rho'(\mathbf{z}) \\
&\qquad = 2p'_\psi \left(\frac{4B}{\sigma_{\min}} + \log \frac{4m^2\sigma^2_{\max}}{\psi} \right) + \int_{\psi_{\mathbf{z}}<\psi} \rho'(\mathbf{z}) \log \frac{1}{\psi_{\mathbf{z}}} \\
&\qquad \leqslant 2\kappa n^2 \psi \left(\frac{4B}{\sigma_{\min}} + \log \frac{4m^2\sigma^2_{\max}}{\psi} \right) + \kappa n^2 \psi \left(1 + \log \frac{1}{\psi} \right)
\end{aligned} \tag{6.112}$$

Finally, we bound $\mathcal{E}_\psi$. By κ-boundedness of $\mathcal{D}'$, the probability that $\exists\, t \in [T]$ s.t. $\psi_{\mathbf{z}'_t} < \psi\ \forall\, t \in [T]$ is at most $\kappa n^2 T \psi$, so if we set $\psi = \frac{\beta'}{2\kappa n^2 T}$ then w.p. $\geqslant 1 - \beta'/2$ the sampling $\mathbf{z}'_t$ from $\mathbf{x}'$ as specified is equivalent to rejection sampling from ${\mathcal{D}'_\psi}^n$, on which the functions $U_{\mathbf{z}}$ are bounded by $\frac{2B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi}$. Therefore with probability $\geqslant 1 - \beta'/2$ by Shalev-Shwartz [2011, Theorem 2.21] and Lemma B.4.1 we have that w.p. $1 - \beta'/2$

$$\begin{aligned}
\mathcal{E}_\psi &\leqslant (B + (\sigma_{\max} - \sigma_{\min})(4B + \sigma_{\max}))\sqrt{\frac{m}{T}} + 2 \left(\frac{4B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi} \right) \sqrt{\frac{2}{T} \log \frac{4}{\beta'}} \\
&= (B + 4B\sigma_{\max} + \sigma^2_{\max})\sqrt{\frac{m(n+1)}{N}} + 2 \left(\frac{4B}{\sigma_{\min}} + \log \frac{2m\sigma_{\max}}{\psi} \right) \sqrt{\frac{2(n+1)}{N} \log \frac{4}{\beta'}}
\end{aligned} \tag{6.113}$$

Combining terms and substituting the selected value for ψ yields the result. □

Experimental details

We evaluate on Adult ("age" and "hours" categories) and Goodreads ("rating" and "page count" categories) data. For the former we use the train and test sets as the public and private data, respectively, while for the latter we use the "History" and "Poetry" genres as the public and private data, respectively. Public data is used to fit Laplace location and scale parameters using the COCOB optimizer run until progress stops. We use the implementation here: `https://github.com/anandsaha/nips.cocob.pytorch`. All evaluations are averages of forty trials.

We use the following reasonable guesses for locations ν, scales σ, and quantile ranges $[a, b]$:

- age: $\nu = 40$, $\sigma = 5$, $a = 10$, $b = 120$
- hours: $\nu = 40$, $\sigma = 2$, $a = 0$, $b = 168$
- rating: $\nu = 2.5$, $\sigma = 0.5$, $a = 0$, $b = 5$
- page count: $\nu = 200$, $\sigma = 25$, $a = 0$, $b = \frac{1000}{1-q}$

Note that, here and elsewhere, using q-dependent range for b only helps the uniform prior, which is the baseline. The scales σ are used to set the scale parameter of the Cauchy distribution for `public quantiles`—its location is fixed by the public quantiles. Meanwhile the locations ν are used to set to *scale* parameter of the half-Cauchy prior used to mix with `PubFit` for robustness (using coefficient 0.1 on the robust prior). We choose this prior because the data are all nonnegative.

6.E.3 Sequential release

In this subsection we use $\boldsymbol{\ell}_{\mathbf{x},\mathbf{f}}(\mathbf{V}, \boldsymbol{\phi}) = \left(\ell^{(q_1)}_{\mathbf{x},\mathbf{f}}(\mathbf{V}_{[1]}, \boldsymbol{\phi}_{[1]}) \cdots \ell^{(q_m)}_{\mathbf{x},\mathbf{f}}(\mathbf{V}_{[m]}, \boldsymbol{\phi}_{[m]})\right)$ to refer to a vector whose ith entry is the loss $\ell^{(q_i)}_{\mathbf{x},\mathbf{f}}(\mathbf{v}, \phi) = U^{(q_i)}_{\mathbf{x}}(\mu_{\frac{\langle \mathbf{v},\mathbf{f}\rangle}{\phi}, \frac{1}{\phi}})$ associated with the ith of m quantiles.

Guarantees

Theorem 6.E.3. Consider a sequence of datasets $\mathbf{x}_t \in [\pm R]^{n_t}$ and associated feature vectors $\mathbf{f}_t \in [\pm F]^d$. Suppose we set the components of $\boldsymbol{\mu}_t$ as the Laplace priors $\boldsymbol{\mu}_{t[i]} = \mu_{\frac{\langle \mathbf{V}_{t[i]}\mathbf{f}_t\rangle}{\boldsymbol{\phi}_{t[i]}}, \frac{1}{\boldsymbol{\phi}_{t[i]}}}$, where $\mathbf{V}_t \in [\pm B/\sigma_{\min}]^{m\times d}$ and $\boldsymbol{\phi}_t \in [1/\sigma_{\max}, 1/\sigma_{\min}]^m$ are determined by separate runs of DP-FTRL with budgets $(\varepsilon'/2, \delta'/2)$ and step-sizes $\eta_1 = \frac{B}{F\sigma_{\min}}\sqrt{\frac{2m\varepsilon'_1}{\lceil\log_2(T+1)\rceil T\left(1+\sqrt{2md\log\frac{T}{\beta'}\log\frac{1}{\delta'}}\right)}}$, and $\eta_2 = \frac{1/\sigma_{\min}}{B+\sigma_{\max}}\sqrt{\frac{m\varepsilon'_2}{2\lceil\log_2(T+1)\rceil T\left(1+\sqrt{2m\log\frac{T}{\beta'}\log\frac{1}{\delta'}}\right)}}$. Then we have regret

$$\begin{aligned}\max_{\substack{\mathbf{W}\in[\pm B]^{m\times d}\\ \boldsymbol{\sigma}\in[\sigma_{\min},\sigma_{\max}]^m}} &\sum_{t=1}^{T} U_{\mathbf{x}_t}(\boldsymbol{\mu}_t) - U_{\mathbf{x}_t}\left(\left(\mu_{\langle \mathbf{W}_{[1]},\mathbf{f}_t\rangle,\boldsymbol{\sigma}_{[1]}} \cdots \mu_{\langle \mathbf{W}_{[m]},\mathbf{f}_t\rangle,\boldsymbol{\sigma}_{[m]}}\right)\right)\\ &\leqslant \frac{B(F+1)+\sigma_{\max}}{\sigma_{\min}}\sqrt{md\lceil\log_2(T+1)\rceil T\left(4+\frac{8}{\varepsilon'}\sqrt{2md\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)}\end{aligned} \tag{6.114}$$

For sufficiently small ε' (including $\varepsilon' \leqslant 1$) we can instead simplify the regret to

$$\frac{4}{\sigma_{\min}}\left(BFd^{\frac{3}{4}} + B + \sigma_{\max}\right)\sqrt{\frac{m\lceil\log_2(T+1)\rceil T}{\varepsilon'}\sqrt{2m\log\frac{T}{\beta'}\log\frac{2}{\delta'}}} \tag{6.115}$$

Proof. Note that

$$\sum_{j=1}^{m}\|\nabla_{\mathbf{V}_{[j]}}\operatorname{LSE}(\boldsymbol{\ell}_{\mathbf{x}_t,\mathbf{f}_t})\|_2^2 \leqslant \|\mathbf{f}_t\|_2^2\sum_{j=1}^{m}\left(\frac{\exp(\ell^{(q_j)}_{\mathbf{x}_t,\mathbf{f}_t})}{\sum_{i=1}^m\exp(\ell^{(q_i)}_{\mathbf{x}_t,\mathbf{f}_t})}\right)^2 \leqslant F^2 d \tag{6.116}$$

and

$$\sum_{j=1}^{m}(\partial_{\boldsymbol{\phi}_{[j]}}\operatorname{LSE}(\boldsymbol{\ell}_{\mathbf{x}_t,\mathbf{f}_t}))^2 \leqslant (4B+\sigma_{\max})^2\sum_{j=1}^{m}\left(\frac{\exp(\ell^{(q_j)}_{\mathbf{x}_t,\mathbf{f}_t})}{\sum_{i=1}^m\exp(\ell^{(q_i)}_{\mathbf{x}_t,\mathbf{f}_t})}\right)^2 \leqslant (4B+\sigma_{\max})^2 \tag{6.117}$$

and so applying Theorem 6.5.1 twice with the assumed budgets and step-sizes yields

$$
\begin{aligned}
&\max_{\substack{\mathbf{W}\in[\pm B]^{m\times d}\\ \boldsymbol{\sigma}\in[\sigma_{\min},\sigma_{\max}]^m}} \sum_{t=1}^{T} U_{\mathbf{x}_t}(\boldsymbol{\mu}_t) - U_{\mathbf{x}_t}\left(\left(\mu_{\langle \mathbf{W}_{[1]},\mathbf{f}_t\rangle,\boldsymbol{\sigma}_{[1]}} \cdots \mu_{\langle \mathbf{W}_{[m]},\mathbf{f}_t\rangle,\boldsymbol{\sigma}_{[m]}}\right)\right)\\
&= \max_{\substack{\mathbf{V}\in[\pm B/\sigma_{\min}]^d\\ \phi_i\in[\frac{1}{\sigma_{\max}},\frac{1}{\sigma_{\min}}]}} \sum_{t=1}^{T} \mathrm{LSE}(\boldsymbol{\ell}_{\mathbf{x}_t,\mathbf{f}_t}(\mathbf{V}_t,\boldsymbol{\phi}_t)) - \mathrm{LSE}(\boldsymbol{\ell}_{\mathbf{x}_t,\mathbf{f}_t}(\mathbf{V},\boldsymbol{\phi}))\\
&\leqslant \frac{m(2B/\sigma_{\min})^2}{2\eta_1} + \eta_1\lceil\log_2(T+1)\rceil T\left(1+\frac{2}{\varepsilon'}\sqrt{2md\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)\sum_{j=1}^{m}\|\nabla_{\mathbf{V}_{[j]}}\mathrm{LSE}(\boldsymbol{\ell}_{\mathbf{x}_t,\mathbf{f}_t})\|_2^2\\
&\quad + \frac{m(\frac{1}{\sigma_{\min}}-\frac{1}{\sigma_{\max}})^2}{2\eta_2} + \eta_2\lceil\log_2(T+1)\rceil T\left(1+\frac{2}{\varepsilon'}\sqrt{2m\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)\sum_{j=1}^{m}(\partial_{\boldsymbol{\phi}_{[j]}}\mathrm{LSE}(\boldsymbol{\ell}_{\mathbf{x}_t,\mathbf{f}_t}))^2\\
&\leqslant \frac{2B^2md}{\eta_1\sigma_{\min}^2} + \eta_1\lceil\log_2(T+1)\rceil TF^2d\left(1+\frac{2}{\varepsilon'}\sqrt{2md\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)\\
&\quad + \frac{m}{2\eta_2\sigma_{\min}^2} + \eta_2\lceil\log_2(T+1)\rceil T(B+\sigma_{\max})^2\left(1+\frac{2}{\varepsilon'}\sqrt{2m\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)\\
&\leqslant \frac{2BF}{\sigma_{\min}}\sqrt{2md\lceil\log_2(T+1)\rceil T\left(1+\frac{2}{\varepsilon'}\sqrt{2md\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)}\\
&\quad + \frac{2}{\sigma_{\min}}(B+\sigma_{\max})\sqrt{2m\lceil\log_2(T+1)\rceil T\left(1+\frac{2}{\varepsilon'}\sqrt{2m\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)}\\
&\leqslant \frac{2}{\sigma_{\min}}\left(B(F+1)+\sigma_{\max}\right)\sqrt{md\lceil\log_2(T+1)\rceil T\left(1+\frac{2}{\varepsilon'}\sqrt{2md\log\frac{T}{\beta'}\log\frac{2}{\delta'}}\right)}
\end{aligned}
\tag{6.118}
$$

□

Experimental details

For sequential release we consider the following tasks:

- Synthetic is a stationary dataset generation scheme in which we randomly sample a one standard Gaussian vector $\mathbf{a}$ for each feature dimension (we use ten) and another $\mathbf{b}$ of size $m+2$, which we sort. On each day t of T we sample the public feature vector $\mathbf{f}_t$, also from a standard normal, and the "ground truth" quantiles q_i on that day are then set by $\langle\mathbf{a},\mathbf{f}_t\rangle+\mathbf{b}_{[i+1]}$. We generate the actual data by sampling from the uniform distributions on $[\langle\mathbf{a},\mathbf{f}_t\rangle+\mathbf{b}_{[i]},\langle\mathbf{a},\mathbf{f}_t\rangle+\mathbf{b}_{[i+1]}]$. The number of points we sample is determined by $\lfloor 100/(m+1)\rfloor$ plus different Poisson-distributed random variable for each; in the "noiseless" setting used in Figure 6.4 (left) the Poisson's scale is zero, so the "ground truth" quantiles are

correct for the dataset, while for Figure 6.5 (left) we use a Poisson with scale five. For the noiseless setting we use 100K timesteps, while for the noisy setting we use 2500.

- CitiBike consists of data downloaded from here: `https://s3.amazonaws.com/tripdata/index.html`, We take the period from September 2015 through November 2022, which is roughly 2500 days, although days with less than ten trips—seemingly data errors—are ignored. For each day we include a feature vector containing seven dimensions for the day of the week, one dimension for a sinusoidal encoding of the day of the year, and six weather features from the Central Park station downloaded from here `https://www.ncei.noaa.gov/cdo-web/`, specifically average wind speed, precipitation, snowfall, snow depth, maximum temperature, and minimum temperature. These are scaled to lie within similar ranges.
- BBC consists of Reddit's `worldnews` subreddit corpus downloaded from the following link: `https://zissou.infosci.cornell.edu/convokit/datasets/subreddit-corpus/corpus-zipped/`. We find all conversations corresponding to a post of a BBC article, specified by the domain `bbc.co.uk`, and collect those with at least ten comments. We compute the Flesch readability score of each comment using the package here `https://github.com/textstat/textstat`. The datasets for computing quantiles are then the collection of scores for each headline; the size is roughly 10K, corresponding to articles between 2008 and 2018. As features we combine a seven-dimensional day-of-the-week encoding, sinusoidal features for the day of the year and the time of day of the post, information about the post itself (whether it is gilded, its own Flesch score, and the number of tokens), and finally a 25-dimensional embedding of the title, set using a normalized sum of GloVe embeddings [Pennington et al., 2014] of the tokens, excluding English stop-words via NLTK [Loper and Bird, 2002].

We again use reasonable guesses of data information to set the static priors, and to initialize the learning schemes.

- Synthetic: $\nu = 0, \sigma = 1, a = -100, b = 100$
- CitiBike: $\nu = 10, \sigma = 1, a = 0, b = 50/(1-q)$
- BBC: $\nu = 50, \sigma = 10, a = -100 - 100/(1-q), b = 100 + 100q$

We use a and b for the static uniform distributions, ν and σ for the static Cauchy distributions, in the case of nonnegative data (CitiBike) we use ν for the *scale* of the half-Cauchy distribution, and for the learning schemes we initialize their Laplace priors to be centered at ν with scale σ. We again use the COCOB optimizer for non-private and proxy learning, and for robustness we mix with the Cauchy (or half-Cauchy for nonnegative data) with coefficient 0.1 on the robust prior. For the `PubPrev` method, we set its scale using σ. For DP-FTRL, we heavily tune it to show the possibility of learning on the synthetic task; the implementation is adapted from the one here: `https://github.com/google-research/DP-FTRL`. All results are reported as averages over forty trials.

Algorithm 13: ApproximateQuantiles with predictions

Input: sorted unrepeated data $\mathbf{x} \in (a,b)^n$, ordered quantiles $q_1, \ldots, q_m \in (0,1)$, priors $\mu_1, \ldots, \mu_m : \mathbb{R} \mapsto \mathbb{R}_{\geqslant 0}$, prior adaptation rule $r \in \{\texttt{conditional}, \texttt{edge}\}$, privacy parameters $\varepsilon_1, \ldots, \varepsilon_m > 0$, branching factor $K \geqslant 2$

`// runs single-quantile algorithm on datapoints` $\hat{\mathbf{x}}$

Method `quantile` $(\hat{\mathbf{x}}, q, \varepsilon, \mu)$ **:**

  Output: $o \in (a,b)$ w.p. $\propto \exp(-\varepsilon \operatorname{Gap}_q(\hat{\mathbf{x}}, o)/2)\mu(o)$

Method `recurse` $(\mathbf{j}, \underline{q}, \overline{q}, \hat{a}, \hat{b})$ **:**

  `// determines` $K-1$ `indices` $\mathbf{i}$ `whose quantiles to compute at this node`

  if $|\mathbf{j}| \geqslant K$ **then**

    $\mathbf{i} \leftarrow \left(\mathbf{j}_{[\lceil |\mathbf{j}|/K \rceil]}, \cdots, \mathbf{j}_{[\lceil (K-1)|\mathbf{j}|/K \rceil]}\right)$

  else

    $\mathbf{i} \leftarrow \mathbf{j}$

  `// restricts dataset to the interval` $(\hat{a}, \hat{b})$

  $\underline{k}_{\mathbf{i}} \leftarrow \min_{\mathbf{x}_{[k]} > \hat{a}} k$

  $\overline{k}_{\mathbf{i}} \leftarrow \max_{\mathbf{x}_{[k]} < \hat{b}} k$

  $\hat{\mathbf{x}}_{\mathbf{i}} \leftarrow \left(\mathbf{x}_{[\underline{k}_{\mathbf{i}}]}, \cdots, \mathbf{x}_{[\overline{k}_{\mathbf{i}}]}\right)$

  `// sets relative quantiles` $\tilde{q}_i$ `and restricts priors to the interval` $[\hat{a}, \hat{b}]$

  for $j = 1, \ldots, |\mathbf{i}|$ **do**

    $\tilde{q}_{\mathbf{i}_{[j]}} \leftarrow (q_{\mathbf{i}_{[j]}} - \underline{q})/(\overline{q} - \underline{q})$

    if $r = \texttt{conditional}$ **then**

      $\hat{\mu}_{\mathbf{i}_{[j]}}(o) \leftarrow \frac{\mu_{\mathbf{i}_{[j]}}(o)}{\mu_{\mathbf{i}_{[j]}}([\hat{a},\hat{b}])} 1_{o \in [\hat{a},\hat{b}]}$

    else

      $\hat{\mu}_{\mathbf{i}_{[j]}}(o) \leftarrow \mu_{\mathbf{i}_{[j]}}(o) 1_{o \in (\hat{a},\hat{b})} + \mu_{\mathbf{i}_{[j]}}((-\infty, \hat{a}])\delta(o - \hat{a}) + \mu_{\mathbf{i}_{[j]}}([\hat{b}, \infty))\delta(o - \hat{b})$

  `// computes` $K-1$ `quantiles` $\mathbf{o_i}$ `and sorts the results`

  $\mathbf{o_i} \leftarrow \left(\texttt{quantile}(\hat{\mathbf{x}}_{\mathbf{i}}, \tilde{q}_{\mathbf{i}_{[1]}}, \varepsilon_{\mathbf{i}_{[1]}}/|\mathbf{i}|, \hat{\mu}_{\mathbf{i}_{[1]}}), \cdots, \texttt{quantile}(\hat{\mathbf{x}}_{\mathbf{i}}, \tilde{q}_{\mathbf{i}_{[|\mathbf{i}|]}}, \varepsilon_{\mathbf{i}_{[|\mathbf{i}|]}}/|\mathbf{i}|, \hat{\mu}_{\mathbf{i}_{[|\mathbf{i}|]}})\right)$

  $\mathbf{o_i} \leftarrow \texttt{sort}(\mathbf{o_i})$

  `// recursively computes remaining indices on the` K `intervals induced by` $\mathbf{o_i}$

  if $|\mathbf{j}| < K$ **then**

    $\mathbf{o} \leftarrow \mathbf{o}_i$

  else

    $\mathbf{o} \leftarrow \texttt{concat}(\texttt{recurse}(\left(\mathbf{j}_{[1]}, \cdots, \mathbf{j}_{[\lceil |\mathbf{j}|/K \rceil - 1]}\right), \underline{q}, q_{\mathbf{i}_{[1]}}, \hat{a}, \mathbf{o}_{[1]}), \left(\mathbf{o}_{[1]}\right))$

    for $j = 2, \ldots, |\mathbf{i}|$ **do**

      $\mathbf{o} \leftarrow \texttt{concat}(\mathbf{o}, \texttt{recurse}(\left(\mathbf{j}_{[\lceil (j-1)|\mathbf{j}|/K \rceil + 1]}, \cdots, \mathbf{j}_{[\lceil j|\mathbf{j}|/K \rceil - 1]}\right), q_{\mathbf{i}_{[j-1]}}, q_{\mathbf{i}_{[j]}}, \mathbf{o}_{[j-1]}, \mathbf{o}_{[j]}))$

      $\mathbf{o} \leftarrow \texttt{concat}(\mathbf{o}, \left(\mathbf{o}_{[j]}\right))$

    $\mathbf{o} \leftarrow \texttt{concat}(\mathbf{o}, \texttt{recurse}(\left(\mathbf{j}_{[\lceil (K-1)|\mathbf{j}|/K \rceil + 1]}, \cdots, \mathbf{j}_{[|\mathbf{j}|]}\right), q_{\mathbf{i}_{[K-1]}}, \overline{q}, \mathbf{o}_{[K-1]}, \hat{b}))$

  Output: $\mathbf{o}$

Output: `recurse` $\left(1, \cdots, m\right), 0, 1, -\infty, \infty)$

Algorithm 14: SeparateCov with predictions (zCDP)

Input: data $\mathbf{X} \in \mathbb{R}^{d \times n}$, symmetric prediction $\mathbf{W} \in \mathbb{R}^{d \times d}$, privacy parameter $\rho > 0$

$\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \leftarrow \mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$

$\hat{\mathbf{\Lambda}} \leftarrow \mathbf{\Lambda} + \mathrm{diag}(\mathbf{z}), \mathbf{z} \sim \mathcal{N}(\mathbf{0}_d, \frac{2}{\rho n^2})$ `// add noise to error eigenvalues`

$\tilde{\mathbf{C}} \leftarrow \mathbf{X}\mathbf{X}^\top/n + \mathbf{Z}$ for $\mathbf{Z}_{[i,j]} = \mathbf{Z}_{[j,i]} \sim \mathcal{N}\left(0, \frac{2}{\rho n^2}\right)$

$\tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top \leftarrow \tilde{\mathbf{C}} - \mathbf{W}$ `// get eigenvectors of noised prediction error`

Output: $\hat{\mathbf{C}} = \tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\top + \mathbf{W}$ `// combine to estimate` $\mathbf{X}\mathbf{X}^\top/n - \mathbf{W}$ `and add` $\mathbf{W}$

Algorithm 15: IterativeEigenvectorSampling with predictions

Input: data matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$, symmetric prediction $\mathbf{W} \in \mathbb{R}^{d \times d}$, privacy parameters $\varepsilon_0^{(\pm 1)}, \ldots, \varepsilon_d^{(\pm 1)}$

initialize $\hat{\mathbf{C}} \leftarrow \mathbf{W}$

for $s, \mathbf{C} \in ((1, \{\mathbf{X}\mathbf{X}^\top - \mathbf{W}\}_+), (-1, \{\mathbf{W} - \mathbf{X}\mathbf{X}^\top\}_+))$ **do**

  `// run the original IterativeEigenvectorSampling on` $\mathbf{C}$`, then add` s `times the result to` $\hat{\mathbf{C}}$

  initialize $\mathbf{C}_1 \leftarrow \mathbf{C}$ and $\mathbf{P}_1 \leftarrow \mathbf{I}_d$

  $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \leftarrow \mathbf{C}$ `// get eigenvalues of` $\mathbf{C}$

  for *i=1,...,d* **do**

    $\lambda_i^{(s)} \leftarrow \mathbf{\Lambda}_{[i]} + \mathrm{Lap}(2/\varepsilon_0^{(s)})$

    $\hat{\boldsymbol{\theta}}_i^{(s)} \leftarrow \mathbf{P}_i^\top \hat{\mathbf{u}}_i^{(s)}$ for $\hat{\mathbf{u}}_i^{(s)}$ sampled w.p. $\propto f_{\mathbf{C}_i}(\mathbf{u}) = \exp\left(\frac{\varepsilon_i^{(s)}}{4}\mathbf{u}^\top \mathbf{C}_i \mathbf{u}\right)$

    set $\mathbf{P}_{i+1} \in \mathbb{R}^{(d-i) \times d}$ to be an orthonormal basis orthogonal to $\hat{\boldsymbol{\theta}}_1^{(s)}, \ldots, \hat{\boldsymbol{\theta}}_i^{(s)}$

    $\mathbf{C}_{i+1} \leftarrow \mathbf{P}_{i+1}\mathbf{C}\mathbf{P}_{i+1}^\top \in \mathbb{R}^{(d-i)\times(d-i)}$

  $\hat{\mathbf{C}} \leftarrow \hat{\mathbf{C}} + s\sum_{i=1}^d \hat{\lambda}_i^{(s)} \hat{\boldsymbol{\theta}}_i^{(s)} \hat{\boldsymbol{\theta}}_i^{(s)^\top}$

Output: $\hat{\mathbf{C}}$

Chapter 7

Learning-augmented scientific computing

Our final study of learning-augmented algorithms extends the field in yet another direction: scientific computing. In particular, we study approximate linear system solving, a bottleneck subroutine in many scientific computations. For example, simulating a partial differential equation (PDE) often involves solving sequences of high-dimensional systems to very high precision [Thomas, 1999]. A vast array of solvers and preconditioners have thus been developed, many of which have tunable parameters that significantly affect runtime [Greenbaum, 1997, Hackbusch, 2016]. There is a long literature analyzing these algorithms, and indeed for some problems we have a strong understanding of the optimal parameters for a given matrix. However, computing them can be more costly than solving the original system, leading to an assortment of heuristics for setting good parameters [Ehrlich, 1981, Golub and Ye, 1999].

We provide an alternative to such heuristics by taking advantage of the fact that we often sequentially solve *many* linear systems. In addition to numerical simulation, this occurs in graphics computations such as mean-curvature flow [Kazhdan et al., 2012], nonlinear system solvers [Marquardt, 1963], and beyond. A natural approach is to treat these instances as data to be passed to a machine learning (ML) algorithm; in particular the framework of online learning that we have been extensively using provides a language to reason about such sequential learning problems. For example, if we otherwise would solve a sequence of linear systems $(\mathbf{A}_1, \mathbf{b}_1), \ldots, (\mathbf{A}_T, \mathbf{b}_T)$ using a given solver with a fixed parameter, can we use ML to do as well as the best choice of that parameter, i.e. can we *minimize regret*? Or, if the matrices are all diagonal shifts of single matrix $\mathbf{A}$, can we learn the functional relationship between the shift c_t and the optimal solver parameter for $\mathbf{A}_t = \mathbf{A} + c_t\mathbf{I}_n$, i.e. can we predict using *context*?

We investigate these questions for the Successive Over-Relaxation (SOR) solver, a generalization of Gauss-Seidel whose relaxation parameter $\omega \in (0, 2)$ dramatically affects the number of iterations (c.f. Figures 7.1 and 7.3, noting the log-scales). SOR and its symmetric variant are well-studied and often used as preconditioners for Krylov methods such as conjugate gradient (CG), as bases for semi-iterative schemes, and as multigrid smoothers. We sequentially set the parameter ω_t for SOR to use when solving each linear system $(\mathbf{A}_t, \mathbf{b}_t)$. Unlike past theoretical studies of related methods [Gupta and Roughgarden, 2017, Bartlett et al., 2022, Balcan et al., 2022], we aim to provide *end-to-end* guarantees—covering the full pipeline from data-intake to

[0]The work presented in this chapter first appeared in Khodak et al. [2024].

efficient learning to execution—while minimizing dependence on the dimension (n can be 10^5 or higher) and precision ($1/\varepsilon$ can be 10^8 or higher). We emphasize that we do *not* seek to immediately improve the empirical state of the art, and also that existing research on saving computation when solving sequences of linear systems (recycling Krylov subspaces, reusing preconditioners, etc.) is complementary to our own, i.e. it can be used in addition to the ideas presented here.

7.1 Contributions

We study two distinct theoretical settings, corresponding to views on the problem from two different approaches to data-driven algorithms. In the first we have a deterministic sequence of instances and study the spectral radius of the iteration matrix, the main quantity of interest in classical analysis of SOR [Young, 1971]. We show how to convert its asymptotic guarantee into a surrogate loss that upper bounds the number of iterations via a quality measure of the chosen parameter, in the style of *algorithms with predictions* [Mitzenmacher and Vassilvitskii, 2021]. The bound holds under a *near-asymptotic* condition implying that convergence occurs near the asymptotic regime, i.e. when the spectral radius of the iteration matrix governs the convergence. We verify the assumption and show that one can learn the surrogate losses using only bandit feedback from the original costs; notably, despite being non-Lipschitz, we take advantage of the losses' unimodal structure to match the optimal $\tilde{\mathcal{O}}(T^{2/3})$ regret for Lipschitz bandits [Kleinberg, 2004]. Our bound also depends only logarithmically on the precision and not at all on the dimension. Furthermore, we extend to the diagonally shifted setting described before, showing that an efficient, albeit pessimistic, contextual bandit (CB) method has $\tilde{\mathcal{O}}(T^{3/4})$ regret w.r.t. the instance-optimal policy that always picks the best ω_t. Finally, we show a similar analysis of learning a relaxation parameter for the more popular (symmetric SOR-preconditioned) CG method.

Our second setting is *semi-stochastic*, with target vectors $\mathbf{b}_t$ drawn i.i.d. from a (radially truncated) Gaussian. This is a reasonable simplification, as convergence usually depends more strongly on $\mathbf{A}_t$, on which we make no extra assumptions. We show that the expected cost of running a symmetric variant of SOR (SSOR) is $\mathcal{O}(\sqrt{n})\operatorname{polylog}(\frac{n}{\varepsilon})$-Lipschitz w.r.t. ω, so we can (a) compete with the optimal number of iterations—rather than with the best upper bound—and (b) analyze more practical, regression-based CB algorithms [Foster and Rakhlin, 2020, Simchi-Levi and Xu, 2021]. We then show $\tilde{\mathcal{O}}(\sqrt[3]{T^2\sqrt{n}})$ regret when comparing to the single best ω and $\tilde{\mathcal{O}}(T^{9/11}\sqrt{n})$ regret w.r.t. the instance-optimal policy in the diagonally shifted setting using a novel, Chebyshev regression-based CB algorithm. While the results do depend on the dimension n, the dependence is much weaker than that of past work on data-driven tuning of a related regression problem [Balcan et al., 2022].

Remark 7.1.1. Likely the most popular algorithms for linear systems are Krylov subspace methods such as CG. While an eventual goal is to understand how to tune (many) parameters of (preconditioned) CG and other algorithms, SOR is a well-studied method and serves as a good starting point. In fact, we show that our near-asymptotic analysis extends directly, and in the semi-stochastic setting there is a natural path to (e.g.) SSOR-preconditioned CG, as it can be viewed as computing polynomials of iteration matrices where SSOR just takes powers. Lastly, apart from its use as a preconditioner and smoother, SOR is still sometimes preferred for direct use too [Fried and Metzler, 1978, Van Vleck and Dwyer, 1985, King et al., 1987, Woźnicki, 1993, 2001].

By studying a scientific computing problem through the lens of data-driven algorithms and online learning, we also make the following contributions to the latter two fields:

1. Ours is the first head-to-head comparison of two leading theoretical approaches to data-driven algorithms applied to the same problem. While the algorithms with predictions approach in Section 7.3 takes better advantage of the scientific computing literature to obtain (arguably) more interpretable and dimension-independent bounds, data-driven algorithm design [Balcan, 2021] competes directly with the quantity of interest in Section 7.4 and enables guarantees for modern CB algorithms.
2. For algorithms with predictions, our near-asymptotic approach may be extendable to other iterative solvers, as we demonstrate with CG. We also show that such performance bounds on a (partially-observable) cost are learnable even when the bounds themselves are too expensive to compute.
3. In data-driven algorithm design, we take the novel theoretical approach of proving continuity of *the expectation of* a discrete cost, rather than showing dispersion of its discontinuities [Balcan et al., 2018b] or bounding predicate complexity [Bartlett et al., 2022].
4. We introduce the idea of using CB to set *instance-adaptive* algorithmic parameters; while we showed (linear) instance-adaptivity via convexity in Chapters 5 and 6, we now go further by taking advantage of multi-instance structure to asymptotically do as well as the *instance-optimal* policy.
5. We show that standard discretization-based bandit algorithms are optimal for sequences of adversarially chosen *semi-Lipschitz* losses that generalize regular Lipschitz functions (c.f. Appendix 7.A).
6. We introduce a CB method combining SquareCB [Foster and Rakhlin, 2020] with Chebyshev polynomial regression to get sublinear regret on Lipschitz losses (c.f. Appendix 7.B).

Lastly, we show how our proposed methods can lead to practical speedups by applying them to the numerical simulation of a 2D heat equation and reducing the runtime by 2-3x. This demonstrates the potential of data-driven scientific computing to significantly impact practical codes.

7.2 Related work

Iterative (discrete) optimization has been studied in learning-augmented algorithms [Dinitz et al., 2021, Chen et al., 2022, Sakaue and Oki, 2022], and our construction of an upper bound under asymptotic convergence is inspired by our ARUBA framework, although unlike before we do not assume access to the bound directly because it depends on hard-to-compute spectral properties. Algorithms with predictions often involve initializing a computation with a prediction of its outcome, e.g. a vector near the solution $\mathbf{A}^{-1}\mathbf{b}$; we do not consider this because the runtime of SOR and other solvers depends fairly weakly on the distance to the initialization.

Scientific computing algorithms have been more extensively studied in data-driven algorithm design, starting with the study by Gupta and Roughgarden [2017] of the sample complexity of learning the step-size of gradient descent, which can also be used to solve linear systems. While their sample complexity guarantee is logarithmic in the precision $1/\varepsilon$, directly applying their

Lipschitz-like analysis in a bandit setting yields regret with a polynomial dependence; note that a typical setting of ε is 10^{-8}. Mathematically, their analysis relies crucially on the iteration reducing error at every step, which is well-known *not* to be the case for SOR (e.g. Trefethen and Embree [2005, Figure 25.6]). Data-driven numerical linear algebra was studied most explicitly by Bartlett et al. [2022], who provided sample complexity framework applicable to many algorithms; their focus is on the offline setting where an algorithm is learned from a batch of samples. While they do not consider linear systems directly, in Appendix 7.4.3 we do compare to the guarantee their framework implies for SOR; we obtain similar sample complexity with an efficient learning procedure, at the cost of a strong distributional assumption on the target vector. Note that generalization guarantees have been shown for convex quadratic programming—which subsumes linear systems—by Sambharya et al. [2023]; they focus on learning-to-initialize, which we do not consider because for high precisions the initialization quality usually does not have a strong impact on cost. Note that all of the above work also does not provide end-to-end guarantees, only e.g. sample complexity bounds.

Online learning guarantees were shown for the related problem of tuning regularized regression by Balcan et al. [2022], albeit in the easier full information setting and with the target of reducing error rather than computation. Their approach relies on the dispersion technique [Balcan et al., 2018b], which often involves showing that discontinuities in the cost are defined by bounded-degree polynomials [Balcan et al., 2020a]. While possibly applicable in our setting, we suspect using it would lead to unacceptably high dependence on the dimension and precision, as the power of the polynomials defining our decision boundaries is $\mathcal{O}(n^{-\log \varepsilon})$. Lastly, we believe our work is notable within this field as a first example of using contextual bandits, and in doing so competing with the provably instance-optimal policy.

Gradient-based meta-learning can also be viewed as a related field, although theoretical studies have focused on learning-theoretic notions of cost such as regret or statistical risk. Furthermore, their guarantees are usually on the error after a fixed number of gradient steps rather than the number of iterations required to converge; targeting the former can be highly suboptimal in scientific computing applications [Arisaka and Li, 2023]. This latter work, which connects meta-learning and data-driven scientific computing, analyzes specific case studies for accelerating numerical solvers, whereas we focus on a general learning guarantee.

Empirically, there are many learned solvers [Luz et al., 2020, Taghibakhshi et al., 2021, Li et al., 2023] and even full simulation replacements [Karniadakis et al., 2021, Li et al., 2021c]; to our knowledge, theoretical studies of the latter have focused on expressivity [Marwah et al., 2021]. Amortizing the cost on future simulations [Amos, 2023], these approaches use offline computation to train models that integrate directly with solvers or avoid solving linear systems altogether. In contrast, the methods we propose are online and lightweight, both computationally and in terms of implementation; unlike many deep learning approaches, the additional computation scales slowly with dimension and needs only black-box access to existing solvers. As a result, our methods can be viewed as reasonable baselines, and we discuss an indirect comparison with the CG-preconditioner-learning approach of Li et al. [2023] in Appendix 7.F. Finally, note that improving the performance of linear solvers across a sequence of related instances has seen a lot of study in the scientific computing literature [Parks et al., 2006, Tebbens and Tůma, 2007, Elbouyahyaoui et al., 2021]. To our knowledge, this work does not give explicit guarantees on the number of iterations, and so a direct theoretical comparison is challenging.

Algorithm 16: Successive over-relaxation (SOR) with relative convergence condition.

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$, parameter $\omega \in (0, 2)$, initial vector $\mathbf{x} \in \mathbb{R}^n$, tolerance $\varepsilon > 0$

$\mathbf{D} + \mathbf{L} + \mathbf{L}^\top \leftarrow \mathbf{A}$ `// D diagonal, L strictly lower triangular`

$\mathbf{W}_\omega \leftarrow \mathbf{D}/\omega + \mathbf{L}$ `// compute the third normal form`

$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$ `// compute initial residual`

for $k = 0, \ldots$ **do**

  if $\|\mathbf{r}_k\|_2 \leqslant \varepsilon \|\mathbf{r}_0\|_2$ **then**

    return k `// return iteration count (for use in learning)`

  $\mathbf{x} = \mathbf{x} + \mathbf{W}_\omega^{-1} \mathbf{r}_k$ `// solve triangular system and update vector`

  $\mathbf{r}_{k+1} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$ `// compute the next residual`

7.3 Asymptotic analysis of learning the relaxation parameter

We start this section by going over the problem setup and the SOR solver. Then we study the asymptotic analysis of the method to derive a good performance upper bound to target as a surrogate loss for the true cost function. Finally, we prove and analyze online learning guarantees.

7.3.1 Setup

At each step $t = 1, \ldots, T$ of (say) a numerical simulation we get a linear system instance, defined by a matrix-vector pair $(\mathbf{A}_t, \mathbf{b}_t) \in \mathbb{R}^{n \times n} \times \mathbb{R}^n$, and are asked for a vector $\mathbf{x} \in \mathbb{R}^n$ such that the norm of its *residual* or *defect* $\mathbf{r} = \mathbf{b}_t - \mathbf{A}_t \mathbf{x}$ is small. For now we define "small" in a relative sense, specifically $\|\mathbf{A}_t \mathbf{x} - \mathbf{b}_t\|_2 \leqslant \varepsilon \|\mathbf{b}_t\|_2$ for some *tolerance* $\varepsilon \in (0, 1)$; note that when using an iterative method initialized at $\mathbf{x} = \mathbf{0}_n$ this corresponds to reducing the residual by a factor $1/\varepsilon$, which we call the *precision*. In applications it can be quite high, and so we will show results whose dependence on it is at worst logarithmic. To make the analysis tractable, we make two assumptions (for now) about the matrices $\mathbf{A}$: they are symmetric positive-definite and consistently-ordered (c.f. Hackbusch [2016, Definition 4.23]). We emphasize that, while not necessary for convergence, both are standard in the analysis of SOR [Young, 1971]; see Hackbusch [2016, Criterion 4.24] for multiple settings where they holds.

To find a suitable $\mathbf{x}$ for each instance in the sequence we apply Algorithm 16 (SOR), which works by multiplying the current residual $\mathbf{r}$ by the inverse of a matrix $\mathbf{W}_\omega$—derived from the diagonal $\mathbf{D}$ and lower-triangular component $\mathbf{L}$ of $\mathbf{A}$—and then adding the result to the current iterate $\mathbf{x}$. Note that multiplication by $\mathbf{W}_\omega^{-1}$ is efficient because $\mathbf{W}_\omega$ is triangular. We will measure the cost of this algorithm by the number of iterations it takes to reach convergence, which we denote by $\texttt{SOR}(\mathbf{A}, \mathbf{b}, \omega)$, or $\texttt{SOR}_t(\omega)$ for short when it is run on the instance $(\mathbf{A}_t, \mathbf{b}_t)$. For simplicity, we will assume that the algorithm is always initialized at $\mathbf{x} = \mathbf{0}_n$, and so the first residual is just $\mathbf{b}$.

Having specified the computational setting, we now turn to the learning objective, which is to sequentially set the parameters $\omega_1, \ldots, \omega_T$ so as to minimize the total number of iterations:

$$\sum_{t=1}^{T} \texttt{SOR}_t(\omega_t) = \sum_{t=1}^{T} \texttt{SOR}(\mathbf{A}_t, \mathbf{b}_t, \omega_t) \tag{7.1}$$

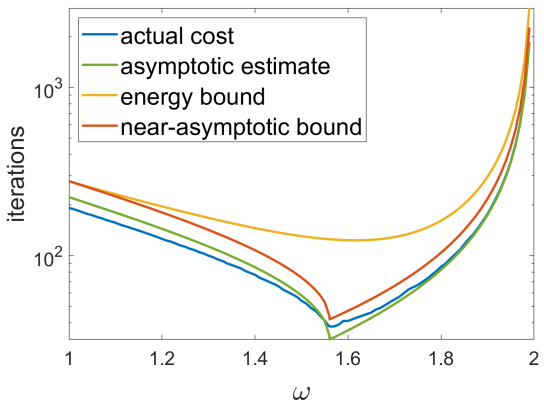

Figure 7.1: Comparison of different cost estimates for solving a linear system where the matrix is a discrete Laplacian of a 100×100 square domain.

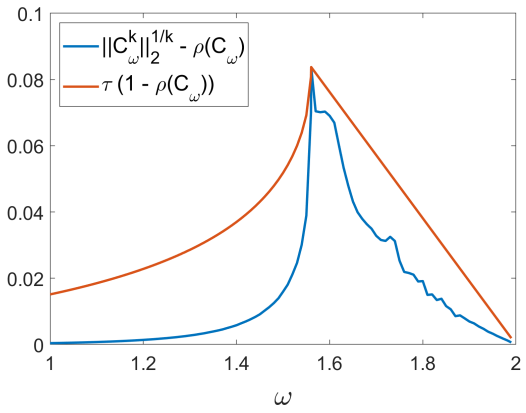

Figure 7.2: Asymptocity as measured by the difference between the spectral norm at iteration k and the spectral radius, together with its upper bound $\tau(1 - \rho(\mathbf{C}_\omega))$.

To set ω_t at some time $t > 1$, we allow the learning algorithm access to the costs $\texttt{SOR}_s(\omega_s)$ incurred at the previous steps $s = 1, \ldots, t-1$; in the literature on online learning this is referred to as the *bandit* or *partial feedback* setting, to distinguish from the (easier, but unreasonable for us) *full information* case where we have access to the cost function $\texttt{SOR}_s$ at every ω in its domain.

Selecting the optimal ω_t using no information about $\mathbf{A}_t$ is impossible, so we must use a *comparator* to obtain an achievable measure of performance. In online learning this is done by comparing the total cost incurred (7.1) to the counterfactual cost had we used a *single*, best-in-hindsight ω at every timestep t. We take the minimum over some domain $\Omega \subset (0, 2)$, as SOR diverges outside it. While in some settings we will compete with every $\omega \in (0, 2)$, we will often algorithmically use $[1, \omega_{\max}]$ for some $\omega_{\max} < 2$. The upper limit ensures a bound on the number of iterations—required by bandit algorithms—and the lower limit excludes $\omega < 1$, which is rarely used because theoretical convergence of vanilla SOR is worse there for realistic problems, e.g. those satisfying our assumptions.

This comparison-based approach for measuring performance is standard in online learning and effectively assumes a good $\omega \in \Omega$ that does well-enough on all problems; in Figure 7.3 (left) we show that this is sometimes the case. However, the right plot in the same figure shows we might do better by using additional knowledge about the instance; in online learning this is termed a *context* and there has been extensive development of contextual bandit algorithms that do as well as the best fixed policy mapping contexts to predictions. We will study an example of this in the *diagonally shifted* setting, in which $\mathbf{A}_t = \mathbf{A} + c_t \mathbf{I}_n$ for scalars $c_t \in \mathbb{R}$; while mathematically simple, this structure arises in natural settings, e.g. solving the heat equation with temporally variable diffusivity, and is well-motivated by other applications [Frommer and Glässner, 1998, Bellavia et al., 2011, Baumann and van Gijzen, 2015, Anzt et al., 2016, Wang et al., 2019]. Furthermore, the same learning algorithms can also be extended to make use of other context information, e.g. rough spectral estimates.

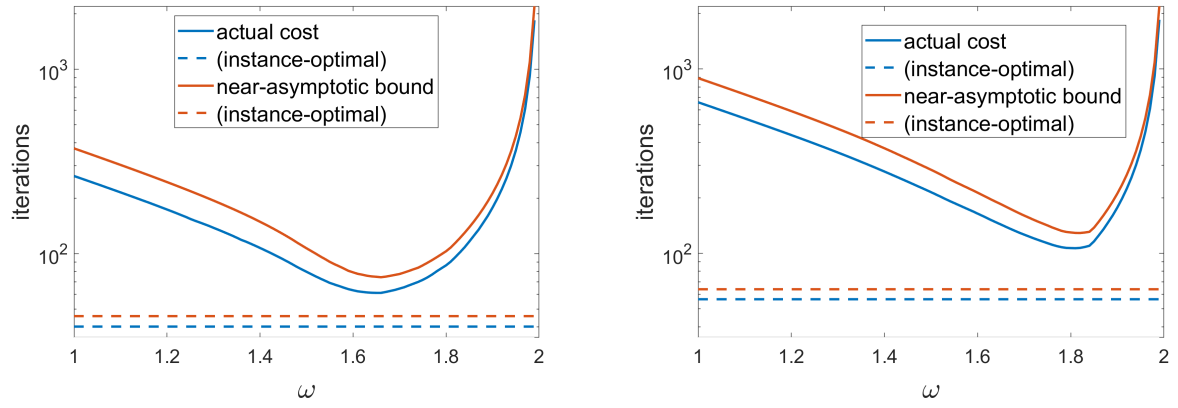

Figure 7.3: Mean performance of different parameters across forty instances of form $\mathbf{A} + \frac{12c-3}{20}\mathbf{I}_n$, where on the left plot $c \sim \text{Beta}(2, 6)$ and on the right $c \sim \text{Beta}(1/2, 3/2)$, the latter being relatively higher-variance. In both cases the dashed line indicates instance-optimal performance, the matrix $\mathbf{A}$ is a discrete Laplacian of a 100×100 square domain, and the targets $\mathbf{b}$ are truncated Gaussians.

7.3.2 Establishing a surrogate upper bound

Our first goal is to solve T linear systems almost as fast as if we had used the best fixed $\omega \in \Omega$ without knowing it in advance. In online learning, this corresponds to minimizing *regret*, which for cost functions $\ell_t : \Omega \mapsto \mathbb{R}$ is defined as

$$\text{Regret}(\{\ell_t\}_{t=1}^T) = \sum_{t=1}^T \ell_t(\omega_t) - \min_{\omega\in\Omega} \sum_{t=1}^T \ell_t(\omega) \tag{7.2}$$

In particular, since we can upper-bound the objective (7.1) by $\text{Regret}(\{\texttt{SOR}_t\}_{t=1}^T)$ plus the optimal cost $\min_{\omega\in\Omega} \sum_{t=1}^T \texttt{SOR}_t(\omega)$, if we show that regret is *sublinear* in T then the leading-order term in the upper bound corresponds to the cost incurred by the optimal fixed ω.

Many algorithms attaining sublinear regret under different conditions on the losses ℓ_t have been developed [Cesa-Bianchi and Lugosi, 2006, Bubeck and Cesa-Bianchi, 2012]. However, few handle losses with discontinuities—i.e. most algorithmic costs—and those that do (necessarily) need additional conditions on their locations [Balcan et al., 2018b, 2020a]. At the same time, numerical analysis often deals more directly with continuous asymptotic surrogates for cost, such as convergence rates. Thus we can try to apply ARUBA by finding *upper bounds* U_t on $\texttt{SOR}_t$ that are both (a) learnable and (b) reasonably tight in practice. We can then aim for overall performance nearly as good as the optimal $\omega \in \Omega$ as measured by these upper bounds:

$$\sum_{t=1}^T \texttt{SOR}_t(\omega_t) \leqslant \sum_{t=1}^T U_t(\omega_t) = \text{Regret}(\{U_t\}_{t=1}^T) + \min_{\omega\in\Omega} \sum_{t=1}^T U_t(\omega) = o(T) + \min_{\omega\in\Omega} \sum_{t=1}^T U_t(\omega) \tag{7.3}$$

A natural approach to get a bound U_t is via the *defect reduction matrix* $\mathbf{C}_\omega = \mathbf{I}_n - \mathbf{A}(\mathbf{D}/\omega + \mathbf{L})^{-1}$, so named because the residual at iteration k is $\mathbf{C}_\omega^k \mathbf{b}$ and $\mathbf{b}$ is the first residual. Under

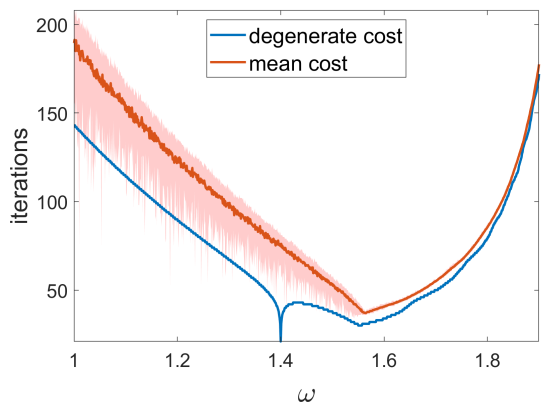

Figure 7.4: Solver cost for $\mathbf{b}$ drawn from a truncated Gaussian v.s. $\mathbf{b}$ a small eigenvector of $\mathbf{C}_{1.4}$.

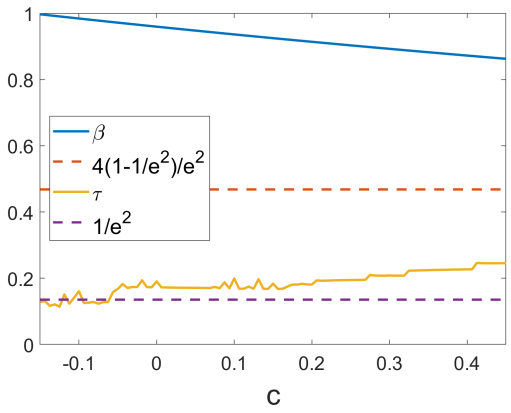

Figure 7.5: Values of τ and β for $\mathbf{A} + c\mathbf{I}_n$ for different c, where the matrix $\mathbf{A}$ is a discrete Laplacian of a 100×100 square domain.

our assumptions on $\mathbf{A}$, Young [1971] shows that the spectral radius $\rho(\mathbf{C}_\omega)$ of $\mathbf{C}_\omega$ is a (nontrivial to compute) piecewise function of ω with a unique minimum in $[1, 2)$. Since we have error $\|\mathbf{C}_\omega^k \mathbf{b}\|_2 / \|\mathbf{b}\|_2 \leqslant \|\mathbf{C}_\omega^k\|_\infty$ at iteration k, $\rho(\mathbf{C}_\omega) = \lim_{k\to\infty} \sqrt[k]{\|\mathbf{C}_\omega^k\|_\infty}$ asymptotically bounds how much the error is reduced at each step. It is thus often called the *asymptotic convergence rate* and the number of iterations is said to be roughly bounded by $\frac{-\log \varepsilon}{-\log \rho(\mathbf{C}_\omega)}$ (e.g. Hackbusch [2016, Equation 2.31b]). However, while it is tempting to use this as our upper bound, in fact it is often not one at all since $\mathbf{C}_\omega$ is not normal and so SOR often goes through a transient phase where the residual norm first *increases* before decreasing [Trefethen and Embree, 2005, Figure 25.6].

Thus we must either take a different approach or make assumptions. Note that one *can* show an ω-dependent, finite-time convergence bound for SOR via the energy norm [Hackbusch, 2016, Corollary 3.45], but this can give rather loose upper bounds on the number of iterations (c.f. Figure 7.1). Instead, we make the following assumption, which roughly states that convergence always occurs *near* the asymptotic regime, where nearness is measured by a parameter $\tau \in (0, 1)$:

Assumption 7.3.1. There exists $\tau \in (0, 1)$ s.t. $\forall\, \omega \in \Omega$ the matrix $\mathbf{C}_\omega = \mathbf{I}_n - \mathbf{A}(\mathbf{D}/\omega + \mathbf{L})^{-1}$ satisfies $\|\mathbf{C}_\omega^k\|_\infty \leqslant (\rho(\mathbf{C}_\omega) + \tau(1 - \rho(\mathbf{C}_\omega)))^k$ at $k = \min_{\|\mathbf{C}_\omega^{i+1}\mathbf{b}\|_2 < \varepsilon \|\mathbf{b}\|_2} i$.

This effectively assumes an upper bound $\rho(\mathbf{C}_\omega) + \tau(1 - \rho(\mathbf{C}_\omega))$ on the empirically observed convergence rate, which gives us a measure of the quality of each parameter ω for the given instance $(\mathbf{A}, \mathbf{b})$. Note that the specific form of the surrogate convergence rate was chosen both because it is convenient mathematically—it is a convex combination of 1 and the asymptotic rate $\rho(\mathbf{C}_\omega)$—and because empirically we found the degree of "asymptocity" as measured by $\|\mathbf{C}_\omega^k\|_\infty^{1/k} - \rho(\mathbf{C}_\omega)$ for k right before convergence to vary reasonably similarly to a fraction of $1 - \rho(\mathbf{C}_\omega)$ (c.f. Figure 7.2). This makes intuitive sense, as the parameters ω for which convergence is fastest have the least time to reach the asymptotic regime. Finally, note that since $\lim_{k\to\infty} \|\mathbf{C}_\omega^k\|_\infty^{1/k} = \rho(\mathbf{C}_\omega)$, for every $\gamma > 0$ there always exists k' s.t. $\|\mathbf{C}_\omega^k\|_\infty \leqslant (\rho(\mathbf{C}_\omega) + \gamma)^k\ \forall\, k \geqslant k'$; therefore, since $1 - \rho(\mathbf{C}_\omega) > 0$, we view Assumption 7.3.1 not as a restriction on $\mathbf{C}_\omega$ (and thus on $\mathbf{A}$), but rather as an an assumption on ε and $\mathbf{b}$. Specif-

Algorithm 17: Online tuning of a linear system solver using `Tsallis-INF`. The probabilities can be computed using Newton's method (e.g. Zimmert and Seldin [2021, Algorithm 2]).

Input: linear system solver $\texttt{SOLVE} : \mathbb{R}^{n\times n} \times \mathbb{R}^n \times \Omega \mapsto \mathbb{Z}_{>0}$, instance sequence $\{(\mathbf{A}_t, \mathbf{b}_t)\}_{t=1}^T \subset \mathbb{R}^{n\times n} \times \mathbb{R}^n$, normalization $K > 0$, parameter grid $\mathbf{g} \in \Omega^d$, step-size $\eta > 0$

$\mathbf{k} \leftarrow \mathbf{0}_d$ `// initialize vector of cumulative costs`

for $t = 1, \ldots, T$ **do**

- $\mathbf{p} \leftarrow \arg\min_{\mathbf{p}\in\triangle_d} \langle \mathbf{k}, \mathbf{p}\rangle - \frac{4K}{\eta}\sum_{i=1}^d \sqrt{\mathbf{p}_{[i]}}$ `// compute probabilities`
- sample $i_t \in [d]$ w.p. $\mathbf{p}_{[i_t]}$ and set $\omega_t = \mathbf{g}_{[i_t]}$ `// sample action from grid`
- $\mathbf{k}_{[i_t]} \leftarrow \mathbf{k}_{[i_t]} + \frac{\texttt{SOLVE}(\mathbf{A}_t, \mathbf{b}_t, \omega_t) - 1}{\mathbf{p}_{[i_t]}}$ `// run solver and update cost`

ically, the former should be small enough that $\mathbf{C}_\omega^i$ reaches that asymptotic regime for some i before the criterion $\|\mathbf{C}_\omega^k \mathbf{b}\|_2 \leqslant \varepsilon \|\mathbf{b}\|_2$ is met; for similar reasons, the latter should not happen to be an eigenvector corresponding to a tiny eigenvalue of $\mathbf{C}_\omega$ (c.f. Figure 7.4 (left)).

Having established this surrogate of the spectral radius, we can use it to obtain a reasonably tight upper bound U on the cost (c.f. Figure 7.1). Crucially for learning, we can also establish the following properties via the functional form of $\rho(\mathbf{C}_\omega)$ derived by Young [1971]:

Lemma 7.3.1. Define $U(\omega) = 1 + \frac{-\log\varepsilon}{-\log(\rho(\mathbf{C}_\omega) + \tau(1-\rho(\mathbf{C}_\omega)))}$, $\alpha = \tau + (1-\tau)\max\{\beta^2, \omega_{\text{max}} - 1\}$, and $\omega^* = 1 + \beta^2/(1+\sqrt{1-\beta^2})^2$, where $\beta = \rho(\mathbf{I}_n - \mathbf{D}^{-1}\mathbf{A})$. Then the following holds:

1. U bounds the iteration cost and is itself bounded: $\texttt{SOR}(\mathbf{A}, \mathbf{b}, \omega) < U(\omega) \leqslant 1 + \frac{-\log\varepsilon}{-\log\alpha}$
2. U is decreasing towards ω^*, and it is $\frac{-(1-\tau)\log\varepsilon}{\alpha\log^2\alpha}$-Lipschitz on $\omega \geqslant \omega^*$ if either $\tau \geqslant \frac{1}{e^2}$ or $\beta^2 \geqslant \frac{4}{e^2}(1 - \frac{1}{e^2})$

Lemma 7.3.1 introduces a quantity $\alpha = \tau + (1-\tau)\max\{\beta^2, \omega_{\text{max}} - 1\}$ that appears in the upper bound $U(\omega)$ and its Lipschitz constant. It will in some sense measure the difficulty of learning: if α is close to 1 for many instances then learning will be harder. Crucially, all quantities in the result are spectral and do not depend on the dimensionality of the matrix. Furthermore, while in the next subsection we *will* require that τ and β satisfy the conditions for (partial) Lipschitzness in the second item above, this does *not* truly reduce the matrices our analysis is applicable to, as we can always re-define τ in Assumption 7.3.1 to be at least $1/e^2$. Our analysis does not strongly depend on this latter restriction; it is largely done for simplicity and because it does not exclude too many settings of interest. In-particular, we find for $\varepsilon \geqslant 10^{-8}$ that τ is typically indeed larger than $\frac{1}{e^2}$, and furthermore τ is likely quite high whenever β is small, as it suggests the matrix is near-diagonal and so ω near one will converge extremely quickly (c.f. Figure 7.5 (right)).

7.3.3 Performing as well as the best fixed ω

Having shown these properties of U, we now show that it is learnable via Tsallis-INF [Abernethy et al., 2015, Zimmert and Seldin, 2021], a bandit algorithm which at each instance t samples ω_t from a discrete probability distribution over a grid of d relaxation parameters, runs SOR with ω_t

on the linear system $(\mathbf{A}_t, \mathbf{b}_t)$, and uses the number of iterations required $\texttt{SOR}_t(\omega_t)$ as feedback to update the probability distribution over the grid. The scheme is described in full in Algorithm 17. Note that it is a relative of the simpler and more familiar Exp3 algorithm [Auer et al., 2002], but has a slightly better dependence on the grid size d. In Theorem 7.3.1, we bound the cost of using the parameters ω_t suggested by Tsallis-INF by the total cost of using the best fixed parameter $\omega \in \Omega$ at all iterations—as measured by the surrogate bounds U_t—plus a term that increases sublinearly in T and a term that decreases in the size of the grid.

Theorem 7.3.1. Define $\alpha_t = \tau_t + (1 - \tau_t)\max\{\beta_t^2, \omega_{\text{max}} - 1\}$, where $\beta_t = \rho(\mathbf{I}_n - \mathbf{D}_t^{-1}\mathbf{A}_t)$ and τ_t is the minimal τ satisfying Assumption 7.3.1 and the second part of Lemma 7.3.1. If we run Algorithm 17 using SOR initialized at $\mathbf{x} = \mathbf{0}_n$ as the solver, $\mathbf{g}_{[i]} = 1 + (\omega_{\text{max}} - 1)\frac{i}{d}$ as the parameter grid, normalization $K \geqslant \frac{-\log\varepsilon}{-\log\alpha_{\text{max}}}$ for $\alpha_{\text{max}} = \max_t \alpha_t$, and step-size $\eta = 1/\sqrt{T}$ then the expected number of iterations is bounded as

$$\mathbb{E}\sum_{t=1}^{T} \texttt{SOR}_t(\omega_t) \leqslant 2K\sqrt{2dT} + \sum_{t=1}^{T} \frac{-\log\varepsilon}{d\log^2\alpha_t} + \min_{\omega\in(0,\omega_{\text{max}}]} \sum_{t=1}^{T} U_t(\omega) \tag{7.4}$$

Using $\omega_{\text{max}} = 1 + \max_t \left(\frac{\beta_t}{1+\sqrt{1-\beta_t^2}}\right)^2$, $K = \frac{-\log\varepsilon}{-\log\alpha_{\text{max}}}$, and $d = \sqrt[3]{\frac{T}{2}\bar{\gamma}^2\log^2\alpha_{\text{max}}}$, for $\bar{\gamma} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{\log^2\alpha_t}$, yields

$$\begin{aligned}\mathbb{E}\sum_{t=1}^{T} \texttt{SOR}_t(\omega_t) &\leqslant 3\log\frac{1}{\varepsilon}\sqrt[3]{\frac{2\bar{\gamma}T^2}{\log^2\alpha_{\text{max}}}} + \min_{\omega\in(0,2)}\sum_{t=1}^{T} U_t(\omega) \\ &\leqslant 3\log\frac{1}{\varepsilon}\sqrt[3]{\frac{2T^2}{\log^4\alpha_{\text{max}}}} + \min_{\omega\in(0,2)}\sum_{t=1}^{T} U_t(\omega)\end{aligned} \tag{7.5}$$

Thus *asymptotically* (as $T \to \infty$) the *average* cost on each instance is that of the best fixed $\omega \in (0, 2)$, as measured by the surrogate loss functions $U_t(\omega)$. The result clearly shows that the difficulty of the learning problem can be measured by how close the values of α_t are to one. As a quantitative example, for the somewhat "easy" case of $\tau_t \leqslant 0.2$ and $\beta_t \leqslant 0.9$, the first term is $< T\log\frac{1}{\varepsilon}$—i.e. we take at most $\log\frac{1}{\varepsilon}$ excess iterations on average—after around 73K instances.

The proof of Theorem 7.3.1 (c.f. Section 7.D) takes advantage of the fact that the upper bounds U_t are always decreasing wherever they are not locally Lipschitz; thus for any $\omega \in (0, \omega_{\text{max}}]$ the next highest grid value in $\mathbf{g}$ will either be better or $\mathcal{O}(1/d)$ worse. This allows us to obtain the same $\mathcal{O}(T^{2/3})$ rate as the optimal Lipschitz-bandit regret [Kleinberg, 2004], despite U_t being only semi-Lipschitz. One important note is that setting ω_{max}, K, and d to obtain this rate involves knowing bounds on spectral properties of the instances. The optimal ω_{max} requires a bound on $\max_t \beta_t$ akin to that used by solvers like Chebyshev semi-iteration; assuming this and a reasonable sense of how many iterations are typically required is enough to estimate α_{max} and then set $d = \sqrt[3]{\frac{T/2}{\log^2\alpha_{\text{max}}}}$, yielding the right-hand bound in (7.5). Lastly, we note that Tsallis-INF adds quite little computational overhead: it has a per-instance update cost of $\mathcal{O}(d)$, which for $d = \mathcal{O}(\sqrt[3]{T})$ is likely to be negligible in practice.

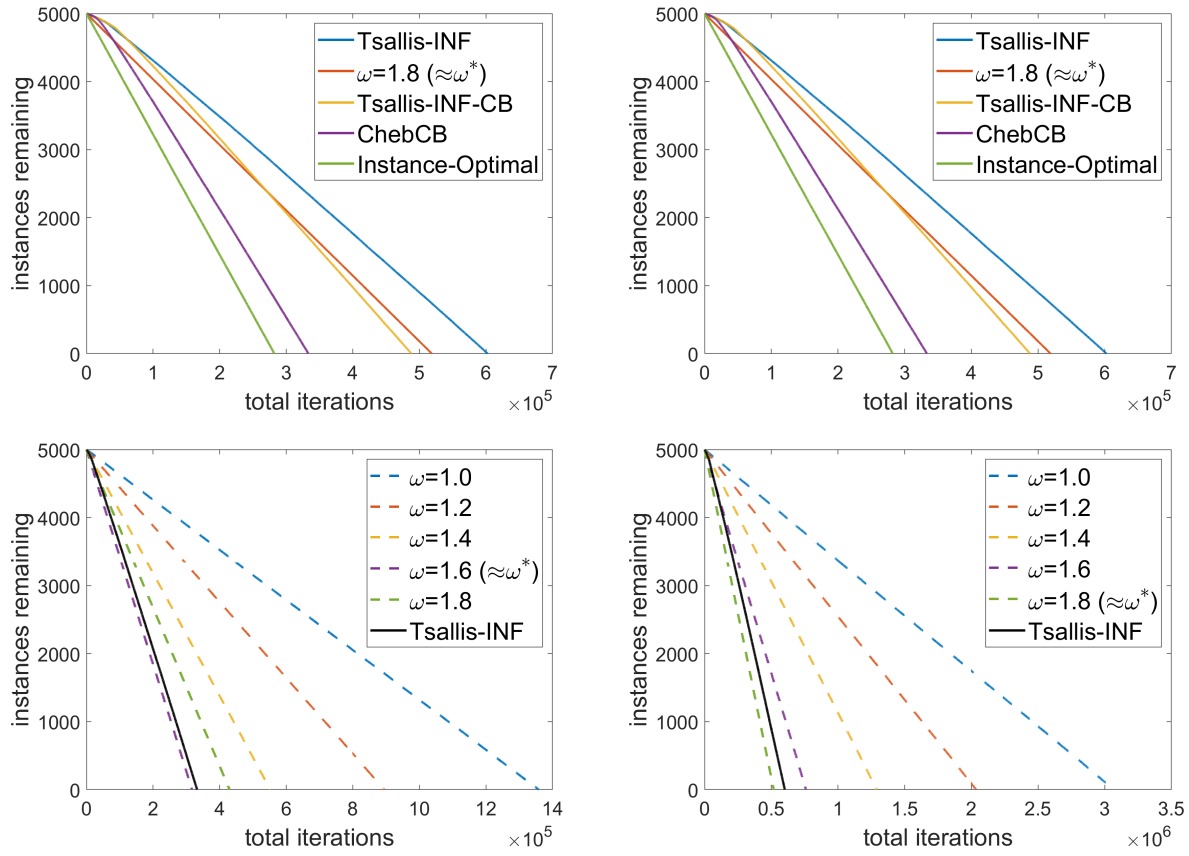

Figure 7.6: Average across forty trials of the time (in iterations) needed to solve 5K diagonally shifted systems with $\mathbf{A}_t = \mathbf{A} + \frac{12c-3}{20}\mathbf{I}_n$ for $c \sim \text{Beta}(\frac{1}{2}, \frac{3}{2})$ (left) and $c \sim \text{Beta}(2, 6)$ (right); as in Figure 7.3, $\mathbf{A}$ is a 100×100 Laplacian and the targets $\mathbf{b}$ are sampled from truncated Gaussians. The plots on the upper row compare different learning schemes while those on the bottom compare `Tsallis-INF` to different fixed choices of ω.

7.3.4 The diagonally shifted setting

The previous analysis is useful when a fixed ω is good for most instances $(\mathbf{A}_t, \mathbf{b}_t)$. A non-fixed comparator can be much stronger (c.f. the dashed lines in Figure 7.3), so in this section we study how to use additional, known structure in the form of diagonal shifts: $\mathbf{A}_t = \mathbf{A} + c_t\mathbf{I}_n$ for a fixed $\mathbf{A}$ and scalars c_t. It is easy to see that selecting an instance-dependent ω_t using the value of c_t is exactly the *contextual bandit* setting [Beygelzimer et al., 2011], in which the comparator is a fixed *policy* $f : \mathbb{R} \mapsto \Omega$ that maps c_t to a good parameter. In this setting we bound $\text{Regret}_f(\{\ell_t\}_{t=1}^T) = \sum_{t=1}^T \ell_t(\omega_t) - \sum_{t=1}^T \ell_t(f(c_t))$, so if f is the optimal mapping from c_t to ω then sublinear regret implies doing nearly optimally at every instance. In our case, the policy ω^* minimizing U_t is a well-defined function of $\mathbf{A}_t$ (c.f. Lemma 7.3.1) and thus of c_t [Young, 1971]; in fact, the policy is Lipschitz w.r.t. c_t (c.f. Lemma 7.D.1). This allows us to use a very simple algorithm—discretizing the space of offsets c_t into m intervals and running Tsallis-INF separately on each—to obtain $\mathcal{O}(T^{3/4})$ regret w.r.t. the instance-optimal policy ω^*:

Theorem 7.3.2 (c.f. Theorem 7.D.1)**.** Suppose all offsets c_t lie in $[c_{\min}, c_{\min} + C]$ for some $c_{\min} > -\lambda_{\min}(\mathbf{A})$, and define $L = \frac{1+\beta_{\max}}{\beta_{\max}\sqrt{1-\beta_{\max}^2}} \left(\frac{\lambda_{\min}(\mathbf{D})+c_{\min}+1}{\lambda_{\min}(\mathbf{D})+c_{\min}}\right)^2$ for $\beta_{\max}$ as in Theorem 7.3.1. Then there is a discretization of this interval s.t. running Algorithm 17 separately on each sequence of contexts in each bin with appropriate parameters results in expected cost

$$\mathbb{E}\sum_{t=1}^{T} \mathtt{SOR}_t(\omega_t) \leqslant \sqrt[4]{\frac{54C^3L^3T}{\log^2 \alpha_{\max}}} + \frac{4\log\frac{1}{\varepsilon}}{\log\frac{1}{\alpha_{\max}}}\sqrt[4]{24CLT^3} + \sum_{t=1}^{T} U_t(\omega^*(c_t)) \tag{7.6}$$

Observe that, in addition to α_t, the difficulty of this learning problem also depends on the maximum spectral radius $\beta_{\max}$ of the Jacobi matrices $\mathbf{I}_n - \mathbf{D}^{-1}(c_t)\mathbf{A}(c_t)$ via the Lipschitz constant L of ω^*.

7.3.5 Tuning preconditioned conjugate gradient

CG is perhaps the most-used solver for positive definite systems; while it can be run without tuning, in practice significant acceleration can be realized via a good preconditioner such as (symmetric) SOR. The effect of ω on CG performance can be somewhat distinct from that of regular SOR, requiring a separate analysis. We use the condition number analysis of Axelsson [1994, Theorem 7.17] to obtain an upper bound $U^{\mathtt{CG}}(\omega)$ on the number of iterations required $\mathtt{CG}(\mathbf{A}, \mathbf{b}, \omega)$ to solve a system. While the resulting bounds match the shape of the true performance less exactly than the SOR bounds (c.f. Figure 7.9), they still provide a somewhat reasonable surrogate. After showing that these functions are also semi-Lipschitz (c.f. Lemma 7.D.2), we can bound the cost of tuning CG using Tsallis-INF:

Theorem 7.3.3. Set $\mu_t = \rho(\mathbf{D}_t\mathbf{A}_t^{-1})$, $\mu_{\max} = \max_t \mu_t$, $\overline{\sqrt{\mu}} = \frac{1}{T}\sum_{t=1}^{T}\sqrt{\mu}_t$, and $\kappa_{\max} = \max_t \kappa(\mathbf{A}_t)$. If $\min_t \mu_t - 1$ is a positive constant then for Algorithm 17 using preconditioned CG as the solver there exists a parameter grid $\mathbf{g} \in [2\sqrt{2}+2, \omega_{\max}]^d$ and normalization $K > 0$ such that

$$\mathbb{E}\sum_{t=1}^{T} \mathtt{CG}_t(\omega_t) = \mathcal{O}\left(\sqrt[3]{\frac{\log^2\frac{\sqrt{\kappa_{\max}}}{\varepsilon}}{\log^2\frac{\sqrt{\mu_{\max}}-1}{\sqrt{\mu_{\max}}+1}}\overline{\sqrt{\mu}}T^2}\right) + \min_{\omega\in(0,2)}\sum_{t=1}^{T} U_t(\omega) \tag{7.7}$$

Observe that the rate in T remains the same as for SOR, but the difficulty of learning now scales mainly with the spectral radii of the matrices $\mathbf{D}_t\mathbf{A}_t^{-1}$.

7.4 A stochastic analysis of symmetric SOR

Assumption 7.3.1 in the previous section effectively encodes the idea that convergence will not be too quick for a typical target vector $\mathbf{b}$, e.g. it will not be a low-eigenvalue eigenvector of $\mathbf{C}_\omega$ for some otherwise suboptimal ω (e.g. Figure 7.4 (left)). Another way of staying in a "typical regime" is randomness, which is what we assume in this section. Specifically, we assume that $\mathbf{b}_t = m_t\mathbf{u}_t \;\forall\, t \in [T]$, where $\mathbf{u}_t \in \mathbb{R}^n$ is uniform on the unit sphere and m_t^2 is a χ^2 random variable with n degrees of freedom truncated to $[0, n]$. Since the standard n-dimensional Gaussian is exactly the case of untruncated m_t^2, $\mathbf{b}$ can be described as coming from a radially truncated

normal distribution. Note also that the exact choice of truncation was done for convenience; any finite bound $\geqslant n$ yields similar results.

We also make two other changes: (1) we study *symmetric* SOR (SSOR) and (2) we use an absolute convergence criterion, i.e. $\|\mathbf{r}_k\|_2 \leqslant \varepsilon$, not $\|\mathbf{r}_k\|_2 \leqslant \varepsilon\|\mathbf{r}_0\|_2$. Symmetric SOR (c.f. Algorithm 23) is very similar to the original, except the linear system being solved at every step is now symmetric: $\breve{\mathbf{W}}_\omega = \frac{\omega}{2-\omega}\mathbf{W}_\omega\mathbf{D}^{-1}\mathbf{W}_\omega^\top$. Note that the defect reduction matrix $\breve{\mathbf{C}}_\omega = \mathbf{I}_n - \mathbf{A}\breve{\mathbf{W}}_\omega^{-1}$ is still not normal, but it *is* (non-orthogonally) similar to a symmetric matrix, $\mathbf{A}^{-1/2}\breve{\mathbf{C}}_\omega\mathbf{A}^{1/2}$. SSOR is twice as expensive per-iteration, but often converges in fewer steps, and is commonly used as a base method because of its spectral properties (e.g. by the Chebyshev semi-iteration, c.f. Hackbusch [2016, Section 8.4.1]).

7.4.1 Regularity of the expected cost function

We can then show that the expected cost $\mathbb{E}_\mathbf{b}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega)$ is Lipschitz w.r.t. ω (c.f. Corollary 7.E.1). Our main idea is the observation that, whenever the error $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2$ falls below the tolerance ε, randomness should ensure that it does not fall so close to the threshold that the error $\|\breve{\mathbf{C}}_{\omega'}^k\mathbf{b}\|_2$ of a nearby ω' is not also below ε. Although clearly related to dispersion [Balcan et al., 2018b], here we study the behavior of a continuous function around a threshold, rather than the locations of the costs' discontinuities.

Our approach has two ingredients, the first being Lipschitzness of the error $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2$ at each iteration k w.r.t. ω, which ensures $\|\breve{\mathbf{C}}_{\omega'}^k\mathbf{b}\|_2 \in (\varepsilon, \varepsilon + \mathcal{O}(|\omega - \omega'|)]$ if $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2 \leqslant \varepsilon < \|\breve{\mathbf{C}}_{\omega'}^k\mathbf{b}\|_2$. The second ingredient is anti-concentration, specifically that the probability that $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2$ lands in $(\varepsilon, \varepsilon + \mathcal{O}(|\omega - \omega'|)]$ is $\mathcal{O}(|\omega - \omega'|)$. While intuitive, both steps are made difficult by powering: for high k the random variable $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2$ is highly concentrated because $\rho(\breve{\mathbf{C}}_\omega) \ll 1$; in fact its measure over the interval is $\mathcal{O}(|\omega - \omega'|/\rho(\breve{\mathbf{C}}_\omega)^k)$. To cancel this, the Lipschitz constant of $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2$ must scale with $\rho(\breve{\mathbf{C}}_\omega)^k$, which we can show because switching to SSOR makes $\breve{\mathbf{C}}_\omega^k$ is similar to a normal matrix. The other algorithmic modification we make—using absolute rather than relative tolerance—is so that $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2^2$ is (roughly) a sum of i.i.d. χ^2 random variables; note that the square of relative tolerance criterion $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2^2/\|\mathbf{b}\|_2^2$ does not admit such a result. At the same time, absolute tolerance does not imply an a.s. bound on the number of iterations if $\|\mathbf{b}\|_2$ is unbounded, which is why we truncate its distribution.

Lipschitzness follows because $|\mathbb{E}_\mathbf{b}\texttt{SSOR}(\omega) - \mathbb{E}_\mathbf{b}\texttt{SSOR}(\omega')|$ can be bounded using Jensen's inequality by the probability that ω and ω' have different costs $k \neq l$, which is at most the probability that $\|\breve{\mathbf{C}}_\omega^k\mathbf{b}\|_2$ or $\|\breve{\mathbf{C}}_{\omega'}^l\mathbf{b}\|_2$ land in an interval of length $\mathcal{O}(|\omega - \omega'|)$. Note that the Lipschitz bound includes an $\tilde{\mathcal{O}}(\sqrt{n})$ factor, which results from $\breve{\mathbf{C}}_\omega^k$ having stable rank $\ll n$ due to powering. Regularity of $\mathbb{E}_\mathbf{b}\texttt{SSOR}$ leads directly to regret guarantee for the same algorithm as before, Tsallis-INF:

Theorem 7.4.1. Define $\kappa_{\max} = \max_t \kappa(\mathbf{A}_t)$ to be the largest condition number and $\beta_{\min} = \min_t \rho(\mathbf{I}_n - \mathbf{D}_t^{-1}\mathbf{A}_t)$. Then there exists $K = \Omega(\log\frac{n}{\varepsilon})$ s.t. running Algorithm 17 with SSOR has regret

$$\mathbb{E}\sum_{t=1}^T \texttt{SSOR}_t(\omega_t) - \min_{\omega\in[1,\omega_{\max}]}\sum_{t=1}^T \texttt{SSOR}_t(\omega) \leqslant 2K\sqrt{2dT} + \frac{32K^4T}{\beta_{\min}^4 d}\sqrt{\frac{2n\kappa_{\max}}{\pi}} \tag{7.8}$$

Algorithm 18: ChebCB: SquareCB with a follow-the-leader oracle and polynomial regressor class.

Input: linear system solver $\texttt{SOLVE} : \mathbb{R}^{n\times n} \times \mathbb{R}^n \times \Omega \mapsto \mathbb{Z}_{>0}$, instance sequence $\{(\mathbf{A}_t, \mathbf{b}_t)\}_{t=1}^T \subset \mathbb{R}^{n\times n} \times \mathbb{R}^n$, context sequence $\{c_t\}_{t=1}^T \subset [c_{\min}, c_{\min} + C]$, learning rate $\eta > 0$, parameter grid $\mathbf{g} \in \Omega^d$, Chebyshev polynomial features $\mathbf{f} : [c_{\min}, c_{\min} + C] \mapsto \mathbb{R}^{m+1}$, normalizations $K, L, N > 0$

for $t = 1, \dots, T$ **do**

$\theta_i \leftarrow \underset{|\theta_{[0]}|\le\frac{1}{N}, |\theta_{[j]}|\le\frac{2CL}{KNj}}{\arg\min} \sum_{\substack{s=1 \\ i_s=i}}^{t-1} \left(\langle\theta, \mathbf{f}(c_s)\rangle - \frac{k_s}{KN}\right)^2 \; \forall\, i \in [d]$ `// update models`

$\mathbf{s}_{[i]} \leftarrow \langle\theta_i, \mathbf{f}(c_t)\rangle \; \forall\, i \in [d]$ `// compute model predictions`

$i^* \leftarrow \arg\min_{i\in[d]} \mathbf{s}_{[i]}$

$\mathbf{p}_{[i]} \leftarrow \frac{1}{d+\eta(\mathbf{s}_{[i]} - \mathbf{s}_{[i^*]})} \; \forall\, i \ne i^*$ `// compute probability of each action`

$\mathbf{p}_{[i^*]} \leftarrow 1 - \sum_{i\ne i^*} \mathbf{p}_{[i]}$

sample $i_t \in [d]$ w.p. $\mathbf{p}_{[i_t]}$ and set $\omega_t = \mathbf{g}_{[i_t]}$ `// sample action`

$k_t \leftarrow \texttt{SOLVE}(\mathbf{A}_t, \mathbf{b}_t, \omega_t) - 1$ `// run solver and update cost`

Setting $d = \Theta(K^2\sqrt[3]{nT})$ yields a regret bound of $\mathcal{O}(\log^2 \frac{n}{\varepsilon}\sqrt[3]{T^2\sqrt{n}})$. Note that, while this shows convergence to the true optimal parameter, the constants in the regret term are much worse, not just due to the dependence on n but also in the powers of the number of iterations. Thus this result can be viewed as a proof of the asymptotic ($T \to \infty$) correctness of Tsallis-INF for tuning SSOR.

7.4.2 Chebyshev regression for diagonal shifts

For the shifted setting, we can use the same approach to prove that $\mathbb{E}_{\mathbf{b}}\texttt{SSOR}(\mathbf{A} + c\mathbf{I}_n, \mathbf{b}, \omega)$ is Lipschitz w.r.t. the diagonal offset c (c.f. Corollary 7.E.2); for $n = O(1)$ this implies regret $\tilde{\mathcal{O}}(T^{3/4}\sqrt{n})$ for the same discretization-based algorithm as in Section 7.3.4. While optimal for Lipschitz functions, the method does not readily adapt to nice data, leading to various smoothed comparators [Krishnamurthy et al., 2019, Majzoubi et al., 2020, Zhu and Mineiro, 2022]; however, as we wish to compete with the true optimal policy, we stay in the original setting and instead highlight how this section's semi-stochastic analysis allows us to study a very different class of bandit algorithms.

In particular, since we are now working directly with the cost function rather than an upper bound, we are able to utilize a more practical regression-oracle algorithm, SquareCB [Foster and Rakhlin, 2020]. It assumes a class of regressors $h : [c_{\min}, c_{\min} + C] \times [d] \mapsto [0, 1]$ with at least one function that perfectly predicts the expected performance $\mathbb{E}_{\mathbf{b}}\texttt{SSOR}(\mathbf{A} + c\mathbf{I}_n, \mathbf{b}, \mathbf{g}_{[i]})$ of each action $\mathbf{g}_{[i]}$ given the context c; a small amount of model misspecification is allowed. If there exists an online algorithm that can obtain low regret w.r.t. this function class, then SquareCB can obtain low regret w.r.t. any policy.

To apply it we must specify a suitable class of regressors, bound its approximation error, and specify an algorithm attaining low regret over this class. Since m terms of the Chebyshev

series suffice to approximate a Lipschitz function with error $\tilde{\mathcal{O}}(1/m)$, we use Chebyshev polynomials in c with learned coefficients—i.e. models $\langle \theta, \mathbf{f}(c) \rangle = \sum_{j=0}^{m} \theta_{[j]} P_j(c)$, where P_j is the jth Chebyshev polynomial—as our regressors for each action. To keep predictions bounded, we add constraints $|\theta_{[j]}| = \mathcal{O}(1/j)$, which we can do without losing approximation power due to the decay of Chebyshev series coefficients. This allows us to show $\mathcal{O}(dm \log T)$ regret for Follow-The-Leader via Hazan et al. [2007, Theorem 5] and then apply Foster and Rakhlin [2020, Theorem 5] to obtain the following guarantee:

Theorem 7.4.2 (Corollary of Theorem 7.B.4)**.** Suppose $c_{\min} > -\lambda_{\min}(\mathbf{A})$. Then Algorithm 18 with appropriate parameters has regret w.r.t. any policy $f : [c_{\min}, c_{\min} + C] \mapsto \Omega$ of

$$\mathbb{E} \sum_{t=1}^{T} \mathtt{SSOR}_t(\omega_t) - \sum_{t=1}^{T} \mathtt{SSOR}_t(f(c_t)) \leqslant \tilde{\mathcal{O}} \left(d\sqrt{mnT} + \frac{T\sqrt{dn}}{m} + \frac{T\sqrt{n}}{d} \right) \tag{7.9}$$

Setting $d = \Theta(T^{2/11})$ and $m = \Theta(T^{3/11})$ yields $\tilde{\mathcal{O}}(T^{9/11}\sqrt{n})$ regret, so we asymptotically attain instance-optimal performance, albeit at a rather slow rate. The rate in n is also worse than e.g. our semi-stochastic result for comparing to a fixed ω (c.f. Theorem 7.4.1), although to obtain this the latter algorithm uses $d = \mathcal{O}(\sqrt[3]{n})$ grid points, making its overhead nontrivial. Experimentally, we compare ChebCB to the Section 7.3.4 algorithm that uses multiple runs of Tsallis-INF (among other methods), and find that, despite the former's worse guarantees, it seems able to converge to an instance-optimal policy much faster than the latter (c.f. Figure 7.7 (left)).

7.4.3 Additional theoretical implications and comparisons

Before going into details of our experimental results, we briefly discuss some additional sample complexity implications of our work and compare our results to two theoretical baselines.

Sample complexity implications and a comparison to the Goldberg-Jerrum framework

While not the focus of our work, we briefly note the generalization implications of our semi-stochastic analysis. Suppose for any $\alpha > 0$ we have $T = \tilde{\mathcal{O}}(\frac{1}{\alpha^2} \operatorname{polylog} \frac{n}{\delta})$ i.i.d. samples from a distribution $\mathcal{D}$ over matrices $\mathbf{A}_t$ satisfying the assumptions in Section 7.3.1 and truncated Gaussian targets $\mathbf{b}_t$. Then empirical risk minimization $\hat{\omega} = \arg\min_{\hat{\omega} \in \mathbf{g}} \sum_{t=1}^{T} \mathtt{SSOR}(\mathbf{A}_t, \mathbf{b}_t, \omega)$ over a uniform grid $\mathbf{g} \in [1, \omega_{\max}]^d$ of size $d = \tilde{\mathcal{O}}(\sqrt{nT})$ will be α-suboptimal w.p. $\geqslant 1 - \delta$:

Corollary 7.4.1. Let $\mathcal{D}$ be a distribution over matrix-vector pairs $(\mathbf{A}, \mathbf{b}) \in \mathbb{R}^{n \times n} \times \mathbb{R}^n$ where $\mathbf{A}$ satisfies the SOR conditions and for every $\mathbf{A}$ the conditional distribution of $\mathcal{D}$ given $\mathbf{A}$ over $\mathbb{R}^n$ is the truncated Gaussian. For every $T \geqslant 1$ consider the algorithm that draws T samples $(\mathbf{A}_t, \mathbf{b}_t) \sim \mathcal{D}$ and outputs $\hat{\omega} = \arg\min_{\hat{\omega} \in \mathbf{g}} \sum_{t=1}^{T} \mathtt{SSOR}_t(\omega)$, where $\mathbf{g}_{[i]} = 1 + (\omega_{\max} - 1)\frac{i - 1/2}{d}$ and $d = \frac{L\sqrt{T}}{K}$ for L as in Corollary 7.E.1. Then $T = \tilde{\mathcal{O}}\left(\frac{1}{\alpha^2} \operatorname{polylog} \frac{n}{\varepsilon\delta}\right)$ samples suffice to ensure $\mathbb{E}_{\mathcal{D}} \mathtt{SSOR}(\hat{\mathbf{A}}, \mathbf{b}, \hat{\omega}) \leqslant \min_{\omega \in [1, \omega_{\max}]} \mathtt{SSOR}(\mathbf{A}, \mathbf{b}, \omega) + \alpha$ holds w.p. $\geqslant 1 - \delta$.

This matches directly applying the GJ framework of Bartlett et al. [2022, Theorem 3.3] to our problem:

Corollary 7.4.2. In the same setting as Corollary 7.4.1 but generalizing the distribution to any one whose target vector support is $\sqrt{n}$-bounded, empirical risk minimization (running $\hat{\omega} = \arg\min_{\omega \in [1, \omega_{\max}]} \sum_{t=1}^{T} \texttt{SSOR}_t(\omega)$) has sample complexity $\tilde{\mathcal{O}}\left(\frac{1}{\alpha^2} \operatorname{polylog} \frac{n}{\varepsilon\delta}\right)$.

At the same, recent generalization guarantees for tuning regularization parameters of linear regression by Balcan et al. [2022, Theorem 3.2]—who applied dual function analysis [Balcan et al., 2021a]—have a quadratic dependence on the instance dimension. Unlike both results—which use uniform convergence—our bound also uses a (theoretically) efficient learning procedure, at the cost of a strong (but in our view reasonable) distributional assumption on the target vectors.

Approximating the spectral radius of the Jacobi iteration matrix

Because the asymptotically optimal ω is a function of the spectral radius $\beta = \rho(\mathbf{M}_1)$ of the Jacobi iteration matrix, a simple baseline is to approximate β using an eigenvalue solver and then run SOR with the corresponding approximately best ω. It is difficult to compare our results to this approach directly, since the baseline will always run extra matrix iterations while bandit algorithms will asymptotically run no more than the comparator. Furthermore, $\mathbf{M}_1$ is not a normal matrix, a class for which it turns out to be surprisingly difficult to find bounds on the number of iterations required to approximate its largest eigenvalue within some tolerance $\alpha > 0$.

A comparison can be made in the diagonal offset setting by modifying this baseline somewhat and making the assumption that $\mathbf{A}$ has a constant diagonal, so that $\mathbf{M}_1$ is symmetric and we can use randomized block-Krylov to obtain a $\hat{\beta}$ satisfying $|\hat{\beta}^2 - \beta^2| = \mathcal{O}(\varepsilon)$ in $\tilde{\mathcal{O}}(1/\sqrt{\varepsilon})$ iterations w.h.p. [Musco and Musco, 2015, Theorem 1]. To modify the baseline, we consider a *preprocessing* algorithm which discretizes $[c_{\min}, c_{\min} + C]$ into d grid points, runs k iterations of randomized block-Krylov on the Jacobi iteration matrix of each matrix $\mathbf{A} + c\mathbf{I}_n$ corresponding to offsets c in this grid, and then for each new offset c_t we set ω_t using the optimal parameter implied by the approximate spectral radius of the Jacobi iteration matrix of $\mathbf{A} + c\mathbf{I}_n$ corresponding to the closest c in the grid. This algorithm thus does $\tilde{\mathcal{O}}(dk)$ matrix-vector products of preprocessing, and since the upper bounds U_t are $\frac{1}{2}$-Hölder w.r.t. ω while the optimal policy is Lipschitz w.r.t. β^2 over an appropriate domain $[1, \omega_{\max}]$ it will w.h.p. use at most $\tilde{\mathcal{O}}(\sqrt{1/k^2 + 1/d})$ more iterations at each step $t \in [T]$ compared to the optimal policy. Thus w.h.p. the total regret compared to the optimal policy ω^* is

$$\sum_{t=1}^{T} \texttt{SOR}_t(\omega_t) = \tilde{\mathcal{O}}\left(dk + T/d + T/\sqrt{k}\right) + \sum_{t=1}^{T} U_t(\omega^*(c_t)) \tag{7.10}$$

Setting $d = \sqrt[4]{T}$ and $k = \sqrt{T}$ yields the rate $\tilde{\mathcal{O}}(T^{3/4})$, which can be compared directly to our $\tilde{\mathcal{O}}(T^{3/4})$ rate for the discretized Tsallis-INF algorithm in Theorem 7.3.2. The rate of approximating $\rho(\mathbf{M}_1)$ thus matches that of our simplest approach, although unlike the latter (and also unlike ChebCB) it does not guarantee performance as good as the optimal policy in the semi-stochastic setting, where ω^* might not be optimal. Intuitively, the randomized block-Krylov baseline will also suffer from spending computation on points $c \in [c_{\min}, c_{\min} + C]$ that it does not end up seeing.

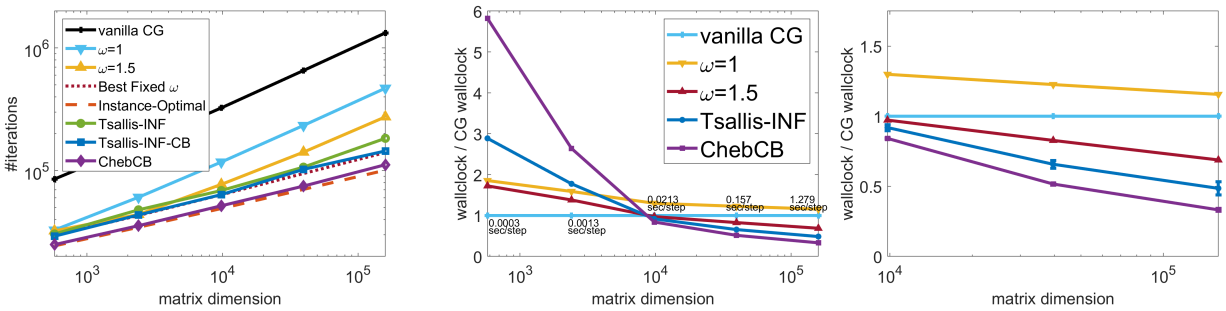

Figure 7.7: Cost of running the numerical simulation of the 2D heat equation in iterations (left) and normalized total wallclock time (center & right); the normalization is by the average number of seconds required when using vanilla CG to solve the linear systems (the latter's runtime is displayed as numbers in the middle). The right plot shows 95% confidence intervals across the three trials for Tsallis-INF and ChebCB at the three higher-dimensional evaluations.

7.5 Accelerating a 2D heat equation solver

In the experimental results so far we have looked at linear system instances $(\mathbf{A} + c\mathbf{I}_n, \mathbf{b})$ where both the target vectors $\mathbf{b}$ and the coefficients of the diagonal offsets determining the system matrix are synthetic, i.i.d. random variables. As demonstrated in Figure 7.6, the algorithms we propose are indeed able to accelerate solving sequences of such instances. However, the adversarial setting in which we proved our results suggests that we can go beyond i.i.d. coefficients c, and our argument for studying the semi-stochastic setting in Section 7.4 was that it preserved the properties of "typical" instances by not making any additional assumptions on $\mathbf{A}$. Furthermore, while we have mainly focused on tuning SOR alone, in Section 7.3.5 we also gave evidence of the applicability of our approach to tuning its use as a preconditioner for CG. Thus it is relevant to ask whether the proposed methods can accelerate the solving of *non-synthetic* sequences of linear system instances using solvers that are more commonly used in state-of-the-art scientific computing systems.

We answer in the affirmative by evaluating our algorithms on a sequence of linear system instances coming from a numerical simulation. In particular, we apply our methods to the task of simulating the 2D heat equation

$$\partial_t u(t, \mathbf{x}) = \kappa(t)\Delta_{\mathbf{x}} u(t, \mathbf{x}) + f(t, \mathbf{x}) \tag{7.11}$$

over the domain $\mathbf{x} \in [0, 1]^2$ and $t \in [0, 5]$. We use a five-point finite difference discretization with size denoted $n_{\mathbf{x}} = 1/\Delta_{\mathbf{x}}$, so that when an implicit time-stepping method such as Crank-Nicolson is applied with timestep Δ_t the numerical simulation requires sequentially solving a sequence of linear systems $(\mathbf{A}_t, \mathbf{b}_t)$ with $\mathbf{A}_t = \mathbf{I}_{(n_{\mathbf{x}}-1)^2} - \kappa((t + 1/2)\Delta_t)\mathbf{A}$ for a fixed matrix $\mathbf{A}$ (that depends on Δ_t and $\Delta_{\mathbf{x}}$) corresponding to the discrete Laplacian of the system [LeVeque, 2007, Equation 12.29]. Each $\mathbf{A}_t$ is positive definite, and moreover note that mathematically the setting is equivalent to an instantiation of the diagonal offset setting introduced in Section 7.3.4, since the linear system is equivalent to $c_t\mathbf{I}_{(n_{\mathbf{x}}-1)^2} - \mathbf{A} = c_t\mathbf{b}_t$ for $c_t = 1/\kappa((t + 1/2)\Delta_t)$. However, for simplicity we will simply pass $\kappa((t+1/2\Delta_t))$ as contexts to CB methods. In the experiments, we set $\kappa(t) = \max\{0.01\sin(2\pi t)), -10\sin(2\pi t)\}$, a functional form chosen to make the instance-

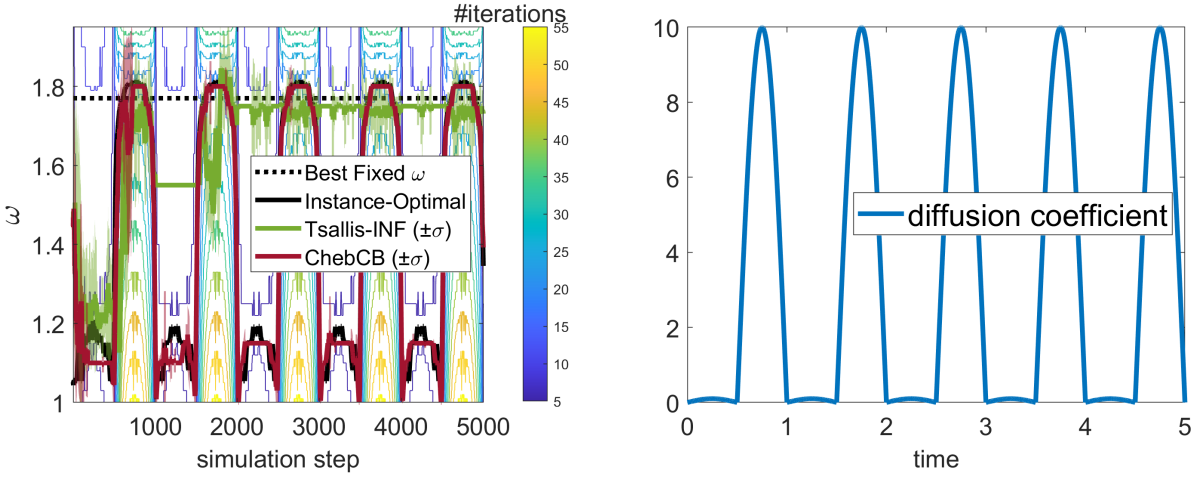

Figure 7.8: Parameters chosen at each timestep of a 2D heat simulation, overlaid on a contour plot of the cost of solving the system at step t with parameter ω (left); the periodic behavior of the instance-optimal action is driven by the time-varying diffusion coefficient $\kappa(t)$ (right).

optimal ω behave roughly periodically (c.f. Figure 7.8). For the complete problem specification, as well as algorithmic changes, see Appendix 7.F.

We set $\Delta_t = 10^{-3}$, thus making $T = 5000$, and evaluate our approach across five spatial discretizations: $n_{\mathbf{x}} = 25, 50, 100, 200, 400$; the resulting linear systems have size $n = (n_{\mathbf{x}} - 1)^2$. At each timestep, we solve each linear system using CG to relative precision $\varepsilon = 10^{-8}$. The baselines we consider are vanilla (unpreconditioned) CG and SSOR-CG with $\omega = 1$ or $\omega = 1.5$; as comparators we also evaluate performance when using the best fixed ω in hindsight at each round, and when using the *instance-optimal* ω at each round. Recall that we showed that Tsallis-INF has sublinear regret w.r.t. the surrogate cost of the best fixed ω of SSOR-CG (Theorem 7.3.3), and that ChebCB has sublinear regret w.r.t. the instance-optimal ω for SOR in the semi-stochastic setting of Section 7.4. Since both methods are randomized, we take the average of three runs.

Figure 7.7 (left) shows both methods substantially outperform all three baselines, except at $n_{\mathbf{x}} = 25$ and $n_{\mathbf{x}} = 50$, when $\omega = 1.5$ almost recovers the best fixed parameter; furthermore, ChebCB does better then the best fixed ω in hindsight in most cases. In Figure 7.7 (center & right) we also show that—at high-enough dimensions—this reduction in the number of iterations leads to an overall improvement in the *runtime* of the simulation. Lastly, we give additional details for the plot in Figure 7.8 (left), which shows the actions taken by the various algorithms for a simulation at $n_{\mathbf{x}} = 100$. For clarity all lines are smoothed using a moving average with a window of 25, and for Tsallis-INF and ChebCB we also shade $\pm$ one standard deviation computed over this window. The plot shows that Tsallis-INF converges to an action close to the best fixed ω in hindsight, and that ChebCB fairly quickly follows the instance-optimal path, with the standard deviation of both decreasing over time.

Our empirical results demonstrate the utility of learning as the problem scale increases. In particular, at lower dimensions the learning-based approaches have slower overall runtime because of overhead associated with learning; ChebCB in particular solves a small constrained linear regression at each step. However, this overhead does *not* scale with matrix dimension, and we expect data-driven approaches to have the greatest impact in higher dimensions. Figure 7.7 (cen-

ter & right) also show that vanilla (unpreconditioned) CG is faster than SSOR-preconditioned CG with $\omega = 1$ despite having more iterations because each iteration is more costly; thus learning can be needed to making preconditioned methods useful at all. Finally, we also observe that ChebCB is much better in practice than Tsallis-INF-CB, despite having a worse regret bound.

To get a comparative sense of how these improvements compare with previous approaches to data-driven scientific computing, we can consider the results in Li et al. [2023, Table 1], who learn a (deep-learning-based) preconditioner for CG to simulate the 2D heat equation. In the precision $\varepsilon = 10^{-8}$ case their solver takes 2.3 seconds, while Gauss-Seidel (i.e. SSOR with $\omega = 1$) takes 2.995 seconds, a roughly 1.3x improvement. In our most closely comparable setting, Tsallis-INF and ChebCB are roughly 2.4x and 3.5x faster than Gauss-Seidel, respectively (and have other advantages such as simplicity and being deployable in an online fashion without pretraining). We caveat this comparison by noting that Li et al. [2023] consider a statistical, not online, learning setup, and their matrix structure may be significantly different—it results from a finite element method rather than finite differences.[1]

7.6 Conclusion

We have shown that bandit algorithms provably learn to parameterize SOR, an iterative linear system solver, and do as well asymptotically as the best fixed ω in terms of either (a) a near-asymptotic measure of cost or (b) expected cost. We further show that a modern *contextual* bandit method attains near-instance-optimal performance. Both procedures require only the iteration count as feedback and have limited computational overhead settings, making them practical to deploy. Furthermore, the theoretical techniques—especially the use of contextual bandits for taking advantage of instance structure and Section 7.4.1's conversion of anti-concentrated Lipschitz criteria to Lipschitz expected costs—have the strong potential to be applicable to other domains of data-driven algorithm design. Finally, we also take initial steps towards accelerating practical numerical simulations using such approaches in Section 7.5, where we demonstrate 2-3x speedups over default approaches. Such performance improvements suggest expanding these ideas to more general settings and integrating them into more practical codebases is a useful direction for future work.

At the same time, only the near-asymptotic results yield reasonable bound on the instances needed to attain good performance, with the rest having large spectral and dimension-dependent factors; the latter is the most obvious area for improvement. Furthermore, the near-asymptotic upper bounds are somewhat loose for sub-optimal ω and for preconditioned CG, and as discussed in Section 7.3.4 do not seem amenable to regression-based CB. Beyond this, a natural direction is to attain semi-stochastic results for non-stationary solvers like preconditioned CG, or either type of result for the many other algorithms in scientific computing. Practically speaking, work on multiple parameters—e.g. the spectral bounds used for Chebyshev semi-iteration, or multiple relaxation parameters for Block-SOR—would likely be most useful. A final direction is to design online learning algorithms that exploit properties of the losses beyond Lipschitzness, or CB algorithms that take better advantage of such functions.

[1] A direct comparison is difficult without the authors' code, which as of this writing has not been made public.

Algorithm 19: General form of `Tsallis-INF`. The probabilities can be computed using Newton's method (e.g. Zimmert and Seldin [2021, Algorithm 2]).

Input: loss sequence $\{\ell_t : [a,b] \mapsto [0,K]\}_{t=1}^T$, action set $\mathbf{g} \in [a,b]^d$, step-sizes $\eta_1, \dots, \eta_T > 0$

$\mathbf{k} \leftarrow \mathbf{0}_d$ `// initialize vector of cumulative losses`

for $t = 1, \dots, T$ **do**

  $\mathbf{p} \leftarrow \arg\min_{\mathbf{p} \in \triangle_d} \langle \mathbf{k}, \mathbf{p} \rangle - \frac{4K}{\eta_t} \sum_{i=1}^d \sqrt{\mathbf{p}_{[i]}}$ `// compute probabilities`

  sample $i_t \in [d]$ with probability $\mathbf{p}_{[i_t]}$ `// sample index of an action`

  $\mathbf{k}_{[i_t]} \leftarrow \mathbf{k}_{[i_t]} + \ell_t(\mathbf{g}_{[i_t]}) / \mathbf{p}_{[i_t]}$ `// play action and update losses`

7.A Semi-Lipschitz bandits

We consider a sequence of adaptively chosen loss functions $\ell_1, \dots, \ell_T : [a,b] \mapsto [0,K]$ on an interval $[a,b] \subset \mathbb{R}$ and upper bounds $u_1, \dots, u_T : [a,b] \mapsto \mathbb{R}$ satisfying $u_t(x) \geqslant \ell_t(x)\ \forall\ t \in [T], x \in [a,b]$, where $[T]$ denotes the set of integers from 1 to T. Our analysis will focus on the Tsallis-INF algorithm of Abernethy et al. [2015], which we write in its general form in Algorithm 19, although the analysis extends easily to the better-known (but sub-optimal) Exp3 [Auer et al., 2002]. For Tsallis-INF, the following two facts follow directly from known results:

Theorem 7.A.1 (Corollary of Abernethy et al. [2015, Corollary 3.2])**.** If $\eta_t = 1/\sqrt{T}\ \forall\ t \in [T]$ then Algorithm 19 has regret $\mathbb{E}\sum_{t=1}^T \ell_t(\mathbf{g}_{[i_t]}) - \min_{i \in [d]} \sum_{t=1}^T \ell_t(\mathbf{g}_{[i]}) \leqslant 2K\sqrt{2dT}$.

Theorem 7.A.2 (Corollary of Zimmert and Seldin [2021, Theorem 1])**.** If $\eta_t = 2/\sqrt{t}\ \forall\ t \in [T]$ then Algorithm 19 has regret $\mathbb{E}\sum_{t=1}^T \ell_t(\mathbf{g}_{[i_t]}) - \min_{i \in [d]} \sum_{t=1}^T \ell_t(\mathbf{g}_{[i]}) \leqslant 4K\sqrt{dT} + 1$.

We now define a generalization of the Lipschitzness condition that trivially generalizes regular L-Lipschitz functions, as well as the notion of *one-sided Lipschitz* functions studied in the stochastic setting by Dütting et al. [2023].

Definition 7.A.1. Given a constant $L \geqslant 0$ and a point $z \in [a,b]$, we say a function $f : [a,b] \mapsto \mathbb{R}$ is (L,z)**-semi-Lipschitz** if $f(x) - f(y) \leqslant L|x-y|\ \forall\ x,y$ s.t. $|x-z| \leqslant |y-z|$.

We now show that Tsallis-INF with bandit access to ℓ_t on a discretization of $[a,b]$ attains $\mathcal{O}(T^{2/3})$ regret w.r.t. any fixed $x \in [a,b]$ evaluated by any comparator sequence of semi-Lipschitz upper bounds u_t. Note that guarantees for the standard comparator can be recovered by just setting $\ell_t = u_t\ \forall\ t \in [T]$, and that the rate is optimal by Kleinberg [2004, Theorem 4.2].

Theorem 7.A.3. If $u_t \geqslant \ell_t$ is (L_t, z)-semi-Lipschitz $\forall\ t \in [T]$ then Algorithm 19 using action space $\mathbf{g} \in [a,b]^d$ s.t. $\mathbf{g}_{[i]} = a + \frac{b-a}{d} i\ \forall\ i \in [d-1]$ and $\mathbf{g}_{[d]} = z$ has regret

$$\mathbb{E}\sum_{t=1}^T \ell_t(\mathbf{g}_{[i_t]}) - \min_{x \in [a,b]} \sum_{t=1}^T u_t(x) \leqslant 2K\sqrt{2dT} + \frac{b-a}{d}\sum_{t=1}^T L_t \tag{7.12}$$

Setting $d = \sqrt[3]{\frac{(b-a)^2 \bar{L}^2 T}{2K^2}}$ for $\bar{L} = \frac{1}{T}\sum_{t=1}^T L_t$ yields the bound $3\sqrt[3]{2(b-a)\bar{L}K^2T^2}$.

Algorithm 20: Contextual bandit algorithm using multiple runs of Tsallis-INF across a grid of contexts.

Input: loss sequence $\{\ell_t : [a,b] \mapsto [0,K]\}_{t=1}^T$, context sequence $\{c_t\}_{t=1}^T \subset [c, c+C]$, action set $\mathbf{g} \in [a,b]^d$, discretization $\mathbf{h} \in [c, c+C]^m$

for $j = 1, \dots, m$ **do**

  $\mathcal{A}_j = \texttt{Tsallis-INF}(\mathbf{g}, \{\frac{2}{\sqrt{t}}\}_{t=1}^T)$ `// start` m `runs of Algorithm 19`

for $t = 1, \dots, T$ **do**

  $j_t = \min \arg\min_{j \in [m]} |\mathbf{h}_{[j]} - c_t|$ `// pick element of` $\mathbf{h}$ `closest to` c_t

  $i_t \leftarrow \mathcal{A}_{j_t}$ `// get action from` j_t`th run of Algorithm 19`

  $\ell_t(\mathbf{g}_{[i_t]}) \to \mathcal{A}_{j_t}$ `// pass loss to` j_t`th run of Algorithm 19`

Proof. Let $\lceil \cdot \rfloor_{\mathbf{g}}$ denote rounding to the closest element of $\mathbf{g}$ in the direction of z. Then for $x \in [a,b]$ we have $|\lceil x \rfloor_{\mathbf{g}} - z| \leqslant |x - z|$ and $|\lceil x \rfloor_{\mathbf{g}} - x| \leqslant \frac{b-a}{d}$, so applying Theorem 7.A.1 and this fact yields

$$\begin{aligned}
\mathbb{E}\sum_{t=1}^T \ell_t(\mathbf{g}_{[i_t]}) &\leqslant 2K\sqrt{2dT} + \min_{i \in [d]} \sum_{t=1}^T \ell_t(\mathbf{g}_{[i]}) \leqslant 2K\sqrt{2dT} + \min_{i \in [d]} \sum_{t=1}^T u_t(\mathbf{g}_{[i]}) \\
&= 2K\sqrt{2dT} + \min_{x \in [a,b]} \sum_{t=1}^T u_t(\lceil x \rfloor_{\mathbf{g}}) \\
&\leqslant 2K\sqrt{2dT} + \frac{b-a}{d}\sum_{t=1}^T L_t + \min_{x \in [a,b]} \sum_{t=1}^T u_t(x)
\end{aligned} \tag{7.13}$$

□

For contextual bandits, we restrict to (L_t, b)-semi-Lipschitz functions and L_f-Lipschitz policies, obtaining $\mathcal{O}(T^{3/4})$ regret; this rate matches known upper and lower bounds for the case where losses are Lipschitz in both actions and contexts [Lu et al., 2010, Theorem 1], although this does not imply optimality of our result.

Theorem 7.A.4. If $u_t \geqslant \ell_t$ is (L_t, b)-semi-Lipschitz and $c_t \in [c, c+C]\ \forall\ t \in [T]$ then Algorithm 20 using action space $\mathbf{g}_{[i]} = a + \frac{b-a}{d} i$ and $\mathbf{h}_{[j]} = c + \frac{C}{m}(j - \frac{1}{2})$ as the grid of contexts has regret w.r.t. any L_f-Lipschitz policy $f : [c, c+C] \mapsto [a,b]$ of

$$\mathbb{E}\sum_{t=1}^T \ell_t(\mathbf{g}_{[i_t]}) - \sum_{t=1}^T u_t(\pi(c_t)) \leqslant m + 4K\sqrt{dmT} + \left(\frac{CL_f}{m} + \frac{b-a}{d}\right)\sum_{t=1}^T L_t \tag{7.14}$$

Setting $d = \sqrt[4]{\frac{(b-a)^3 \bar{L}^2 T}{4CL_f K^2}}$, $m = \sqrt[4]{\frac{C^3 L_f^3 \bar{L}^2 T}{4(b-a)K^2}}$ yields regret $4\sqrt[4]{4K^2\bar{L}^2(b-a)CL_f T^3} + \sqrt[4]{\frac{C^3 L_f^3 \bar{L}^2 T}{4(b-a)K^2}}$.

Proof. Define $\lceil \cdot \rfloor_{\mathbf{h}}$ to be the operation of rounding to the closest element of $\mathbf{h}$, breaking ties arbitrarily, and set $[T]_j = \{t \in [T] : \lceil c_t \rfloor_{\mathbf{h}} = \mathbf{h}_{[j]}\}$. Furthermore, define $\lceil x \rfloor_{\mathbf{g}}$ to be the smallest

element $\mathbf{g}_{[i]}$ in $\mathbf{g}$ s.t. $x + \frac{CL_f}{2m} \leqslant \mathbf{g}_{[i]}$ (or $\max_{i\in[d]} \mathbf{g}_{[i]}$ if such an element does not exist).

$$\begin{aligned}
\mathbb{E}\sum_{t=1}^{T} \ell_t(\mathbf{g}_{[i_t]}) &= \mathbb{E}\sum_{j=1}^{m}\sum_{t\in[T]_j} \ell_t(\mathbf{g}_{[i_t]}) - \min_{i\in[d]}\sum_{t\in[T]_j} \ell_t(\mathbf{g}_{[i]}) + \min_{i\in[d]}\sum_{t\in[T]_j} \ell_t(\mathbf{g}_{[i]}) \\
&\leqslant m + 4\sum_{j=1}^{m} K\sqrt{d|[T]_j|} + \min_{i\in[d]}\sum_{t\in[T]_j} \ell_t(\mathbf{g}_{[i]}) \\
&\leqslant m + 4K\sqrt{dmT} + \sum_{j=1}^{m}\min_{i\in[d]}\sum_{t\in[T]_j} u_t(\mathbf{g}_{[i]}) \\
&\leqslant m + 4K\sqrt{dmT} + \sum_{t=1}^{T} u_t(\lceil f(\lceil c_t \rfloor_{\mathbf{h}})\rceil_{\mathbf{g}})
\end{aligned} \tag{7.15}$$

where the first inequality follows by Theorem 7.A.2, the second applies Jensen's inequality to the left term and $u_t \geqslant \ell_t$ on the right, and the last uses optimality of each i for each j. Now since f is L_f-Lipschitz we have by definition of $\lceil\cdot\rfloor_{\mathbf{h}}$ that $|f(c_t) - f(\lceil c_t\rfloor_{\mathbf{h}})| \leqslant \frac{CL_f}{2m}$. This in turn implies that $f(c_t) \leqslant \lceil f(\lceil c_t\rfloor_{\mathbf{h}})\rceil_{\mathbf{g}} \leqslant f(c_t) + \frac{CL_f}{m} + \frac{b-a}{d}$ by definition of $\mathbf{g}$ and $\lceil\cdot\rfloor_{\mathbf{g}}$. Since u_t is (L_t, b)-semi-Lipschitz, the result follows. □

7.B Chebyshev regression for contextual bandits

7.B.1 Preliminaries

We first state a Lipschitz approximation result that is standard but difficult-to-find formally. For all $j \in \mathbb{Z}_{\geqslant 0}$ we will use $P_j(x) = \cos(j \arccos(x))$ to denote the jth Chebyshev polynomial of the first kind.

Theorem 7.B.1. Let $f : [\pm 1] \mapsto [\pm K]$ be a K-bounded, L-Lipschitz function. Then for each integer $m \geqslant 0$ there exists $\boldsymbol{\theta} \in \mathbb{R}^{m+1}$ satisfying the following properties:

1. $|\theta_{[0]}| \leqslant K$ and $|\boldsymbol{\theta}_{[j]}| \leqslant 2L/j \;\forall\; j \in [m]$
2. $\max_{x\in[\pm1]} \left|f(x) - \sum_{j=0}^{m} \boldsymbol{\theta}_{[j]} P_j(x)\right| \leqslant \frac{\pi + \frac{2}{\pi}\log(2m+1)}{m+1} L$

Proof. Define $\boldsymbol{\theta}_{[0]} = \frac{1}{\pi}\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} dx$ and for each $j \in [m]$ let $\boldsymbol{\theta}_{[j]} = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)P_j(x)}{\sqrt{1-x^2}} dx$ be the jth Chebyshev coefficient. Since $\int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}} = \pi$ we trivially have $|\boldsymbol{\theta}_{[0]}| \leqslant K$ and by Trefethen [2008, Theorem 4.2] we also have

$$|\boldsymbol{\theta}_{[j]}| \leqslant \frac{2}{\pi j}\int_{-1}^{1} \frac{|f'(x)|}{\sqrt{1-x^2}} dx \leqslant \frac{2L}{\pi j}\int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}} = 2L/j \tag{7.16}$$

for all $j \in [m]$. This shows the first property. For the second, by Trefethen [2008, Theorem 4.4]

Algorithm 21: SquareCB for contextual bandits using an online regression oracle.

Input: loss sequence $\{\ell_t : \mathbf{g} \mapsto [0,1]\}_{t=1}^T$, context sequence $\{c_t\}_{t=1}^T$, learning rate $\eta > 0$, online regression oracle $\mathcal{A}$

for $t = 1, \dots, T$ **do**

$\mathbf{s}_{[i]} \leftarrow \mathcal{A}(c_t, \mathbf{g}_{[i]}) \; \forall \; i \in [d]$ `// compute oracle prediction`

$i^* \leftarrow \arg\min_i \mathbf{s}_{[i]}$

$\mathbf{p}_{[i]} \leftarrow \frac{1}{d + \eta(\mathbf{s}_{[i]} - \mathbf{s}_{[i^*]})} \; \forall \; i \neq i^*$ `// compute action probabilities`

$\mathbf{p}_{[i^*]} \leftarrow 1 - \sum_{i \neq i^*} \mathbf{p}_{[i]}$

sample $i_t \in [d]$ with probability $\mathbf{p}_{[i_t]}$ `// sample index of a grid point`

$((c_t, \mathbf{g}_{[i_t]}), \ell_t(\mathbf{g}_{[i_t]})) \to \mathcal{A}$ `// pass context, action, loss to oracle`

we have that

$$\max_{x \in [-1,1]} \left| f(x) - \sum_{j=0}^{m} \boldsymbol{\theta}_{[j]} P_j(x) \right| \leqslant \left(2 + \frac{4 \log(2m+1)}{\pi^2} \right) \max_{x \in [\pm 1]} |f(x) - p_m^*(x)|$$
$$\leqslant \left(2 + \frac{4 \log(2m+1)}{\pi^2} \right) \frac{L\pi}{2(m+1)} = \frac{\pi + \frac{2}{\pi} \log(2m+1)}{m+1} L \tag{7.17}$$

where p_m^* is the (at most) m-degree algebraic polynomial that best approximates f on $[\pm 1]$ and the second inequality is Jackson's theorem [Cheney, 1982, page 147]. □

Corollary 7.B.1. Let $f : [a,b] \mapsto [\pm K]$ be a K-bounded, L-Lipschitz function on the interval $[a,b]$. Then for each integer $m \geqslant 0$ there exists $\boldsymbol{\theta} \in \mathbb{R}^{m+1}$ satisfying the following properties:

1. $|\boldsymbol{\theta}_{[0]}| \leqslant K$ and $|\boldsymbol{\theta}_{[j]}| \leqslant \frac{L(b-a)}{j}$
2. $\max_{x \in [a,b]} \left| f(x) - \sum_{j=0}^{m} \boldsymbol{\theta}_{[j]} P_j(\frac{2}{b-a}(x-a) - 1) \right| \leqslant \frac{\pi + \frac{2}{\pi} \log(2m+1)}{2(m+1)} L(b-a)$

Proof. Define $g(x) = f(\frac{b-a}{2}(x+1) + a)$, so that $g : [\pm 1] \mapsto [\pm K]$ is K-bounded and $L\frac{b-a}{2}$-Lipschitz. Applying Theorem 7.B.1 yields the result. □

We next state regret guarantees for the SquareCB algorithm of Foster and Rakhlin [2020] in the non-realizable setting:

Theorem 7.B.2 (Foster and Rakhlin [2020, Theorem 5])**.** Suppose for any sequence of actions $a_1, \dots, a_T$ an online regression oracle $\mathcal{A}$ playing regressors $h_1, \dots, h_T \in \mathcal{H}$ has regret upper bound

$$U(T) \geqslant \sum_{t=1}^{T} (\ell_t(c_t, a_t) - h_t(c_t, a_t))^2 - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} (\ell_t(c_t, a_t) - h(c_t, a_t))^2 \tag{7.18}$$

If all losses and regressors have range $[0,1]$ and $\exists \; h \in \mathcal{H}$ s.t. $\mathbb{E}\ell_t(a) = h(c_t, a) + \alpha_t(c_t, a)$ for $|\alpha_t(a)| \leqslant \alpha$ then Algorithm 21 with learning rate $\eta = 2\sqrt{dT/(U(T) + 2\alpha^2 T)}$ has expected

Algorithm 22: SquareCB for Lipschitz contextual bandits using Follow-the-Leader.

Input: loss sequence $\{\ell_t : [a,b] \mapsto [0,K]\}_{t=1}^T$, context sequence $\{c_t \in [c, c+C]\}_{t=1}^T$, learning rate $\eta > 0$, action set $\mathbf{g} \in [a,b]^d$, featurizer $\mathbf{f} : [c, c+C] \mapsto \mathbb{R}^m$, normalizations $L, N > 0$

for $t = 1, \dots, T$ **do**

  for $i = 1, \dots, d$ **do**

    $\boldsymbol{\theta}_i \leftarrow \underset{|\boldsymbol{\theta}_{[0]}| \leqslant \frac{1}{N}, |\boldsymbol{\theta}_{[j]}| \leqslant \frac{2CL}{KNj}}{\arg\min} \sum_{s \in [t-1]_i} \left(\langle \boldsymbol{\theta}, \mathbf{f}(c_s) \rangle - \frac{\ell_t(\mathbf{g}_{[i_s]})}{KN} \right)^2$ `// update models`

    $\mathbf{s}_{[i]} \leftarrow \langle \boldsymbol{\theta}_i, \mathbf{f}(c_t) \rangle$ `// compute model predictions`

  $i^* \leftarrow \arg\min_i \mathbf{s}_{[i]}$

  $\mathbf{p}_{[i]} \leftarrow \frac{1}{d + \eta(\mathbf{s}_{[i]} - \mathbf{s}_{[i^*]})} \; \forall \, i \neq i^*$ `// compute action probabilities`

  $\mathbf{p}_{[i^*]} = 1 - \sum_{i \neq i^*} \mathbf{p}_{[i]}$

  sample $i_t \in [d]$ with probability $\mathbf{p}_{[i_t]}$ and play action $\mathbf{g}_{[i_t]}$

regret w.r.t the the optimal policy $f : [a,b] \mapsto \mathbf{g}$ bounded as

$$\mathbb{E} \sum_{t=1}^T \ell_t(\mathbf{g}_{[i_t]}) - \sum_{t=1}^T \ell_t(h(c_t)) \leqslant 2\sqrt{dTU(T)} + 5\alpha T\sqrt{d} \tag{7.19}$$

SquareCB requires an online regression oracle to implement, for which we will use the Follow-the-Leader scheme. It has the following guarantee for squared losses:

Theorem 7.B.3 (Corollary of Hazan et al. [2007, Theorem 5])**.** Consider the follow-the-leader algorithm, which sequentially sees feature-target pairs $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_T, y_T) \in \mathcal{X} \times [0,1]$ for some subset $\mathcal{X} \subset [0,1]^n$ and at each step sets $\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta} \in \Theta} \sum_{t=1}^T (\langle \mathbf{x}_t, \boldsymbol{\theta} \rangle - y_t)^2$ for some subset $\Theta \subset \mathbb{R}^n$. This algorithm has regret

$$\sum_{t=1}^T (\langle \mathbf{x}_t, \boldsymbol{\theta}_t \rangle - y_t)^2 - \min_{\boldsymbol{\theta} \in \Theta} (\langle \mathbf{x}_t, \theta \rangle - y_t)^2 \leqslant 4B^2 n \left(1 + \log \frac{XDT}{2B} \right) \tag{7.20}$$

for D_Θ the diameter $\max_{\boldsymbol{\theta}, \boldsymbol{\theta}'} \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2$ of Θ, $X = \max_{t \in [T]} \|\mathbf{x}_t\|_2$, and $B = \max_{t \in [T], \boldsymbol{\theta} \in \Theta} |\langle \mathbf{x}_t, \boldsymbol{\theta} \rangle|$.

7.B.2 Regret of ChebCB

Theorem 7.B.4. Suppose $\mathbb{E}\ell_t(x)$ is an L_x-Lipschitz function of actions $x \in [a,b]$ and an L_c-Lipschitz function of contexts $c_t \in [c, c+C]$. Then Algorithm 22 run with learning rate $\eta = 2\sqrt{dT/(U(T) + 2\alpha^2 T)}$ for $U(T)$ and α as in Equations 7.22 and 7.23, respectively, action set $\mathbf{g}_{[i]} = a + (b-a)\frac{i - 1/2}{d}$, Chebyshev features $\mathbf{f}_{[j]}(c_t) = P_j(c_t)$, and normalizations $L = L_c$ and $N = 2 + \frac{4CL_c}{K}(1 + \log m)$ has regret w.r.t. any policy $f : [c, c+C] \mapsto [a,b]$ of

$$\mathbb{E} \sum_{t=1}^T \ell_t(\mathbf{g}_{[i_t]}) - \ell_t(f(c_t)) = \tilde{\mathcal{O}} \left(L_c d\sqrt{mT} + \frac{L_c T\sqrt{d}}{m} + \frac{L_x T}{d} \right) \tag{7.21}$$

Setting $d = \Theta(T^{2/11})$ and $m = \Theta(T^{3/11})$ yields a regret $\tilde{\mathcal{O}}(\max\{L_c, L_x\} T^{9/11})$.

Proof. Observe that the method is equivalent to running Algorithm 21 with the follow-the-leader oracle over an $d(m+1)$-dimensional space Θ with diameter $\sqrt{\frac{d}{N^2}\left(1+\frac{4C^2L_c^2}{K^2}\sum_{j=1}^m \frac{1}{j^2}\right)} \leqslant \frac{\sqrt{dK^2+2dC^2L_c^2\pi^2/3}}{KN}$, features bounded by $\sqrt{1+\sum_{j=1}^m P_j(c_t)} \leqslant \sqrt{m+1}$, and predictions bounded by $|\langle \mathbf{f}(c), \boldsymbol{\theta}\rangle| \leqslant \|\boldsymbol{\theta}\|_1 \|\mathbf{f}(c)\|_\infty \leqslant \frac{1}{N} + \frac{2CL_c}{KN}\sum_{j=1}^m \leqslant \frac{1}{2}$. Thus by Theorem 7.B.3 the oracle has regret at most

$$U(T) = d(m+1)\left(1+\log\frac{T\sqrt{d(m+1)(K^2+2C^2L_c^2\pi^2/3)}}{KN}\right) \tag{7.22}$$

Note that, to ensure the regressors and losses have range in $[0,1]$ we can define the former as $h(c, \mathbf{g}_{[i]}) = \langle \mathbf{f}(c), \boldsymbol{\theta}_i\rangle + \frac{1}{2}$ and the latter as $\frac{\ell_t}{KN} + \frac{1}{2}$ and Algorithm 22 remains the same. Furthermore, the error of the regression approximation is then

$$\alpha = \frac{\pi + \frac{2}{\pi}\log(2m+1)}{2KN(m+1)} CL_c \tag{7.23}$$

We conclude by applying Theorem 7.B.2, unnormalizing by multiplying the resulting regret by KN, and adding the approximation error $\frac{L_x(b-a)}{2d}$ due to the discretization of the action space. □

7.C SOR preliminaries

We will use the following notation:

- $\mathbf{M}_\omega = \mathbf{I}_n - (\mathbf{D}/\omega + \mathbf{L})^{-1}\mathbf{A}$ is the matrix of the first normal form [Hackbusch, 2016, 2.2.1]
- $\mathbf{W}_\omega = \mathbf{D}/\omega + \mathbf{L}$ is the matrix of the third normal form [Hackbusch, 2016, Section 2.2.3]
- $\mathbf{C}_\omega = \mathbf{I}_n - \mathbf{A}(\mathbf{D}/\omega + \mathbf{L})^{-1} = \mathbf{I}_n - \mathbf{A}\mathbf{W}_\omega^{-1} = \mathbf{A}\mathbf{M}_\omega\mathbf{A}^{-1}$ is the defect reduction matrix
- $\breve{\mathbf{M}}_\omega = \mathbf{I}_n - \frac{2-\omega}{\omega}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1}\mathbf{A}$ is the matrix of the first normal form in the case of SSOR [Hackbusch, 2016, 2.2.1]
- $\breve{\mathbf{W}}_\omega = \frac{\omega}{2-\omega}(\mathbf{D}/\omega + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D}/\omega + \mathbf{L}^\top)$ is the matrix of the third normal form in the case of SSOR [Hackbusch, 2016, Section 2.2.3]
- $\breve{\mathbf{C}}_\omega = \mathbf{I}_n - \frac{2-\omega}{\omega}\mathbf{A}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1} = \mathbf{I}_n - \mathbf{A}\breve{\mathbf{W}}_\omega^{-1} = \mathbf{A}\breve{\mathbf{M}}_\omega\mathbf{A}^{-1}$ is the defect reduction matrix in the case of SSOR

We further derive bounds on the number of iterations for SOR and SSOR using the following energy norm estimate:

Theorem 7.C.1 (Corollary of Hackbusch [2016, Theorem 3.44 & Corollary 3.45])**.** If $\mathbf{A} \succ 0$ and $\omega \in (0,2)$ then $\|\mathbf{M}_\omega\|_{\mathbf{A}}^2 \leqslant 1 - \frac{\frac{2-\omega}{\omega}\gamma}{\left(\frac{2-\omega}{2\omega}\right)^2 + \frac{\gamma}{\omega} + \rho(\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{L}^\top) - \frac{1}{4}}$, where $\gamma = 1 - \rho(\mathbf{D}^{-1}(\mathbf{L}+\mathbf{L}^\top))$.

Corollary 7.C.1. Let K_ω be the maximum number of iterations that SOR needs to reach error $\varepsilon > 0$. Then for any $\omega \in (0,2)$ we have $K_\omega \leqslant 1 + \frac{-\log\frac{\varepsilon}{2\sqrt{\kappa(A)}}}{-\log\nu_\omega(\mathbf{A})}$, where $\nu_\omega(\mathbf{A})$ is the square root of the upper bound in Theorem 7.C.1.

Proof. By Hackbusch [2016, Equations 2.22c & B.28b] we have at each iteration k of Algorithm 16 that

$$\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{r}_0\|_2} \leqslant \frac{2}{\|\mathbf{r}_0\|_2}\|\mathbf{A}^{\frac{1}{2}}\|_\infty\|\mathbf{M}_\omega^k\|_{\mathbf{A}}\|\mathbf{A}^{-\frac{1}{2}}\mathbf{r}_0\|_2 \leqslant 2\sqrt{\kappa(\mathbf{A})}\|\mathbf{M}_\omega\|_{\mathbf{A}}^k \tag{7.24}$$

Setting the r.h.s. equal to ε and solving for k yields the result. □

Corollary 7.C.2. Let $\breve{K}_\omega$ be the maximum number of iterations that SSOR (Algorithm 23) needs to reach (absolute) error $\varepsilon > 0$. Then for any $\omega \in (0,2)$ we have $\breve{K}_\omega \leqslant 1 + \frac{-\log \frac{\varepsilon}{2\|\mathbf{b}\|_2\sqrt{\kappa(A)}}}{-2\log \nu_\omega(\mathbf{A})}$, where $\nu_\omega(\mathbf{A})$ is the square root of the upper bound in Theorem 7.C.1.

Proof. By Hackbusch [2016, Equations 2.22c & B.28b] we have at each iteration k of Algorithm 16 that

$$\|\mathbf{r}_k\|_2 \leqslant 2\|\mathbf{A}^{\frac{1}{2}}\|_\infty\|\breve{\mathbf{M}}_\omega^k\|_{\mathbf{A}}\|\mathbf{A}^{-\frac{1}{2}}\mathbf{r}_0\|_2 \leqslant 2\|\mathbf{b}\|_2\sqrt{\kappa(\mathbf{A})}\|\mathbf{M}_\omega\|_{\mathbf{A}}^{2k} \tag{7.25}$$

Setting the r.h.s. equal to ε and solving for k yields the result. □

7.D Near-asymptotic proofs

7.D.1 Proof of Lemma 7.3.1

Proof. For the first claim, suppose $l = \min_{\|\mathbf{C}_\omega^k\mathbf{b}\|_2 < \varepsilon\|\mathbf{b}\|_2} k > U(\omega)$. Then

$$\varepsilon \leqslant \frac{\|\mathbf{C}_\omega^{l-1}\mathbf{b}\|_2}{\|\mathbf{b}\|_2} \leqslant \frac{\|\mathbf{C}_\omega^{l-1}\|_\infty\|\mathbf{b}\|_2}{\|\mathbf{b}\|_2} \leqslant (\rho(\mathbf{C}_\omega) + \tau(1-\rho(\mathbf{C}_\omega)))^{l-1} < \varepsilon \tag{7.26}$$

so by contradiction we must have $\min_{\|\mathbf{C}_\omega^k\mathbf{b}\|_2 < \varepsilon\|\mathbf{b}\|_2} k \leqslant U(\omega)$. Now by Hackbusch [2016, Theorem 4.27] and similarity of $\mathbf{C}_\omega$ and $\mathbf{M}_\omega$ we have that $\rho(\mathbf{C}_\omega) = \frac{1}{4}\left(\omega\beta + \sqrt{\omega^2\beta^2 - 4(\omega-1)}\right)^2$ for $\omega < \omega^* = 1 + \left(\frac{\beta}{1+\sqrt{1-\beta^2}}\right)^2$ and $\omega - 1$ otherwise. Therefore on $\omega < \omega^*$ we have $\rho(\mathbf{C}_\omega) \leqslant \rho(\mathbf{C}_1) = \beta^2$ and on $\omega \geqslant \omega^*$ we have $\rho(\mathbf{C}_\omega) \leqslant \omega_{\text{max}} - 1$. This concludes the second part of the first claim. The first part of the second claim follows because $\rho(\mathbf{C}_\omega)$ is decreasing on $\omega < \omega^*$. For the second part, we compute the derivative $|\partial_\omega U(\omega)| = \frac{(\tau-1)\log\varepsilon}{(\tau+(1-\tau)(\omega-1))\log^2(\tau+(1-\tau)(\omega-1))}$. Since $\tau + (1-\tau)(\omega-1) \geqslant \frac{1}{e^2}$ by assumption—either by nonnegativity of $\omega - 1$ if $\tau \geqslant \frac{1}{e^2}$ or because otherwise $\beta^2 \geqslant \frac{4}{e^2}(1-\frac{1}{e^2})$ implies $\tau + (1-\tau)(\omega-1) \geqslant (1-\frac{1}{e^2})\left(\frac{\beta}{1+\sqrt{1-\beta^2}}\right)^2 \geqslant \frac{1}{e^2}$—the derivative is increasing in ω and so is at most $\frac{-(1-\tau)\log\varepsilon}{\alpha\log^2\alpha}$. □

7.D.2 Proof of Theorem 7.3.1

Proof. The first bound follows from Theorem 7.A.3 by noting that Lemma 7.3.1 implies that the functions $U_t - 1$ are $\left(\frac{-(1-\tau_t)\log\varepsilon}{\alpha_t \log^2 \alpha_t}, \omega_{\max}\right)$-semi-Lipschitz over $[1, \omega_{\max}]$ and the functions $\mathtt{SOR}_t - 1 \leqslant U_t - 1$ are $\frac{-\log\varepsilon}{-\log\alpha_t}$-bounded. To extend the comparator domain to $(0, \omega_{\max}]$, note that Lemma 7.3.1.2 implies that all U_t are decreasing on $\omega \in (0, 1)$. To extend the comparator domain again in the second bound, note that the setting of $\omega_{\max}$ implies that the minimizer $1 + \beta_t^2/(1 + \sqrt{1-\beta_t^2})$ of each U_t is at most $\omega_{\max}$, and so all functions U_t are increasing on $\omega \in (\omega_{\max}, 2)$. □

7.D.3 Approximating the optimal policy

Lemma 7.D.1. Define $\mathbf{A}(c) = \mathbf{A} + c\mathbf{I}_n$ for all $c \in [c_{\min}, \infty)$, where $c_{\min} > -\lambda_{\min}(\mathbf{A})$. Then $\omega^*(c) = 1 + \left(\frac{\beta_c}{\sqrt{1-\beta_c^2}+1}\right)^2$ is $\frac{6\beta_{\max}(1+\beta_{\max})/\sqrt{1-\beta_{\max}^2}}{\left(\sqrt{1-\beta_{\max}^2}+1\right)^2}\left(\frac{\lambda_{\min}(\mathbf{D})+c_{\min}+1}{\lambda_{\min}(\mathbf{D})+c_{\min}}\right)^2$-Lipschitz, where $\beta_{\max} = \max_c \beta_c$ is the maximum over $\beta_c = \rho(\mathbf{I}_n - (\mathbf{D} + c\mathbf{I}_n)^{-1}(\mathbf{A} + c\mathbf{I}_n))$.

Proof. We first compute

$$
\begin{aligned}
\partial_c \beta_c &= \partial_c \rho(\mathbf{I}_n - (\mathbf{D} + c\mathbf{I}_n)^{-1}(\mathbf{A} + c\mathbf{I}_n)) \\
&= \partial_c \lambda_{\max}(\mathbf{I}_n - (\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}) \\
&= \mathbf{v}_1^\top \partial_c((\mathbf{I}_n - (\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + \mathbf{c}\mathbf{I}_n)^{-\frac{1}{2}}))\mathbf{v}_1 \\
&= -\mathbf{v}_1^\top(\partial_c((\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}})(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}} \\
&\qquad + (\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}\partial_c(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}} \\
&\qquad + (\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + c\mathbf{I}_n)\partial_c((\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}))\mathbf{v}_1 \\
&= \frac{1}{2}\mathbf{v}_1^\top((\mathbf{D} + c\mathbf{I}_n)^{-\frac{3}{2}}(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}} - 2c(\mathbf{D} + c\mathbf{I}_n)^{-1} \\
&\qquad + (\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{3}{2}})\mathbf{v}_1 \\
&= \frac{1}{2}\mathbf{v}_1^\top(\mathbf{I}_n + (\mathbf{D} + c\mathbf{I}_n)^{-1})(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + c\mathbf{I})(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{I}_n + (\mathbf{D} + c\mathbf{I}_n)^{-1})\mathbf{v}_1 \\
&\quad - \frac{1}{2}\mathbf{v}_1^\top(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}\mathbf{v}_1 \\
&\quad - \frac{1}{2}\mathbf{v}_1^\top(\mathbf{D} + c\mathbf{I}_n)^{-\frac{3}{2}}(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{3}{2}}\mathbf{v}_1 - c\mathbf{v}_1^\top(\mathbf{D} + c\mathbf{I}_n)^{-1}\mathbf{v}_1 \\
&= \frac{1}{2}\mathbf{v}_1^\top(\mathbf{I}_n + (\mathbf{D} + c\mathbf{I}_n)^{-1})(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + c\mathbf{I})(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{I}_n + (\mathbf{D} + c\mathbf{I}_n)^{-1})\mathbf{v}_1 \\
&\quad - \frac{1}{2}\mathbf{v}_1^\top(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A} + 3c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{1}{2}}\mathbf{v}_1 \\
&\quad - \frac{1}{2}\mathbf{v}_1^\top(\mathbf{D} + c\mathbf{I}_n)^{-\frac{3}{2}}(\mathbf{A} + c\mathbf{I}_n)(\mathbf{D} + c\mathbf{I}_n)^{-\frac{3}{2}}\mathbf{v}_1
\end{aligned}
\tag{7.27}
$$

The first component is positive and the matrix has eigenvalues bounded by $\left(1 + \frac{1}{\lambda_{\min}(\mathbf{D})+c}\right)^2 \frac{1+\beta_c}{2}$,

while the last term is negative and the matrix has If $c \geqslant 0$ the positive component has spectral radius at most $\frac{1+\beta_c}{2}\left(1 + \frac{1}{(\lambda_{\min}(\mathbf{D})+c)^2}\right)$. If the middle term is negative, subtracting $2\mathbf{A}$ from the middle matrix shows that its magnitude is bounded by $\frac{3}{2}(1+\beta_c)$. If the middle term is positive—which can only happen for negative c—its magnitude is bounded by $\frac{-3c/2}{\lambda_{\min}(\mathbf{D})+c} \leqslant \frac{3\lambda_{\min}(\mathbf{D})/2}{\lambda_{\min}(\mathbf{D})+c}$. Combining all terms yields a bound of $3\left(\frac{\lambda_{\min}(\mathbf{D})+c+1}{\lambda_{\min}(\mathbf{D})+c}\right)^2(1+\beta_c)$. We then have that

$$|\partial_c \omega^*(c)| = \frac{2\beta_c/\sqrt{1-\beta_c^2}}{\left(\sqrt{1-\beta_c^2}+1\right)^2}|\partial_c \beta_c| = \frac{6\beta_c(1+\beta_c)/\sqrt{1-\beta_c^2}}{\left(\sqrt{1-\beta_c^2}+1\right)^2}\left(\frac{\lambda_{\min}(\mathbf{D})+c+1}{\lambda_{\min}(\mathbf{D})+c}\right)^2 \tag{7.28}$$

The result follows because $\frac{2x(1+x)/\sqrt{1-x^2}}{\left(\sqrt{1-x^2}+1\right)^2}$ increases monotonically on $x \in [0,1)$ and the bound itself decreases monotonically in c □

Theorem 7.D.1. Suppose $c_t \in [c_{\min}, c_{\min}+C]\ \forall\ t \in [T]$, where $c_{\min} > -\lambda_{\min}(\mathbf{A})$, and define $\beta_{\max} = \max_t \beta_t$. Then if we run Algorithm 20 with losses $(\texttt{SOR}_t(\cdot)-1)/K$ normalized by $K \geqslant \frac{-\log\varepsilon}{-\log\alpha_{\max}}$ for $\alpha_{\max} = \max_t \alpha_t$, action set $\mathbf{g}_{[i]} = 1+(\omega_{\max}-1)\frac{i}{d}$ for $\omega_{\max} \geqslant 1+\left(\frac{\beta_{\max}}{1+\sqrt{1-\beta_{\max}^2}}\right)^2$, and context discretization $\mathbf{h}_{[j]} = c_{\min} + \frac{C}{m}\left(j-\frac{1}{2}\right)$, then the number of iterations will be bounded in expectation as

$$\mathbb{E}\sum_{t=1}^{T}\texttt{SOR}_t(\omega_t) \leqslant m + 4K\sqrt{dmT} + \left(\frac{CL^*/m}{\omega_{\max}-1}+\frac{1}{d}\right)\sum_{t=1}^{T}\frac{-\log\varepsilon}{\log^2\alpha_t} + \sum_{t=1}^{T}U_t(\omega^*(c_t)) \tag{7.29}$$

where L^* is the Lipschitz constant from Lemma 7.D.1. Setting $\omega_{\max} = 1+\left(\frac{\beta_{\max}^2}{1+\sqrt{1-\beta_{\max}^2}}\right)^2$, $K = \frac{-\log\varepsilon}{-\log\alpha_{\max}}$, $d = \sqrt[4]{\frac{\bar\gamma^2 T\log^2\alpha_{\max}}{24CL}}$, and $m = \sqrt[4]{54C^3L^3\bar\gamma^2 T\log^2\alpha_{\max}}$ where $\bar\gamma = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{\log^2\alpha_t}$ and $\tilde L = \left(\frac{\lambda_{\min}(\mathbf{D})+c_{\min}+1}{\lambda_{\min}(\mathbf{D})+c_{\min}}\right)^2\frac{1+\beta_{\max}}{\beta_{\max}\sqrt{1-\beta_{\max}^2}}$, yields

$$\begin{aligned} E\sum_{t=1}^{T}\texttt{SOR}_t(\omega_t) &\leqslant \sqrt[4]{54C^3L^3\bar\gamma^2T\log^2\alpha_{\max}} + 4\log\frac{1}{\varepsilon}\sqrt[4]{\frac{24CL\bar\gamma T^3}{\log^2\alpha_{\max}}} + \sum_{t=1}^{T}U_t(\omega^*(c_t)) \\ &\leqslant \sqrt[4]{\frac{54C^3L^3T}{\log^2\alpha_{\max}}} + \frac{4\log\frac{1}{\varepsilon}}{\log\frac{1}{\alpha_{\max}}}\sqrt[4]{24CLT^3} + \sum_{t=1}^{T}U_t(\omega^*(c_t)) \end{aligned} \tag{7.30}$$

Proof. The bound follows from Theorem 7.A.4 by noting that Lemma 7.3.1 implies that the functions $U_t - 1$ are $\left(\frac{-(1-\tau_t)\log\varepsilon}{\alpha_t\log^2\alpha_t}, \omega_{\max}\right)$-semi-Lipschitz over $[1,\omega_{\max}]$ and the functions $\texttt{SOR}_t - 1 \leqslant U_t - 1$ are $\frac{-\log\varepsilon}{-\log\alpha_t}$-bounded. Note that for the choice of $\omega_{\max}$ we the interval $[1,\omega_{\max}]$ contains the range of the optimal policy ω^*, and further by Lemma 7.D.1 it is L^*-Lipschitz over $[c_{\min}, c_{\min}+C]$. □

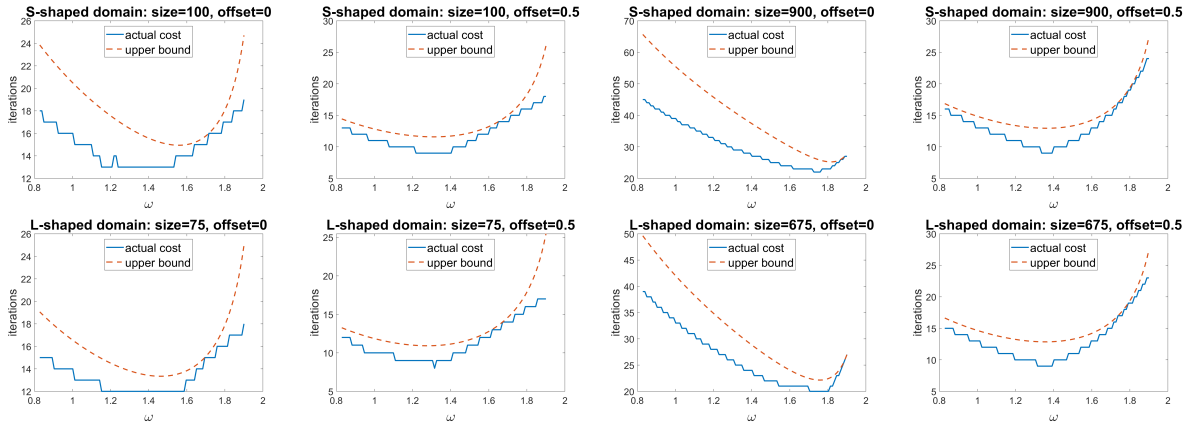

Figure 7.9: Comparison of actual cost of running SSOR-preconditioned CG and the upper bounds computed in Section 7.D.4 as functions of the tuning parameter $\omega \in [2\sqrt{2}-2, 1.9]$ on various domains.

7.D.4 Extension to preconditioned CG

While CG is an iterative algorithm, for simplicity we define it as the solution to a minimization problem in the Krylov subspace:

Definition 7.D.1. $\mathtt{CG}(\mathbf{A}, \mathbf{b}, \omega) = \min_{\|\mathbf{A}\mathbf{x}_k - \mathbf{b}\|_2 \leqslant \varepsilon} k$ for $\mathbf{x}_k = \underset{\mathbf{x}=\mathbf{P}_k(\breve{\mathbf{W}}_\omega^{-1}\mathbf{A})\breve{\mathbf{W}}_\omega^{-1}\mathbf{b}}{\arg\min} \|\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}\|_{\mathbf{A}}$ where the minimum is taken over all degree k polynomials $\mathbf{P}_k : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n\times n}$.

Lemma 7.D.2. Let $\mathbf{A}$ be a positive-definite matrix and $\mathbf{b} \in \mathbb{R}^n$ any vector. Define

$$U^{\mathtt{CG}}(\omega) = 1 + \frac{\tau \log\left(\frac{\sqrt{\kappa(\mathbf{A})}}{\varepsilon} + \sqrt{\frac{\kappa(\mathbf{A})}{\varepsilon^2} - 1}\right)}{-\log\left(1 - \frac{4}{2+\sqrt{\frac{4}{2-\omega} + \frac{\mu(2-\omega)}{\omega} + \frac{4\nu\omega}{2-\omega}}}\right)} \tag{7.31}$$

for $\mu = \lambda_{\max}(\mathbf{D}\mathbf{A}^{-1}) \geqslant 1$, $\nu = \lambda_{\max}((\mathbf{L}\mathbf{D}^{-1}\mathbf{L}^\top - \mathbf{D}/4)\mathbf{A}^{-1}) \in [-1/4, 0]$, and τ the smallest constant (depending on $\mathbf{A}$ and $\mathbf{b}$) s.t. $U^{\mathtt{CG}} \geqslant \mathtt{CG}(\mathbf{A}, \mathbf{b}, \cdot)$. Then the following holds

1. $\tau \in (0, 1]$
2. if $\mu > 1$ then $U^{\mathtt{CG}}$ is minimized at $\omega^* = \frac{2}{1+\sqrt{\frac{2}{\mu}(1+2\nu)}}$ and monotonically increases away from ω^* in both directions
3. $U^{\mathtt{CG}}$ is $\left(\frac{\mu+4\nu+4}{4\mu\nu+2\mu-1}\tau\sqrt{\mu\sqrt{2}}, 2\sqrt{2}-2\right)$-semi-Lipschitz on $[2\sqrt{2}-2, 2)$
4. if $\mu \leqslant \mu_{\max}$ then $U^{\mathtt{CG}} \leqslant 1 + \frac{\tau\log(\frac{2}{\varepsilon}\sqrt{\kappa(\mathbf{A})})}{-\log\left(1-\frac{2}{1+\sqrt{\gamma}}\right)}$ on $[2\sqrt{2}-2, \frac{2}{1+1/\sqrt{\mu_{\max}}}]$, where $\gamma \leqslant \frac{7+3\mu_{\max}}{8}$.

Proof. By Hackbusch [2016, Theorem 10.17], the kth residual of SSOR-preconditioned CG

satisfies

$$\|\mathbf{r}_k(\omega)\|_2 = \|\mathbf{b}-\mathbf{A}\mathbf{x}_k\|_2 \leqslant \sqrt{\|\mathbf{A}\|_\infty}\|\mathbf{A}^{-1}\mathbf{b}-\mathbf{x}_k\|_{\mathbf{A}} \leqslant \sqrt{\|\mathbf{A}\|_\infty}\frac{2x^k}{1+x^{2k}}\|\mathbf{A}^{-1}\mathbf{b}-\mathbf{x}_0\|_{\mathbf{A}} \leqslant \frac{2\sqrt{\kappa(\mathbf{A})}x^k}{1+x^{2k}}\|\mathbf{r}_0\|_2 \tag{7.32}$$

for $x = \frac{\sqrt{\kappa(\breve{\mathbf{W}}_\omega^{-1}\mathbf{A})}-1}{\sqrt{\kappa(\breve{\mathbf{W}}_\omega^{-1}\mathbf{A})}+1} = 1 - \frac{2}{\sqrt{\kappa(\breve{\mathbf{W}}_\omega^{-1}\mathbf{A})}+1}$. By Axelsson [1994, Theorem 7.17] we have

$$\kappa(\breve{\mathbf{W}}_\omega^{-1}\mathbf{A}) \leqslant \frac{1+\frac{\mu}{4\omega}(2-\omega)^2+\omega\nu}{2-\omega} \tag{7.33}$$

Combining the two inequalities above yields the first result. For the second, we compute the derivative w.r.t. ω:

$$\frac{\partial_\omega U^{\mathrm{CG}}}{\tau} = \frac{8(2\nu+1)\omega^2 - 4\mu(2-\omega)^2}{(2-\omega)\omega\sqrt{\frac{4}{2-\omega}+\frac{\mu(2-\omega)}{\omega}+\frac{4\nu\omega}{2-\omega}}(\mu(2-\omega)^2+4\omega(\nu\omega+\omega-1))} \tag{7.34}$$

Since $\nu \in [-1/4, 0]$ and $\mu > 1$, we have that $\mu(2-\omega)^2+4\omega(\nu\omega+\omega-1) \geqslant (2-\omega)^2+3\omega^2-4\omega$, which is nonnegative. Therefore the derivative only switches signs once, at the zero of specified in the second result. The monotonic increase property follows by positivity of the numerator on $\omega > \omega^*$. The third property follows by noting that since U^{CG} is increasing on $\omega > \omega^*$ we only needs to consider $\omega \in [2\sqrt{2}-2, \omega^*]$, where the numerator of the derivative is negative; here we have

$$\frac{|\partial_\omega U^{\mathrm{CG}}|}{\tau} \leqslant \frac{4\mu(2-\omega)}{\omega\sqrt{\frac{4}{2-\omega}+\frac{\mu(2-\omega)}{\omega}+\frac{4\nu\omega}{2-\omega}}(\mu(2-\omega)^2+4\omega(\nu\omega+\omega-1))} \leqslant \frac{\mu+4\nu+4}{4\mu\nu+2\mu-1}\sqrt{\mu\sqrt{2}} \tag{7.35}$$

where we have used $\mu(2-\omega)^2+4\omega(\nu\omega+\omega-1) \geqslant \frac{16\mu\nu+8\mu-4}{\mu+4\nu+4}$ and $\omega \geqslant 2\sqrt{2}-2$. For the last result we use the fact that $\kappa(\breve{\mathbf{W}}_\omega^{-1}\mathbf{A})$ is maximal at the endpoints of the interval and evaluate it on those endpoints to bound $\gamma \leqslant \frac{1}{2} + \frac{\max\{(\mu_{\max}+1)\sqrt{2}, 3\sqrt{\mu_{\max}}\}}{4} \leqslant \frac{1}{2} + \frac{3(\mu_{\max}+1)}{8}$. □

We plot the bounds from Lemma 7.D.2 in Figure 7.9. Note that $\tau \in (0, 1]$ is an instance-dependent parameter that is defined to effectively scale down the function as much as possible while still being an upper bound on the cost; it thus allows us to exploit the shape of the upper bound without having it be too loose. This is useful since upper bounds for CG are known to be rather pessimistic, and we are able to do this because our learning algorithms do not directly access the upper bound anyway. Empirically, we find τ to often be around $3/4$ or larger.

Proof of Theorem 7.3.3

Proof. By Lemma 7.D.2 the functions $U_t - 1 \geqslant \mathsf{CG}_t - 1$ are $\left(\frac{\mu_t+4\nu_t+4}{4\mu_t\nu_t+2\mu_t-1}\sqrt{\mu_t\sqrt{2}}, 2\sqrt{2}-2\right)$-semi-Lipschitz and $\frac{\log(\frac{2}{\varepsilon}\sqrt{\kappa_{\max}})}{\log\frac{\sqrt{6\mu_{\max}+14}+4}{\sqrt{6\mu_{\max}+14}-4}}$-bounded on $[2\sqrt{2}-2, \frac{2}{1+1/\sqrt{\mu_{\max}}}]$; note that by the assumption

on $\min_t \mu_t$ and the fact that $\nu_t \geqslant 1/4$ the semi-Lipschitz constant is $\mathcal{O}(\sqrt{\mu_t})$. Therefore the desired regret w.r.t. any $\omega \in [2\sqrt{2}-2, \frac{2}{1+1/\sqrt{\mu_{\max}}}]$ follows, and extends to the rest of the interval because Lemma 7.D.2.2 also implies all functions U_t are increasing away from this interval. □

7.E Semi-stochastic proofs

7.E.1 Regularity of the criterion

Lemma 7.E.1. $\|\breve{\mathbf{C}}_\omega^k \mathbf{b}\|_2$ is $\rho(\breve{\mathbf{C}}_\omega)^{k-1}\|\mathbf{b}\|_2 k\sqrt{\kappa(\mathbf{A})}\left(\frac{1}{2-\omega_{\max}} + 2\rho(\mathbf{D}\mathbf{A}^{-1})\right)$-Lipschitz w.r.t. $\omega \in \Omega$.

Proof. Taking the derivative, we have that

$$
\begin{aligned}
|\partial_\omega \|\breve{\mathbf{C}}_\omega^k \mathbf{b}\|_2| &= \frac{|\partial_\omega[(\breve{\mathbf{C}}_\omega^k \mathbf{b})^\top \breve{\mathbf{C}}_\omega^k \mathbf{b}]|}{\|\breve{\mathbf{C}}_\omega^k \mathbf{b}\|_2} \\
&= \frac{\left|(\breve{\mathbf{C}}_\omega^k \mathbf{b})^\top \sum_{i=1}^k \left[\breve{\mathbf{C}}_\omega^{i-1}(\partial_\omega \breve{\mathbf{C}}_\omega)\breve{\mathbf{C}}_\omega^{k-i}\mathbf{b}\right]\right|}{\|\breve{\mathbf{C}}_\omega^k \mathbf{b}\|_2} \\
&\leqslant \left\|\sum_{i=1}^k \breve{\mathbf{C}}_\omega^{i-1}(\partial_\omega \breve{\mathbf{C}}_\omega)\breve{\mathbf{C}}_\omega^{k-i}\mathbf{b}\right\|_\infty \\
&\leqslant \sum_{i=1}^k \left\|\breve{\mathbf{C}}_\omega^{i-1}(\partial_\omega \breve{\mathbf{C}}_\omega)\breve{\mathbf{C}}_\omega^{k-i}\mathbf{b}\right\|_\infty \\
&= \sum_{i=1}^k \left\|\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{-\frac{1}{2}}\breve{\mathbf{C}}_\omega^{i-1}\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{-\frac{1}{2}}(\partial_\omega \breve{\mathbf{C}}_\omega)\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{-\frac{1}{2}}\breve{\mathbf{C}}_\omega^{k-i}\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{-\frac{1}{2}}\mathbf{b}\right\|_\infty \\
&\leqslant \|\mathbf{b}\|_2\sqrt{\kappa(\mathbf{A})}\sum_{i=1}^k \|\!|(\mathbf{A}^{-\frac{1}{2}}\breve{\mathbf{C}}_\omega\mathbf{A}^{\frac{1}{2}})^{i-1}|\!\|_\infty \|\!|\mathbf{A}^{-\frac{1}{2}}(\partial_\omega \breve{\mathbf{C}}_\omega)\mathbf{A}^{\frac{1}{2}}|\!\|_\infty \|\!|(\mathbf{A}^{-\frac{1}{2}}\breve{\mathbf{C}}_\omega\mathbf{A}^{\frac{1}{2}})^{k-i}|\!\|_\infty \\
&= \rho(\breve{\mathbf{C}}_\omega)^{k-1}\|\mathbf{b}\|_2 k \|\!|\mathbf{A}^{-\frac{1}{2}}(\partial_\omega \breve{\mathbf{C}}_\omega)\mathbf{A}^{\frac{1}{2}}|\!\|_\infty \sqrt{\kappa(\mathbf{A})}
\end{aligned}
\tag{7.36}
$$

where the first inequality is due to Cauchy-Schwartz, the second is the triangle inequality, and the third is due to the sub-multiplicativity of the norm. The last line follows by symmetry of $\mathbf{A}^{-\frac{1}{2}}\breve{\mathbf{C}}_\omega\mathbf{A}^{\frac{1}{2}}$, which implies that the spectral norm of any of power equals that power of its spectral radius, which by similarity is also the spectral radius of $\breve{\mathbf{C}}_\omega$. Next we use a matrix calculus

tool [Laue et al., 2018] to compute

$$\begin{aligned}\partial_\omega \breve{\mathbf{C}}_\omega &= \left(\frac{1}{\omega} + \frac{2-\omega}{\omega^2}\right)\mathbf{A}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1} \\ &\quad - \frac{2-\omega}{\omega^3}\mathbf{A}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1} \\ &\quad + \frac{2-\omega}{\omega^3}\mathbf{A}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1} \\ &= \left(\frac{1}{2-\omega} + \frac{1}{\omega}\right)\mathbf{A}\breve{\mathbf{W}}_\omega^{-1} - \frac{1}{\omega^2}\mathbf{A}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}\breve{\mathbf{W}}_\omega^{-1} - \frac{1}{\omega^2}\mathbf{A}\breve{\mathbf{W}}_\omega^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1}\end{aligned} \tag{7.37}$$

so since $\|\mathbf{A}^{\frac{1}{2}}\breve{\mathbf{W}}_\omega^{-1}\mathbf{A}^{\frac{1}{2}}\|_\infty = \|\mathbf{I}_n - \mathbf{A}^{-\frac{1}{2}}\breve{\mathbf{C}}_\omega\mathbf{A}^{\frac{1}{2}}\|_\infty \leqslant 1 + \rho(\mathbf{A}^{-\frac{1}{2}}\breve{\mathbf{C}}_\omega\mathbf{A}^{\frac{1}{2}}) \leqslant 2$ and

$$\begin{aligned}\|\mathbf{A}^{\frac{1}{2}}(\mathbf{D}/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}\breve{\mathbf{W}}_\omega^{-1}\mathbf{A}^{\frac{1}{2}}\|_\infty &= \|\mathbf{A}^{\frac{1}{2}}\breve{\mathbf{W}}_\omega^{-1}\mathbf{D}(\mathbf{D}/\omega + \mathbf{L})^{-1}\mathbf{A}^{\frac{1}{2}}\|_\infty \\ &= \|\breve{\mathbf{W}}_\omega^{-1}\mathbf{D}\mathbf{W}_\omega^{-1}\mathbf{A}\|_{\mathbf{A}} \\ &\leqslant \|\breve{\mathbf{W}}_\omega^{-1}\mathbf{D}\|_{\mathbf{A}}\|\mathbf{I}_n - \mathbf{M}_\omega\|_{\mathbf{A}} \\ &\leqslant 2\|\breve{\mathbf{W}}_\omega^{-1}\mathbf{A}\|_{\mathbf{A}}\|\mathbf{A}^{-\frac{1}{2}}\mathbf{D}\mathbf{A}^{-\frac{1}{2}}\|_\infty \\ &= 2\rho(\mathbf{D}\mathbf{A}^{-1})\|\mathbf{I}_n - \breve{\mathbf{M}}_\omega\|_{\mathbf{A}} \leqslant 4\rho(\mathbf{D}\mathbf{A}^{-1})\end{aligned} \tag{7.38}$$

we have by applying $\omega \in [1, \omega_{\text{max}}]$ that

$$\|\mathbf{A}^{-\frac{1}{2}}(\partial_\omega\breve{\mathbf{C}}_\omega)\mathbf{A}^{\frac{1}{2}}\|_\infty \leqslant \frac{2}{2-\omega} + \frac{2}{\omega} + \frac{8\rho(\mathbf{D}\mathbf{A}^{-1})}{\omega^2} \leqslant \frac{4}{2-\omega_{\text{max}}} + 8\rho(\mathbf{D}\mathbf{A}^{-1}) \tag{7.39}$$

□

Lemma 7.E.2. $\|\breve{\mathbf{C}}_\omega^k(c)\mathbf{b}\|_2$ is $\frac{10}{\lambda_{\min}(\mathbf{A})+c_{\min}}\rho(\breve{\mathbf{C}}_\omega)^{k-1}(c)\|\mathbf{b}\|_2 k\sqrt{\kappa(\mathbf{A})}$-Lipschitz w.r.t. all $c \geqslant c_{\min} > -\lambda_{\min}(\mathbf{A}(c))$, where (c) denotes matrices derived from $\mathbf{A}(c) = \mathbf{A} + c\mathbf{I}_n$.

Proof. We take the derivative as in the above proof of Lemma 7.E.1:

$$|\partial_c\|\breve{\mathbf{C}}_\omega^k(c)\mathbf{b}\|_2| = \rho(\breve{\mathbf{C}}_\omega(c))^{k-1}\|\mathbf{b}\|_2 k\|\mathbf{A}^{-\frac{1}{2}}(c)(\partial_c\breve{\mathbf{C}}_\omega(c))\mathbf{A}^{\frac{1}{2}}(c)\|_\infty\sqrt{\kappa(\mathbf{A}(c))} \tag{7.40}$$

We then again apply the matrix calculus tool of Laue et al. [2018] to get

$$\begin{aligned}\partial_c\breve{\mathbf{C}}_\omega(c) &= -\frac{2-\omega}{\omega}(\mathbf{D}(c)/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(c)(\mathbf{D}(c)/\omega + \mathbf{L})^{-1} \\ &\quad + \frac{2-\omega}{\omega^2}\mathbf{A}(c)(\mathbf{D}(c)/\omega + \mathbf{L}^\top)^{-2}\mathbf{D}(c)(\mathbf{D}(c)/\omega + \mathbf{L})^{-1} \\ &\quad - \frac{2-\omega}{\omega}\mathbf{A}(c)(\mathbf{D}(c)/\omega + \mathbf{L}^\top)^{-1}(\mathbf{D}(c)/\omega + \mathbf{L})^{-1} \\ &\quad + \frac{2-\omega}{\omega^2}\mathbf{A}(c)(\mathbf{D}(c)/\omega + \mathbf{L}^\top)^{-1}\mathbf{D}(c)(\mathbf{D}(c)/\omega + \mathbf{L})^{-2} \\ &= -\frac{2-\omega}{\omega}(\breve{\mathbf{W}}_\omega^{-1}(c) + \mathbf{A}(c)\mathbf{W}_\omega^{-T}(c)\mathbf{W}_\omega^{-1}(c)) \\ &\quad + \frac{2-\omega}{\omega^2}\mathbf{A}(c)(\mathbf{W}_\omega^{-T}(c)\breve{\mathbf{W}}_\omega^{-1}(c) + \breve{\mathbf{W}}_\omega^{-1}(c)\mathbf{W}_\omega^{-1}(c))\end{aligned} \tag{7.41}$$

Algorithm 23: Symmetric SOR with absolute convergence condition.

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, parameter $\omega \in (0,2)$, initial vector $\mathbf{x} \in \mathbb{R}^n$, tolerance $\varepsilon > 0$

$\mathbf{D} + \mathbf{L} + \mathbf{L}^\top \leftarrow \mathbf{A}$ `// D diagonal, L strictly lower triangular`

$\breve{\mathbf{W}}_\omega \leftarrow \frac{\omega}{2-\omega}(\mathbf{D}/\omega + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D}/\omega + \mathbf{L}^\top)$ `// compute third normal form`

$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$ `// compute initial residual`

for $k = 0, \ldots$ **do**

  if $\|\mathbf{r}_k\|_2 > \varepsilon$ **then**

    return k `// return iteration count (for use in learning)`

  $\mathbf{x} = \mathbf{x} + \breve{\mathbf{W}}_\omega^{-1}\mathbf{r}_k$ `// solve triangular systems and update vector`

  $\mathbf{r}_{k+1} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$ `// compute the next residual`

Output: k

By symmetry of $\breve{\mathbf{W}}_\omega^{-1}(c)$ we have

$$\begin{aligned}\|\mathbf{A}^{-\frac{1}{2}}(c)\breve{\mathbf{W}}_\omega^{-1}(c)\mathbf{A}^{\frac{1}{2}}(c)\|_\infty = \|\breve{\mathbf{W}}_\omega^{-1}(c)\|_{\mathbf{A}(c)} &\leqslant \|\breve{\mathbf{W}}_\omega^{-1}(c)\mathbf{A}(c)\|_{\mathbf{A}(c)}\|\mathbf{A}^{-1}(c)\|_\infty \\ &= \|\mathbf{I}_n - \breve{\mathbf{M}}_\omega(c)\|_{\mathbf{A}(c)}\rho(\mathbf{A}^{-1}(c)) \leqslant 2\rho(\mathbf{A}^{-1}(c))\end{aligned} \tag{7.42}$$

Furthermore

$$\begin{aligned}\|\mathbf{A}^{\frac{1}{2}}(c)\mathbf{W}_\omega^{-T}(c)\mathbf{W}_\omega^{-1}(c)\mathbf{A}^{\frac{1}{2}}(c)\|_\infty &= \|\mathbf{A}^{-\frac{1}{2}}(c)(\mathbf{I}_n - \mathbf{M}_\omega^\top(c))(\mathbf{I}_n - \mathbf{M}_\omega(c))\mathbf{A}^{-\frac{1}{2}}(c)\|_\infty \\ &\leqslant \|\mathbf{A}^{-1}\|_\infty\|\mathbf{I}_n - \mathbf{M}_\omega(c)\|^2_{\mathbf{A}(c)} \leqslant 4\rho(\mathbf{A}^{-1}(c))\end{aligned} \tag{7.43}$$

and

$$\|\mathbf{A}^{\frac{1}{2}}(c)\mathbf{W}_\omega^{-T}(c)\breve{\mathbf{W}}_\omega^{-1}(c)\mathbf{A}^{\frac{1}{2}}(c)\|_\infty = \|\mathbf{A}^{\frac{1}{2}}(c)\breve{\mathbf{W}}_\omega^{-1}(c)\mathbf{W}_\omega^{-1}(c)\mathbf{A}^{\frac{1}{2}}(c)\|_\infty \leqslant 4\rho(\mathbf{A}^{-1}(c)) \tag{7.44}$$

so by the lower bound of $\frac{1}{\lambda_{\min}(\mathbf{A})+c_{\min}}$ on $\rho(\mathbf{A}^{-1}(c))$ we have the result. □

7.E.2 Anti-concentration

Lemma 7.E.3. Let $\mathbf{X} \in \mathbb{R}^{n\times n}$ be a nonzero matrix and $\mathbf{b} = m\mathbf{u}$ be a product of independent random variables $m \geqslant 0$ and $\mathbf{u} \in \mathbb{R}^n$ with $m^2 \in [0,n]$ a χ^2-squared random variable with n degrees of freedom truncated to the interval $[0,n]$ and $\mathbf{u}$ distributed uniformly on the surface of the unit sphere. Then for any interval $I = (\varepsilon, \varepsilon+\Delta] \subset \mathbb{R}$ for $\varepsilon, \Delta > 0$ we have that $\Pr(\|\mathbf{X}\mathbf{b}\|_2 \in I) \leqslant \frac{2\Delta}{\rho(\mathbf{X})}\sqrt{\frac{2}{\pi}}$.

Proof. Let f be the p.d.f. of $\mathbf{b}$ and g be the p.d.f. of $\mathbf{g} \sim \mathcal{N}(\mathbf{0}_n, \mathbf{I}_n)$. Then by the law of total

probability and the fact that $\mathbf{b}$ follows the distribution of $\mathbf{g}$ conditioned on $\|\mathbf{b}\|_2^2 \leqslant n$ we have that

$$\begin{aligned}
\Pr(\|\mathbf{X}\mathbf{b}\|_2 \in I) &= \int_{\|\mathbf{x}\|_2^2 \leqslant n} \Pr(\|\mathbf{X}\mathbf{b}\|_2 \in I | \mathbf{b} = \mathbf{x}) df(\mathbf{x}) \\
&= \frac{\int_{\|\mathbf{x}\|_2^2 \leqslant n} \Pr(\|\mathbf{X}\mathbf{b}\|_2 \in I | \mathbf{b} = \mathbf{x}) dg(\mathbf{x})}{\int_{\|\mathbf{x}\|_2^2 > n} dg(\mathbf{x})} \\
&\leqslant 2 \int_{\|\mathbf{x}\|_2^2 \leqslant n} \Pr(\|\mathbf{X}\mathbf{g}\|_2 \in I | \mathbf{g} = \mathbf{x}) dg(\mathbf{x}) \\
&\leqslant 2 \int_{\mathbb{R}^n} \Pr(\|\mathbf{X}\mathbf{g}\|_2 \in I | \mathbf{g} = \mathbf{x}) dg(\mathbf{x}) = 2 \Pr(\|\mathbf{X}\mathbf{g}\|_2 \in I)
\end{aligned} \tag{7.45}$$

where the second inequality uses the fact that a χ^2 random variable with n degrees of freedom has more than half of its mass below n. Defining the orthogonal diagonalization $\mathbf{Q}^\top \mathbf{\Lambda} \mathbf{Q} = \mathbf{X}^\top \mathbf{X}$ and noting that $\mathbf{Q}\mathbf{g} \sim \mathcal{N}(\mathbf{0}_n, \mathbf{I}_n)$, we then have that

$$\|\mathbf{X}\mathbf{g}\|_2^2 = (\mathbf{Q}\mathbf{g})^\top \mathbf{\Lambda} \mathbf{Q}\mathbf{g} = \sum_{i=1}^n \mathbf{\Lambda}_{[i,i]} \chi_i^2 \tag{7.46}$$

for i.i.d. $\chi_1, \ldots, \chi_n \sim \mathcal{N}(0,1)$. Let h, h_1, and h_{-1} be the densities of $\sum_{i=1}^n \mathbf{\Lambda}_{[i,i]} \chi_i^2$, $\mathbf{\Lambda}_{[1,1]} \chi_1^2$, and $\sum_{i=2}^n \mathbf{\Lambda}_{[i,i]} \chi_i^2$, respectively, and let $u(a)$ be the uniform measure on the interval $(a, a + 2\varepsilon\Delta + \Delta^2]$. Then since the density of the sum of independent random variables is their convolution, we can apply Young's inequality to obtain

$$\begin{aligned}
\Pr(\|\mathbf{X}\mathbf{g}\|_2 \in I) &= \Pr(\|\mathbf{X}\mathbf{g}\|_2^2 \in (\varepsilon^2, (\varepsilon + \Delta)^2]) \\
&\leqslant \max_{a \geqslant \varepsilon^2} \int_a^{a + 2\varepsilon\Delta + \Delta^2} h(x) dx \\
&= \max_{a \geqslant \varepsilon^2} \int_{-\infty}^{\infty} u(x - a) h(x) dx \\
&= \|u * h\|_{L^\infty([\varepsilon, \infty))} \\
&= \|u * h_1 * h_{-1}\|_{L^\infty([\varepsilon, \infty))} \\
&\leqslant \|u * h_1\|_{L^\infty([\varepsilon, \infty))} \|h_{-1}\|_{L^1([\varepsilon, \infty))} \\
&\leqslant \max_{a \geqslant \varepsilon^2} \int_a^{a + 2\varepsilon\Delta + \Delta^2} h_1(x) dx \\
&= \max_{a \geqslant \varepsilon} \int_a^{a + 2\varepsilon\Delta + \Delta^2} \frac{e^{-\frac{x}{2\mathbf{\Lambda}_{[1,1]}}}}{\sqrt{2\pi \mathbf{\Lambda}_{[1,1]} x}} dx \\
&\leqslant \max_{a \geqslant \varepsilon^2} \sqrt{\frac{2(a + 2\varepsilon\Delta + \Delta^2)}{\pi \mathbf{\Lambda}_{[1,1]}}} - \sqrt{\frac{2a}{\pi \mathbf{\Lambda}_{[1,1]}}} = \Delta \sqrt{\frac{2}{\pi \mathbf{\Lambda}_{[1,1]}}}
\end{aligned} \tag{7.47}$$

Substituting into the first equation and using $\mathbf{\Lambda}_{[i,i]} = \|\mathbf{X}\|_\infty^2 \geqslant \rho(\mathbf{X})^2$ yields the result. □

7.E.3 Lipschitz expectation

Lemma 7.E.4. Suppose $\mathbf{b} = m\mathbf{u}$, where m and $\mathbf{u}$ are independent random variables with $\mathbf{u}$ distributed uniformly on the surface of the unit sphere and $m^2 \in [0, n]$ a χ^2-squared random variable with n degrees of freedom truncated to the interval $[0, n]$. Define K as in Corollary 7.C.2, $\beta = \min_x \rho(\mathbf{I}_n - \mathbf{D}_x^{-1}\mathbf{A}_x)$, and $\texttt{SSOR}(x) = \min_{\|\breve{\mathbf{C}}_x^k \mathbf{b}\|_2 \leqslant \varepsilon} k$ to be the number of iterations to convergence when the defect reduction matrix depends on some scalar $x \in \mathcal{X}$ for some bounded interval $\mathcal{X} \subset \mathbb{R}$. If $\|\breve{\mathbf{C}}_x^k \mathbf{b}\|_2$ is $L\rho(\breve{\mathbf{C}}_x)^{k-1}$-Lipschitz a.s. w.r.t. any $x \in \mathcal{X}$ then $\mathbb{E}\texttt{SSOR}$ is $\frac{32K^3L\sqrt{2/\pi}}{\beta^4}$-Lipschitz w.r.t. x.

Proof. First, note that by Hackbusch [2016, Theorem 6.26]

$$\rho(\breve{\mathbf{C}}_x) = \rho(\breve{\mathbf{M}}_x) = \|\breve{\mathbf{M}}_x^2\|_{\mathbf{A}_x} \geqslant \rho(\mathbf{M}_x)^2 \geqslant \left(\frac{\beta}{1 + \sqrt{1 - \beta^2}}\right)^4 \geqslant \frac{\beta^4}{16} \tag{7.48}$$

Now consider any $x_1, x_2 \in \mathcal{X}$ s.t. $|x_1 - x_2| \leqslant \frac{\varepsilon\beta^4\sqrt{\pi/2}}{2K^3L}$, assume w.l.o.g. that $x_1 < x_2$, and pick $x' \in [x_1, x_2]$ with maximal $\rho(\breve{\mathbf{C}}_x)$. Then setting $\rho_{x'} = \rho(\breve{\mathbf{C}}_{x'})$ we have that $\|\breve{\mathbf{C}}_{x_i}^k \mathbf{b}\|_2$ is $L\rho_{x'}^{k-1}$-Lipschitz for both $i = 1, 2$ and all $k \in [K]$. Therefore starting with Jensen's inequality we have that

$$\begin{aligned}
&|\mathbb{E}\texttt{SSOR}(x_i) - \mathbb{E}\texttt{SSOR}(x')| \\
&\leqslant \mathbb{E}|\texttt{SSOR}(x_i) - \texttt{SSOR}(x')| \\
&= \sum_{k=1}^{K}\sum_{l=1}^{K} |k - l| \Pr(\texttt{SSOR}(x_i) = k \cap \texttt{SSOR}(x') = l) \\
&\leqslant K \sum_{k=1}^{K} \left(\sum_{l<k} \Pr(\|\breve{\mathbf{C}}_{x_i}^l \mathbf{b}\|_2 > \varepsilon \cap \|\breve{\mathbf{C}}_{x'}^l \mathbf{b}\|_2 \leqslant \varepsilon) + \sum_{l>k} \Pr(\|\breve{\mathbf{C}}_{x_i}^k \mathbf{b}\|_2 \leqslant \varepsilon \cap \|\breve{\mathbf{C}}_{x'}^k \mathbf{b}\|_2 > \varepsilon) \right) \\
&\leqslant K \sum_{k=1}^{K} \sum_{l<k} \Pr(\|\breve{\mathbf{C}}_{x'}^l \mathbf{b}\|_2 \in (\varepsilon - L\rho_{x'}^{l-1}|x_i - x'|, \varepsilon]) \\
&\quad + K \sum_{k=1}^{K} \sum_{l>k} \Pr(\|\breve{\mathbf{C}}_{x'}^k \mathbf{b}\|_2 \in (\varepsilon, \varepsilon + L\rho_{x'}^{k-1}|x_i - x'|]) \\
&\leqslant K \sum_{k=1}^{K} \left(\sum_{l<k} \frac{2L\rho_{x'}^{l-1}\sqrt{2/\pi}}{\rho(\breve{\mathbf{C}}_{x'}^l)} |x_i - x'| + \sum_{k=1}^{K}\sum_{l>k} \frac{2L\rho_{x'}^{k-1}\sqrt{2/\pi}}{\rho(\breve{\mathbf{C}}_{x'}^k)} |x_i - x'| \right) \\
&\leqslant \frac{2K^3L\sqrt{2/\pi}}{\rho_{x'}} |x_i - x'| \leqslant \frac{32K^3L\sqrt{2/\pi}}{\beta^4}
\end{aligned} \tag{7.49}$$

where the second inequality follows by the definition of $\texttt{SSOR}$, the third by Lipschitzness, and the fourth by the anti-concentration result of Lemma 7.E.3. Since this holds for any nearby pairs $x_1 < x_2$, taking the summation over the interval $\mathcal{X}$ completes the proof. □

Corollary 7.E.1. Under the assumptions of Lemma 7.D.1, the function $\mathbb{E}_{\mathbf{b}}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega)$ is $\frac{32K^4\sqrt{2n\kappa(\mathbf{A})/\pi}}{\beta^4}\left(\frac{1}{2-\omega_{\max}} + 2\rho(\mathbf{D}\mathbf{A}^{-1})\right)$-Lipschitz w.r.t. $\omega \in [1, \omega_{\max}] \subset (0, 2)$.

Proof. Apply Lemmas 7.E.1 and 7.D.1, noting that $\|\mathbf{b}\|_2 \leqslant \sqrt{n}$ by definition. □

Corollary 7.E.2. Under the assumptions of Lemma 7.D.1, the function $\mathbb{E}_{\mathbf{b}}\texttt{SSOR}(\mathbf{A}(c), \mathbf{b}, \omega)$ is $\max_c \frac{320K^4\sqrt{2n\kappa(\mathbf{A}(c))/\pi}}{\beta^4(\lambda_{\min}(\mathbf{A})+c_{\min})}$-Lipschitz w.r.t. $c \geqslant c_{\min} > -\lambda_{\min}$.

Proof. Apply Lemma 7.E.2 and 7.D.1, noting that $\|\mathbf{b}\|_2 \leqslant \sqrt{n}$ by definition. □

7.E.4 Sample complexity

Proof of Corollary 7.4.1

Proof. A standard covering bound (see e.g. Lafferty et al. [2010, Theorem 7.82]) followed by an application of Corollary 7.E.1 implies that w.p. $\geqslant 1 - \delta$

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \hat{\omega}) &\leqslant \min_{\omega \in \mathbf{g}} \mathbb{E}_{\mathcal{D}}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega) + 3K\sqrt{\frac{2}{T}\log\frac{2d}{\delta}} \\
&= \min_{\omega \in \mathbf{g}} \mathbb{E}_{\mathbf{A}}[\mathbb{E}_{\mathbf{b}}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega)|\mathbf{A}] + 3K\sqrt{\frac{2}{T}\log\frac{2d}{\delta}} \\
&\leqslant \min_{\omega \in [1, \omega_{\max}]} \mathbb{E}_{\mathbf{A}}\left[\mathbb{E}_{\mathbf{b}}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega) + \frac{L}{d}\middle|\mathbf{A}\right] + 3K\sqrt{\frac{2}{T}\log\frac{2d}{\delta}} \qquad (7.50) \\
&= \min_{\omega \in [1, \omega_{\max}]} \mathbb{E}_{\mathcal{D}}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega) + \frac{L}{d} + 3K\sqrt{\frac{2}{T}\log\frac{2d}{\delta}} \\
&\leqslant \min_{\omega \in [1, \omega_{\max}]} \mathbb{E}_{\mathcal{D}}\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega) + 4K\sqrt{\frac{2}{T}\log\frac{2LT}{K\delta}}
\end{aligned}$$

Noting that by Corollary 7.E.1 we have $L = \mathcal{O}(K^4\sqrt{n}) = \mathcal{O}(\sqrt{n}\log^4\frac{n}{\varepsilon})$ yields the result. □

Proof of Corollary 7.4.2

Proof. For every $(\mathbf{A}, \mathbf{b})$ pair in the support of $\mathcal{D}$ and any $r \in \mathbb{R}$ it is straightforward to define a GJ algorithm [Bartlett et al., 2022, Definition 3.1] that checks if $\texttt{SSOR}(\mathbf{A}, \mathbf{b}, \omega) > r$ by computing $\|\mathbf{r}_k(\omega)\|_2^2 = \|\check{\mathbf{C}}_\omega^k\mathbf{b}\|_2^2$—a degree $2k$ polynomial—for every $k \leqslant \lfloor r \rfloor$ and returning "True" if one of them satisfies $\|\mathbf{r}_k(\omega)\|_2^2 \leqslant \varepsilon^2$ and "False" otherwise (and automatically return "True" for $r \geqslant K$ and "False" for $r < 1$). Since the degree of this algorithm is at most $2K$, the predicate complexity is at most K, and the parameter size is 1, by Bartlett et al. [2022, Theorem 3.3] the pseudodimension of $\{\texttt{SSOR}(\cdot, \cdot, \omega) : \omega \in [1, \omega_{\max}]\}$ is $\mathcal{O}(\log K)$. Using the bounded assumption on the target vector—$\texttt{SSOR} \leqslant K = \mathcal{O}(\log\frac{n}{\varepsilon})$—-completes the proof. □

7.F Experimental details

All numerical results were generated in MATLAB on a laptop and can be re-generated by running the scripts available at `https://github.com/mkhodak/learning-to-relax`.

7.F.1 Algorithmic modifications

Since we do not have access to problem parameters, we experimented with a few approaches to setting them automatically or heuristically on the simplest (low variance) setting below and then used the same settings for the rest of the experiments (high variance and heat equation). Furthermore, because the default step-size/learning rate settings in both algorithms are rather pessimistic, we use more aggressive time-varying approaches in practice. For Tsallis-INF we set $\eta_t = 2/\sqrt{t}$, which is what is used in the anytime variant [Zimmert and Seldin, 2021]. As for ChebCB, we use an increasing schedule $\eta_t = \mathcal{O}(t)$; note that Simchi-Levi and Xu [2021] also use an increasing learning rate schedule for setting inverse gap-weighted probabilities.

7.F.2 Initial conditions and forcing for the 2D heat simulation

Recall that we are aiming to solve the 2D heat equation (7.11) with periodically-varying diffusion coefficient $\kappa(t) = \max\{0.01\sin(2\pi t)), -10\sin(2\pi t)\}$. We set the initial conditions to be $u(0, \mathbf{x}) = \mathbf{b}_{(\frac{1}{2}\ \frac{1}{2}),\frac{1}{4}}(\mathbf{x})$ and the forcing function to be $f(t, \mathbf{x}) = 32\mathbf{b}_{(\frac{1}{2}+\cos(16\pi t)/4, \frac{1}{2}+\cos(16\pi t)/4),1/8}(\mathbf{x})$, where $b_{\mathbf{c},r}(\mathbf{x})$ is a *bump function* centered at $\mathbf{c} \in \mathbb{R}^2$ with radius $r > 0$ and defined to be $\exp\left(-\frac{1}{1-\|\mathbf{x}-\mathbf{c}\|_2^2/r^2}\right)$ if $\|\mathbf{x} - \mathbf{c}\|_2 < r$ and 0 otherwise. This specific forcing function—effectively a bump circling around the center of the domain—is chosen to ensure that the linear system solutions are not too close to each other or to zero.

Part III

Architecture search

Chapter 8

Overview

The third and final part of this thesis is concerned with neural architecture search (NAS), a subfield of automated machine learning (AutoML) focused on finding suitable neural network architectures for learning tasks. While there has been a lot of empirical progress on designing search spaces and heuristic search algorithms for finding good networks for computer vision tasks, the theoretical understanding of these approximation methods has been minimal and their applicability to diverse types of data beyond images and text is limited. Following the current overview chapter, we make significant progress in both of these directions in the next two chapters: the first focuses on developing an understanding of the highly successful *weight-sharing* heuristic for searching discrete neural architecture spaces while the second designs efficient but expressive search spaces for discovering neural *operations* that perform well on diverse tasks.

8.1 Literature

Simple architectural hyperparameters such as network width have been a part of hyperparameter search spaces for some time [Bergstra and Bengio, 2012, Li et al., 2018a], but NAS became an important subfield as it became more clear that significant performance improvements could be realized through intricate architectural innovations that could only be captured through changes to neural operations and how they are connected [Zoph et al., 2018]. Although at first exorbitantly expensive because of the combinatorially large search spaces, important heuristics such as *weight-sharing* have achieved state-of-the-art performance while drastically reducing the computational cost of NAS to just that of training a single *shared-weights network* [Pham et al., 2018]. Methods such as DARTS [Liu et al., 2019b] combine weight-sharing with a continuous relaxation of the discrete search space to allow cheap gradient updates, enabling the use of popular optimizers. However, despite empirical success, weight-sharing remains poorly understood and has received criticism due to issue of rank disorder and poor results on recent benchmarks. At the same time, there has been some theoretical analysis of classical discrete search approaches [White et al., 2020, 2021].

On the empirical side, the development of NAS algorithms and especially search spaces has been centered closely on vision and—to a lesser degree—text tasks; for example, most NAS operation spaces only contain a few operations such as convolutions [Liu et al., 2019b, Mei et al.,

2020, Zela et al., 2020b, Dong and Yang, 2020], which may not be useful for domains where CNNs are ineffective. Applications of NAS outside vision largely follow the same pattern of combining human-designed operations [Nekrasov et al., 2019, Wang et al., 2020b]. On the other extreme, AutoML-Zero [Real et al., 2020] demonstrates the possibility of evolving all aspects of ML from scratch. This leaves significant room for a middle ground with large and domain-agnostic search spaces that still allow the use of well-tested methods based on weight-sharing and continuous relaxation.

8.2 Contributions

The architecture search contributions in this thesis start with first steps towards a mathematical understanding of weight-sharing in Chapter 9, which studies this critical NAS paradigm from the perspective of both optimization and generalization. While significant theoretical work remains, our investigation does inspire a more relaxed view of architectural parameters that allows for a significant expansion of the types of neural operations contained in architecture search spaces. We build upon this in Chapter 10, which focuses on designing search spaces that work for diverse tasks beyond vision and language processing.

8.2.1 Understanding weight-sharing

Motivated by the challenge of developing simple and efficient methods that achieve state-of-the-art performance, in Chapter 9 we study how to best handle the goals and optimization objectives of NAS. We start by observing that weight-sharing subsumes architecture hyperparameters as another set of learned parameters of the shared-weights network, in effect extending the class of functions being learned. We argue that NAS with weight-sharing can be studied not only as a bilevel optimization problem but also as a single-level objective in which architectural decisions are treated as learned parameters rather than hyperparameters. Our setup clarifies recent concerns about rank disorder and makes clear that proper regularization and optimization of the chosen objective is critical to obtaining high-quality solutions.

While many regularization approaches have been implicitly proposed in recent NAS efforts, we start by focusing instead on the question of optimizing architecture parameters, which may not be amenable to standard procedures such as SGD that work well for standard neural network weights. We propose to improve existing NAS algorithms by re-parameterizing architecture parameters over the simplex and updating them using exponentiated gradient, a variant of (offline) mirror descent [Nemirovski and Yudin, 1983, Beck and Teboulle, 2003] that converges quickly over this domain and enjoys favorable sparsity properties. This simple modification—which we call the **Geometry-Aware Exponentiated Algorithm (GAEA)**—is easily applicable to numerous methods, including popular NAS algorithms such as (first-order) DARTS [Liu et al., 2019b]. To show correctness and efficiency of our scheme, we prove polynomial-time stationary-point convergence of block-stochastic mirror descent—a family of geometry-aware gradient algorithms that includes GAEA—over a continuous relaxation of the single-level NAS objective. These are the first finite-time convergence guarantees for gradient-based NAS. Empirically, we demonstrate that GAEA improves upon high-performance NAS algorithms on several computer

vision benchmarks, including NAS-Bench-201 [Dong and Yang, 2020] and the the more challenging DARTS search space [Liu et al., 2019b], with evaluations on both CIFAR [Krizhevksy, 2009] and ImageNet [Russakovsky et al., 2015]. Together, our theory and experiments demonstrate a principled way to co-design optimizers and continuous relaxations of discrete NAS search spaces.

While this optimization-based analysis of weight-sharing focuses on the single-level objective, there remains the question of whether to use it or the bilevel objective in practice. Our last contribution in Chapter 9 proposes to use *feature map selection*—the problem of choosing which featurization of data to use in a downstream task such as linear regression or classification—as a simple setting for understanding weight-sharing. As with NAS, we show that weight-sharing provides a useful signal to evaluate many feature map configurations without individual training, giving an empirical justification for studying the paradigm through this lens. However, the resulting generalization analysis yields a sample complexity argument that favors bilevel rather than single-level optimization, suggesting a tradeoff between optimization and generalization in weight-sharing that practical algorithms should address.

8.2.2 Architecture search for diverse tasks

The success of deep learning for computer vision and natural language processing has spurred growing interest in enabling similar breakthroughs for other domains such as biology, healthcare, and the physical sciences. Indeed there is enormous potential for NAS to help automate model development in these diverse areas, but despite extensive research devoted to architecture search most of the approaches have subpar performance beyond the (usually vision) tasks on which they were developed. In Chapter 10, the last in this thesis, we use weight-sharing as a starting point for developing architecture search spaces for diverse tasks, using its relaxation of architectural parameters to search over larger spaces of candidate architectures. The hope is that these search spaces are both large-enough to contain the "right" neural operations for diverse tasks while being small enough to be efficiently and successfully searched.

XD-operations

We start with the observation that in weight-sharing we use a gradient algorithm to optimize a "supernet" objective that interpolates the set of possible operations on each edge in a neural network using a continuous relaxation, and that most past work simply uses a convex combination of operations to do so [Liu et al., 2019b]. This leads to a re-imagining of NAS operation spaces based upon an alternative continuous relaxation that exploits the fact that most commonly used operations such as convolutions return linear transforms diagonalized by the discrete Fourier transform (DFT). Replacing the DFT matrices in the diagonal decomposition by a more expressive family of efficient linear transforms known as *Kaleidoscope* or *K-matrices* [Dao et al., 2020] yields the set of **Expressive Diagonalization (XD) Operations**, which comprise a large search space containing various types of grid-based convolutions and pooling, permutations, certain kinds of graph convolutions, the Fourier Neural Operator (FNO) from the PDE literature [Li et al., 2021c], and infinitely many more.

We leverage XD-operations to take critical steps towards a broader NAS that enables the discovery of good design patterns with limited human specification from data in under-explored domains. To do so we develop a simple procedure which transforms any backbone convolutional neural network (CNN) into an architecture search space by replacing its operations with XD-operations. This space is then searched using a simple weight-sharing algorithm that needs only a small amount of tuning to find effective operations. As a simple first demonstration, we show that XD-operations yield models that are 15% more accurate than standard discrete search spaces on *permuted* CIFAR-10, highlighting the fragility of standard NAS operation spaces on new datasets, and thus the need for XD-operations. We then demonstrate the effectiveness of XD-operations in a series of applications showing that, starting from vanilla CNNs, they consistently outperform custom-designed operations. This includes:

- **Learning to solve partial differential equations (PDEs):** when substituted into a simple CNN backbone, XD-operations outperform convolutions and the dense prediction NAS method Auto-DeepLab [Liu et al., 2019a], and even achieve lower error than custom-designed, state-of-the-art operations (FNOs) across three problems with different dimensionalities (Burgers' equation, Darcy Flow, and Navier-Stokes). Our method also maintains consistent performance across different resolutions, a major stated advantage of FNOs over previous methods.
- **Protein folding:** on the task of predicting residue distances in a polypeptide chain—a key component of the protein folding problem—we substitute XD-operations into vanilla ResNets and achieve lower error than cyclically-dilated ResNets adapted specifically for this setting [Adhikari, 2019]. Furthermore, our ResNet-34 XD outperforms the reported error of the much deeper Dilated ResNet-258.
- **Music modeling:** on two next-note music prediction tasks, we show that substituting XD-operations into an undilated CNN outperforms temporal convolutional networks (TCNs), which are exponentially-dilated 1D CNNs that themselves outperform standard convolutional and recurrent networks [Bai et al., 2018].

DASH

While these results show the potential of a larger search space for improving performance on diverse tasks, they hide a significant practical issue: the compute and memory costs associated with training architectures based on XD-operations. Can we take a similar approach of substituting better operations in place of convolutions in widely used CNN backbones, but do so *efficiently*? We answer in the affirmative by introducing a novel NAS method called DASH (**D**iverse-task **A**rchitecture **S**earc**H**), in which we consider a search space of cross-scale dilated convolutions which are effective for multi-scale feature extraction [van den Oord et al., 2016, Yang et al., 2017] and context aggregation [Yu and Koltun, 2016, Chen et al., 2018c]. Our key difference from past search spaces is that we explicitly consider filters with *a wide range of kernel sizes and dilations*—while most NAS methods only handle kernels with maximum size 5 and dilation rate 2, our proposed operator space includes not only the conventional small kernels but also significantly larger ones with size 15 or dilation 127. This design choice is motivated by the fact that large kernels can capture input relations for dense prediction problems [Peng et al.,

2017], model long-range dependencies for sequence tasks [Bai et al., 2018, 2019], and resemble global-attention in Transformers [Liu et al., 2022]. Thus, DASH's cross-scale search space enables adaptation to diverse downstream tasks, unlike prior NAS work which targets image classification and assumes that small kernels are sufficient.

However, *efficiently* searching for an appropriate kernel configuration in this expansive cross-scale search space is non-trivial. Indeed, for existing NAS algorithms, the cost of exploring a combinatorially large set of operators is substantial. Even for weight-sharing methods that are known for efficiency, e.g. DARTS [Liu et al., 2019b], the computational complexity scales directly with the number of kernels considered and quadratically with the largest kernel size. To overcome this obstacle, DASH explores multi-scale convolutions via three techniques—the first two exploit mathematical properties of convolutions, and the last one takes advantage of fast matrix multiplication on GPUs. Specifically:

1. Using the **linearity** of convolutions, we mix several convolutions by computing one convolution equipped with a combined kernel rather than applying each filter separately and aggregating multiple outputs. While the number of convolution computations required by the naive aggregation of $|K|$ possible kernel sizes and $|D|$ possible dilations is $\mathcal{O}(|K||D|)$, our approach has $\mathcal{O}(1)$ complexity, independent of the search space size.
2. Using the **diagonalization** of convolutions, we relegate a major portion of the computation to element-wise multiplication in the Fourier domain, minimizing the effect of the largest kernel size on the complexity of our algorithm. For instance, a standard 1D convolution requires $\mathcal{O}(nk)$ operations to convolve a size-k kernel with a length-n input, but a Fourier convolution takes only $\mathcal{O}(n \log n)$, a critical improvement that makes searching over large kernels significantly easier.
3. Our final strategy is to use Kronecker products of undilated kernels and small sparse matrices to compute dilated kernels quickly on GPUs. This brings an additional two-fold speedup on top of the previous techniques.

Aside from these innovations, DASH employs the standard weight-sharing scheme of training a supernet, discretizing to obtain a model, and retraining the model for end tasks [Liu et al., 2019b]. We analyze the asymptotic complexity of the first two techniques and verify the practical utility when all three are combined together. In particular, DASH achieves a ten-fold speedup in total for differentiable NAS over the multi-scale search space. Moreover, we show that searching over large kernels is necessary to solve diverse problems and that each technique on its own cannot scale in this large-kernel setting.

In terms of accuracy performance, we evaluate DASH on ten datasets spanning multiple application areas such as PDE solving, protein folding, and disease prediction from NAS-Bench-360 [Tu et al., 2022], a new benchmark for diverse tasks. DASH yields models with better aggregate performance than those returned by leading AutoML methods as well as hand-designed task-specific architectures; it also beats all past automated approaches on seven of the ten problems and exceeds hand-designed models on seven, simultaneously maintaining strong efficiency relative to weight-sharing methods like DARTS. The empirical success of DASH implies that CNNs with appropriate kernels can be competitive baselines for problems where expert architectures are not available.

8.3 Discussion

Our first chapter on NAS attempts to build up an understanding of modern architecture search, starting with the weight-sharing technique. This has remained a challenging direction for theory, although Oymak et al. [2021] do show generalization guarantees for solving our feature map selection setup via continuous optimization on top of shared weights They also study the use of weight-sharing for activation function search, a setup later extended by Roberts et al. [2023] for the purpose of understanding the discretization step of weight-sharing-based NAS. Developing a complete understanding of architecture search is likely quite challenging, as the problem encompasses that of understanding deep learning itself. Nevertheless, it may be possible to build up a better empirical understanding of phenomena such as weight-sharing in order to make formal mathematical predictions of behavior, as has been done in other areas such as neural network optimization [Cohen et al., 2021].

Our last chapter focuses on applying NAS to diverse tasks beyond vision and language applications, a direction that has continued to see significant research investment, including through AutoML competitions [Roberts et al., 2022] as well as via the development of the NAS-Bench-360 benchmark that we use in our final evaluation [Tu et al., 2022]. At the same time, evidence continues to emerge that much of the NAS literature—which was developed on vision tasks, especially CIFAR-10—does not perform well beyond image data [White et al., 2022]. As a result, there remains significant scope for developing approaches that do yield high-performing architectures outside the vision modality. Here we believe our approach of focusing on operation spaces rather than network topology is promising because it can be used to design search spaces that contain truly novel operations while being easily incorporated into standard residual backbones.

Separately, the rise of large-scale pretrained models that can be fine-tuned on a wide variety of downstream tasks raises questions about the role of NAS in modern ML [Devlin et al., 2019, Dosovitskiy et al., 2021, Liu et al., 2021b]. For example, Lu et al. [2022] showed that Transformer models trained on text data can be effective feature extractors for out-of-modality datasets, suggesting their usefulness for diverse tasks. This finding was taken to its logical conclusion by Shen et al. [2023], who developed a way to fine-tune text Transformers such as BERT and image Transformers such as SWIN on all of the modalities in NAS-Bench-360, outperforming DASH in the process. While the two approaches can be complementary, and NAS retains some advantages such as inference-time performance, these results nevertheless show that the best performance on diverse tasks will likely be obtained by going beyond NAS and regular hyperparameter search to integrate cross-modality transfer into AutoML pipelines. Some promising directions here include learning combinable "tags" for representing domain-specific information and functions that can improve the performance of large language models (LLMs) on specialized tasks [Shen et al., 2024] or using the representation power of LLMs to generate embeddings of tasks [Achille et al., 2019] from their natural language descriptions and take advantage of the resulting task similarity information.

8.A Background

We briefly provide some background on modern approaches in architecture search. Since the possible number of neural network graphs is combinatorially large, significant effort is devoted to defining constraints on architecture search space that are thought to still preserve good settings while dramatically reducing its size. Often these constraints are motivated by common patterns found in existing expert-designed architectures. In this thesis we will at different times make use of two largely incompatible constraints: **micro cell-based search spaces** [Pham et al., 2018] and **network morphisms** [Jin et al., 2018].

8.A.1 Cell-based architecture search

Cell-based search spaces, which we use in Chapter 9, define the search domain using a small directed acyclic graph (DAG) or **cell** on ordered nodes N and edges E, with $|N|$ usually not much larger than ten. Each node $x^{(i)} \in N$ is a feature representation and each edge $(i, j) \in E$ is associated with an operation on the feature of node j passed to node i and aggregated with other inputs to form $x^{(j)}$, with the restriction that a given node j can only receive edges from prior nodes as input. Hence, the feature at node i is $x^{(i)} = \sum_{j<i} o^{(i,j)}(x^{(j)})$. These cells are then *stacked* one after another, with the output of one DAG feeding into the other; usually the number of cells used during search is smaller than when the final architecture is retrained. This stacking both reflects common architectural patterns such as ResNet blocks [He et al., 2016] and also significantly constrains the search domain while allowing very deep architectures. The resulting search spaces are specified by the number of nodes, the number of edges per node, and the set of (usually less than 10) operations O that can be applied at each edge. Thus $\mathcal{A} \subset \{0, 1\}^{|E|\times|O|}$ is the set of all valid architectures for encoded by edge and operation decisions.

8.A.2 Network morphisms

The network morphism approach takes the alternative approach of starting with an existing architecture and evolving it. For example, an approach we take in Chapter 10 is to start with a well-known CNN backbone such as VGG [Simonyan and Zisserman, 2015] or ResNet [He et al., 2016] and searching for convolutions to substitute in places of its convolutions. The resulting architectures are thus as large as expert-designed architectures but the search space itself is constrained to a simple product space over operations. At the same time, many highly successful architectures share similar network design patterns, e.g. Transformers have a roughly ResNet-like structure but with attention blocks instead of convolution [Vaswani et al., 2017].

8.A.3 Weight-sharing

Apart from imposing architectural constraints, the other significant research direction has been to find search heuristics to enable faster traversal of such large search spaces. Perhaps the most successful of these is *weight-sharing* [Pham et al., 2018], a technique that we wil make significant use of as well. Treating both the *shared weights* $\mathbf{w} \in \mathbb{R}^d$ and architecture decisions $a \in \mathcal{A}$ as parameters, weight-sharing methods train a single network or **supernet** subsuming all possible

functions within the search space. Weight-sharing methods have been mainly applied to cell-based search and are usually specified using one of two relaxations: continuous or stochastic.

Gradient-based weight-sharing methods apply continuous relaxations to the architecture space $\mathcal{A}$ in order to compute gradients in a continuous space Θ. Methods like DARTS [Liu et al., 2019b] and its variants [Chen et al., 2019, Laube and Zell, 2019, Hundt et al., 2019, Liang et al., 2019, Noy et al., 2019] relax the search space by considering a *mixture* of operations per edge. For example, in Chapter 9 we will consider a relaxation where the architecture space $\mathcal{A} = \{0,1\}^{|E|\times|O|}$ is relaxed into $\Theta = [0,1]^{|E|\times|O|}$ with the constraint that $\sum_{o\in O} \boldsymbol{\theta}_{[i,j,o]} = 1$, i.e. the operation weights on each edge sum to 1. The feature at node i is then $x^{(i)} = \sum_{j<i}\sum_{o\in O} \boldsymbol{\theta}_{[i,j,o]} o(x^{(j)})$. To get a valid architecture $a \in \mathcal{A}$ from a mixture $\boldsymbol{\theta}$, rounding and pruning are typically employed after the search phase.

The alternative, *stochastic* approach, such as that used by GDAS [Dong and Yang, 2019], instead uses Θ-parameterized distributions $p_{\boldsymbol{\theta}}$ over $\mathcal{A}$ to sample architectures [Pham et al., 2018, Xie et al., 2019, Akimoto et al., 2019, Cai et al., 2019]; unbiased gradients w.r.t. $\boldsymbol{\theta} \in \Theta$ can be computed using Monte Carlo sampling. The goal of all these relaxations is to use simple gradient-based approaches to approximately optimize (9.1) over $a \in \mathcal{A}$ by optimizing (9.2) over $\boldsymbol{\theta} \in \Theta$ instead. However, both the relaxation and the optimizer critically affect the convergence speed and solution quality.

Chapter 9

Understanding weight-sharing

An important tool for automating machine learning, neural architecture search has seen great progress in recent years [Real et al., 2019, Cai et al., 2019]; in particular, *weight-sharing* [Pham et al., 2018] has led to fast algorithms with state-of-the-art results on canonical image classification and language modeling problems [Liu et al., 2019b, Li and Talwalkar, 2019]. At the same time, theoretical study of statistical and optimization questions in this area has been minimal. Motivated by the challenge of developing simple and efficient methods that achieve state-of-the-art performance, we study different optimization objectives in NAS.

9.1 Weight-sharing objectives

Formally, consider the standard supervised ML setup where we have a dataset T of labeled pairs (x, y) drawn from a distribution $\mathcal{D}$ over input/output spaces X and Y. The goal is to use T to search a function class H for $h_{\mathbf{w}} : X \mapsto Y$ parameterized by $\mathbf{w} \in \mathbb{R}^d$ that has low expected test loss $\ell(h_{\mathbf{w}}(x), y)$ when using x to predict the associated y on unseen samples drawn from D, as measured by some loss $\ell : Y \times Y \mapsto [0, \infty)$. A common way to do so is by approximate (regularized) empirical risk minimization (ERM), i.e. finding $\mathbf{w} \in \mathbb{R}^d$ with the smallest average loss over T, via some iterative method $\texttt{Alg}$, e.g. SGD. NAS is often viewed as hyperparameter optimization on top of $\texttt{Alg}$, with each architecture $a \in \mathcal{A}$ corresponding to a function class $H_a = \{h_{\mathbf{w},a} : X \mapsto Y, \mathbf{w} \in \mathbb{R}^d\}$ to be selected by using validation data $V \subset X \times Y$ to evaluate the predictor obtained by fixing a and doing approximate ERM over T:

$$\min_{a \in \mathcal{A}} \sum_{(x,y) \in V} \ell(h_{\mathbf{w}_a, a}(x), y) \qquad \text{s.t.} \qquad \mathbf{w}_a = \texttt{Alg}(T, a) \tag{9.1}$$

Since training individual sets of weights for any sizeable number of architectures is prohibitive, weight-sharing methods instead use a single set of shared weights to obtain validation signal about many architectures at once. In its most simple form, RS-WS [Li and Talwalkar, 2019], these weights are trained to minimize a *non-adaptive* objective, $\min_{\mathbf{w} \in \mathbb{R}^d} \mathbb{E}_a \sum_{(x,y) \in T} \ell(h_{\mathbf{w}_a, a}(x), y)$, where the expectation is over a fixed distribution over architectures $\mathcal{A}$. The final architecture a

[0]The work presented in this chapter first appeared in Li et al. [2021a] and Khodak et al. [2020].

is then chosen to maximize the outer (validation) objective in (9.1) subject to $\mathbf{w}_a = \mathbf{w}$. More frequently used is a *bilevel* objective over some continuous relaxation Θ of the architecture space $\mathcal{A}$, after which a valid architecture is obtained via a discretization step $\texttt{Map} : \Theta \mapsto \mathcal{A}$ [Pham et al., 2018, Liu et al., 2019b]:

$$\min_{\boldsymbol{\theta} \in \Theta} \quad \sum_{(x,y) \in V} \ell(h_{\mathbf{w},\theta}(x), y) \qquad \text{s.t.} \qquad \mathbf{w} \in \arg\min_{\mathbf{u} \in \mathbb{R}^d} \sum_{(x,y) \in T} \ell(h_{\mathbf{u},\theta}(x), y) \tag{9.2}$$

This objective is not significantly different from (9.2), since $\texttt{Alg}(T, a)$ approximately minimizes the empirical risk w.r.t. T; the difference is replacing discrete architectures with relaxed *architecture parameters* $\boldsymbol{\theta} \in \Theta$, w.r.t. which we can take derivatives of the outer objective. This allows (9.2) to be approximated via alternating gradient updates w.r.t. $\mathbf{w}$ and $\boldsymbol{\theta}$. As described in the previous chapter, relaxations can be *stochastic*, so that $\texttt{Map}(\boldsymbol{\theta})$ is a sample from a $\boldsymbol{\theta}$-parameterized distribution [Pham et al., 2018, Dong and Yang, 2019], or a *mixture*, in which case $\texttt{Map}(\boldsymbol{\theta})$ selects architectural decisions with the highest weight in a convex combination given by $\boldsymbol{\theta}$ [Liu et al., 2019b].

While weight-sharing significantly shortens search [Pham et al., 2018], it draws two main criticisms:

- Rank disorder: this describes when the rank of an architecture a according to the validation risk evaluated with fixed shared weights $\mathbf{w}$ is poorly correlated with the one using "standalone" weights $\mathbf{w}_a = \texttt{Alg}(T, a)$. This causes suboptimal architectures to be selected after shared weights search [Yu et al., 2020a, Zela et al., 2020a, Zhang et al., 2020, Pourchot et al., 2020].
- Poor performance: weight-sharing can converge to degenerate architectures [Zela et al., 2020a] and is outperformed by regular hyperparameter tuning on NAS-Bench-201 [Dong and Yang, 2020].

9.2 Optimization in NAS: A single-level study

To study weight-sharing from the optimization perspective, we focus on the single-level objective in Equation 9.1. Why are we able to apply weight-sharing to NAS? The key is that, unlike regular hyperparameters such as step-size, *architectural* hyperparameters directly affect the loss function without requiring a dependent change in the model weights $\mathbf{w}$. Thus we can distinguish architectures without retraining simply by changing architectural decisions. Besides enabling weight-sharing, this point reveals that the goal of NAS is perhaps better viewed as a regular learning problem over an extended class $H_{\mathcal{A}} = \bigcup_{a \in \mathcal{A}} H_a = \{h_{\mathbf{w},a} : X \mapsto Y, \mathbf{w} \in \mathbb{R}^d, a \in \mathcal{A}\}$ that subsumes the architectural decisions as parameters of a larger model class, an unrelaxed "supernet." The natural approach to solving this is by approximate empirical risk minimization, e.g. by approximating continuous objective below on the right using a gradient algorithm and passing the output $\boldsymbol{\theta}$ through $\texttt{Map}$ to obtain a valid architecture:

$$\underbrace{\min_{\mathbf{w} \in \mathbb{R}^d, a \in \mathcal{A}} \quad \sum_{(x,y) \in T} \ell(h_{\mathbf{w},a}(x), y)}_{\text{discrete (unrelaxed) supernet (NAS ERM)}} \qquad \underbrace{\min_{\mathbf{w} \in \mathbb{R}^d, \boldsymbol{\theta} \in \Theta} \quad \sum_{(x,y) \in T} \ell(h_{\mathbf{w},\boldsymbol{\theta}}(x), y)}_{\text{continuous relaxation (supernet ERM)}} \tag{9.3}$$

Several works have optimized this single-level objective as an alternative to bilevel (9.2) [Xie et al., 2019, Li and Talwalkar, 2019]. We argue for its use as the baseline object of study in NAS for three reasons:

1. As discussed above, it is the natural first approach to solving the statistical objective of NAS: finding a good predictor $h_{\mathbf{w},a} \in H_{\mathcal{A}}$ in the extended function class over architectures and weights.
2. The common alternating gradient approach to the bilevel problem (9.2) is in practice very similar to alternating block approaches to ERM (9.3); as we will see, there are established ways of analyzing such methods for the latter objective, while for the former convergence is known only under very strong assumptions such as uniqueness of the inner minimum [Franceschi et al., 2018].
3. While less frequently used in practice than bilevel, single-level optimization can be very effective: we use it to achieve new state-of-the-art results on NAS-Bench-201 [Dong and Yang, 2020].

Understanding NAS as single-level optimization—the usual deep learning setting—makes weight-sharing a natural, not surprising, approach. Furthermore, for methods—both single-level and bilevel—that adapt architecture parameters during search, it suggests that we need not worry about rank disorder as long as we can use optimization to find a single feasible point that generalizes well; we explicitly do *not* need a ranking. Non-adaptive methods such as RS-WS still do require rank correlation to select good architectures after search, but they are explicitly *not* changing $\boldsymbol{\theta}$ and so have no variant solving (9.3). The single-level formulation thus reduces search method design to well-studied questions of how to best regularize and optimize ERM. While there are many techniques for regularizing weight-sharing—including partial channels [Xu et al., 2020b] and validation Hessian penalization [Zela et al., 2020a]—we focus on the second question of optimization.

9.2.1 Geometry-aware gradient algorithms

We seek to minimize the (possibly regularized) empirical risk $f(\mathbf{w}, \boldsymbol{\theta}) = \frac{1}{|T|} \sum_{(x,y)\in T} \ell(h_{\mathbf{w},\boldsymbol{\theta}}(x), y)$ over shared-weights $\mathbf{w} \in \mathbb{R}^d$ and architecture parameters $\boldsymbol{\theta} \in \Theta$. Assuming we have noisy gradients of f w.r.t. $\mathbf{w}$ or $\boldsymbol{\theta}$ at any point $(\mathbf{w}, \boldsymbol{\theta}) \in \mathbb{R}^d \times \Theta$—i.e. $\tilde{\nabla}_{\mathbf{w}} f(\mathbf{w}, \boldsymbol{\theta})$ or $\tilde{\nabla}_{\boldsymbol{\theta}} f(\mathbf{w}, \boldsymbol{\theta})$ satisfying $\mathbb{E}\tilde{\nabla}_{\mathbf{w}} f(\mathbf{w}, \boldsymbol{\theta}) = \nabla_{\mathbf{w}} f(\mathbf{w}, \theta)$ or $\mathbb{E}\tilde{\nabla}_{\boldsymbol{\theta}} f(\mathbf{w}, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\mathbf{w}, \boldsymbol{\theta})$, respectively—our goal is a point where f, or at least its gradient, is small, while taking as few gradients as possible. Our main complication is that architecture parameters lie in a constrained, non-Euclidean domain Θ. Most search spaces $\mathcal{A}$ are product sets of categorical decisions—which operation $o \in O$ to use at edge $e \in E$—so the natural relaxation is a product of $|E|$ $|O|$-simplices. However, NAS methods often re-parameterize Θ to be unconstrained using a softmax and then SGD or Adam [Kingma and Ba, 2015]. Is there a better parameterization-algorithm co-design? We consider a *geometry-aware* approach that uses mirror descent to design NAS methods with better properties depending on the domain; a key desirable property is to return *sparse* architectural parameters to reduce loss from post-search discretization.

Background on (offline) mirror descent

Mirror descent has many formulations [Nemirovski and Yudin, 1983, Beck and Teboulle, 2003, Shalev-Shwartz, 2011]; the *proximal* starts by noting that, in the unconstrained case, an SGD update at $\boldsymbol{\theta} \in \Theta = \mathbb{R}^k$ using gradient estimate $\tilde{\nabla} f(\boldsymbol{\theta})$ with step-size $\eta > 0$ is equivalent to

$$\boldsymbol{\theta} - \eta \tilde{\nabla} f(\boldsymbol{\theta}) \quad = \quad \operatorname*{arg\,min}_{\mathbf{u} \in \mathbb{R}^k} \quad \eta \tilde{\nabla} f(\boldsymbol{\theta}) \cdot \mathbf{u} \quad + \quad \frac{1}{2} \|\mathbf{u} - \boldsymbol{\theta}\|_2^2 \tag{9.4}$$

Here the first term aligns the output with the gradient while the second (proximal) term regularizes for closeness to the previous point as measured by the Euclidean distance. While the SGD update has been found to work well for unconstrained high-dimensional optimization, e.g. deep nets, this choice of proximal regularization may be sub-optimal over a constrained space with sparse solutions. The canonical such setting is optimization over the unit simplex, i.e. when $\Theta = \{\boldsymbol{\theta} \in [0,1]^k : \|\boldsymbol{\theta}\|_1 = 1\}$. Replacing the ℓ_2-regularizer in Equation 9.4 by the relative entropy $\mathbf{u} \cdot (\log \mathbf{u} - \log \boldsymbol{\theta})$, i.e. the KL-divergence, yields the exponentiated gradient:

$$\boldsymbol{\theta} \odot \exp(-\eta \tilde{\nabla} f(\boldsymbol{\theta})) \quad \propto \quad \operatorname*{arg\,min}_{\mathbf{u} \in \Theta} \quad \eta \tilde{\nabla} f(\boldsymbol{\theta}) \cdot \mathbf{u} \quad + \quad \mathbf{u} \cdot (\log \mathbf{u} - \log \boldsymbol{\theta}) \tag{9.5}$$

Note that the full EG update is obtained by ℓ_1-normalizing the l.h.s. It is well-known that EG over the k-dimensional simplex requires only $\mathcal{O}(\log k)/\varepsilon^2$ iterations to achieve a function value ε-away from optimal [Beck and Teboulle, 2003, Theorem 5.1], compared to the $\mathcal{O}(k/\varepsilon^2)$ guarantee of gradient descent. This nearly dimension-independent iteration complexity is achieved by choosing a regularizer—the KL divergence—well-suited to the underlying geometry—the simplex. More generally, mirror descent is specified by a **distance-generating function (DGF)**[1] ϕ that is strongly-convex w.r.t. some norm. ϕ induces a *Bregman divergence* $\mathcal{B}_\phi(\mathbf{u}||\mathbf{v}) = \phi(\mathbf{u}) - \phi(\mathbf{v}) - \nabla\phi(\mathbf{v}) \cdot (\mathbf{u} - \mathbf{v})$ [Bregman, 1967], a notion of distance on Θ that acts as a regularizer in the mirror descent update:

$$\operatorname*{arg\,min}_{\mathbf{u} \in \Theta} \quad \eta \tilde{\nabla} f(\boldsymbol{\theta}) \cdot \mathbf{u} \quad + \quad \mathcal{B}_\phi(\mathbf{u}||\boldsymbol{\theta}) \tag{9.6}$$

For example, to recover SGD (9.4) set $\phi(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}\|_2^2$, which is strongly-convex w.r.t. the Euclidean norm, while EG (9.5) is recovered by setting $\phi(\mathbf{u}) = \mathbf{u} \cdot \log \mathbf{u}$, which is strongly-convex w.r.t. the ℓ_1-norm.

Block-stochastic mirror descent

In the previous section we saw how mirror descent can perform better over certain geometries such as the simplex. However, in weight-sharing we are interested in optimizing over a hybrid geometry containing both the shared weights in an unconstrained Euclidean space and the architecture parameters in a non-Euclidean domain. Thus we focus on optimization over two blocks: shared weights $\mathbf{w} \in \mathbb{R}^d$ and architecture parameters $\theta \in \Theta$, the latter associated with a DGF ϕ that is strongly-convex w.r.t. some norm $\|\cdot\|$. In NAS a common approach is to perform alternating gradient steps on each domain; for example, both ENAS [Pham et al., 2018] and first-order

[1] In Parts I and II we used the term "regularizer" instead of DGF, as is often done in the OCO community.

Algorithm 24: Block-stochastic mirror descent optimization of $f : \mathbb{R}^d \times \Theta \mapsto \mathbb{R}$.

Input: initialization $(\mathbf{w}_1, \boldsymbol{\theta}_1) \in \mathbb{R}^d \times \Theta$, step-size $\eta > 0$, number of iterations $T \geqslant 1$, strongly-convex distance-generating function $\phi : \Theta \mapsto \mathbb{R}$

for iteration $t = 1, \ldots, T$ **do**

  sample $b_t \sim \mathrm{Unif}\{1, 2\}$ `// randomly select update block`

  if block $b_t = 1$ **then**

    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \tilde{\nabla}_{\mathbf{w}} f(\mathbf{w}_t, \boldsymbol{\theta}_t)$ `// SGD update to shared weights`

    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$ `// no update to architecture params`

  else

    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$ `// no update to shared weights`

    $\boldsymbol{\theta}_{t+1} \leftarrow \arg\min_{\mathbf{u} \in \Theta} \eta \tilde{\nabla}_{\boldsymbol{\theta}} f(\mathbf{w}_t, \boldsymbol{\theta}_t) \cdot \mathbf{u} + \mathcal{B}_\phi(\mathbf{u} || \boldsymbol{\theta}_t)$ `// update arch. params`

Output: $(\mathbf{w}_r, \boldsymbol{\theta}_r)$ for $r \sim \mathrm{Unif}\{1, \ldots, T\}$ `// return random iterate`

DARTS [Liu et al., 2019b] alternate between SGD on the shared weights and Adam on architecture parameters. This approach is encapsulated in the block-stochastic algorithm described in Algorithm 24, which at each step chooses one block at random to update using mirror descent (recall that SGD is a variant) and after T steps returns a random iterate. Algorithm 24 generalizes the single-level variant of both ENAS and first-order DARTS if SGD is used to update θ instead of Adam, with some mild caveats: in practice blocks are picked cyclically and the algorithm returns the last iterate, not a a random one.

To analyze the convergence of Algorithm 24 we first state some regularity assumptions:

Assumption 9.2.1. Suppose ϕ is strongly-convex w.r.t. some norm $\|\cdot\|$ on a convex set Θ and the objective function $f : \mathbb{R}^d \times \Theta \mapsto [0, \infty)$ satisfies the following:

1. γ**-relatively-weak-convexity:** $f(\mathbf{w}, \boldsymbol{\theta}) + \gamma\phi(\boldsymbol{\theta})$ is convex on $\mathbb{R}^d \times \Theta$ for some $\gamma > 0$
2. **gradient bound:** $\exists\, G_{\mathbf{w}}$ and $G_{\boldsymbol{\theta}} \geqslant 0$ s.t. $\mathbb{E}\|\tilde{\nabla}_{\mathbf{w}} f(\mathbf{w}, \boldsymbol{\theta})\|_2^2 \leqslant G_{\mathbf{w}}^2$ and $\mathbb{E}\|\tilde{\nabla}_{\boldsymbol{\theta}} f(\mathbf{w}, \boldsymbol{\theta})\|_*^2 \leqslant G_{\boldsymbol{\theta}}^2$

The second assumption is a standard bound on the gradient norm while the first is a generalization of smoothness that allows all smooth and some nonsmooth nonconvex functions [Zhang and He, 2018]. Using these assumptions, our aim will be to show (first-order) ε-stationary-point convergence of Algorithm 24, a standard metric indicating that it has reached a point with no feasible descent direction, up to error ε; for example, in the unconstrained Euclidean case an ε-stationary-point is simply one where the gradient has squared-norm $\leqslant \varepsilon$. The number of steps required to obtain such a point thus measures how fast a first-order method terminates. Stationarity is also significant as a necessary condition for optimality.

In our case Θ may be constrained and so the gradient may never be small, thus necessitating a measure other than gradient norm. We use *Bregman stationarity* [Zhang and He, 2018, Equation 2.11], which measures stationary at a point $(\mathbf{w}, \boldsymbol{\theta})$ using the Bregman divergence between the point and its proximal map $\mathrm{prox}_\lambda(\mathbf{w}, \boldsymbol{\theta}) = \arg\min_{\mathbf{u} \in \mathbb{R}^d \times \Theta} \lambda f(\mathbf{u}) + \mathcal{B}_{\ell_2, \phi}(\mathbf{u} || \mathbf{w}, \boldsymbol{\theta})$ for some $\lambda > 0$:

$$\Delta_\lambda(\mathbf{w}, \boldsymbol{\theta}) = \frac{\mathcal{B}_{\ell_2,\phi}(\mathbf{w}, \boldsymbol{\theta} || \mathrm{prox}_\lambda(\mathbf{w}, \boldsymbol{\theta})) + \mathcal{B}_{\ell_2,\phi}(\mathrm{prox}_\lambda(\mathbf{w}, \boldsymbol{\theta}) || \mathbf{w}, \boldsymbol{\theta})}{\lambda^2} \tag{9.7}$$

Here $\lambda = \frac{1}{2\gamma}$ and the Bregman divergence $\mathcal{B}_{\ell_2,\phi}$ is that of the DGF $\frac{1}{2}\|\mathbf{w}\|_2^2 + \phi(\boldsymbol{\theta})$ that encodes the geometry of the joint optimization domain over $\mathbf{w} \in \mathbb{R}^d$ and $\boldsymbol{\theta}$; note that the dependence of the stationarity measure on γ is standard [Dang and Lan, 2015, Zhang and He, 2018].

To understand why reaching a point $(\mathbf{w}, \boldsymbol{\theta})$ with small Bregman stationarity is a reasonable goal, note that the proximal operator prox_λ has the property that its fixed points, i.e. those satisfying $(\mathbf{w}, \boldsymbol{\theta}) = \mathrm{prox}_\lambda(\mathbf{w}, \boldsymbol{\theta})$, correspond to points where f has no feasible descent direction. Thus measuring how close $(\mathbf{w}, \boldsymbol{\theta})$ is to being a fixed point of prox_λ—as is done using the Bregman divergence in (9.7)—is a good measure of how far away the point is from being a stationary point of f. Finally, note that if f is smooth, ϕ is Euclidean, and Θ is unconstrained—i.e. if we are running SGD over architecture parameters as well—then $\Delta_{\frac{1}{2\gamma}} \leqslant \varepsilon$ implies a $\mathcal{O}(\varepsilon)$-bound on the squared gradient norm, recovering the standard definition of ε-stationarity. More intuition on proximal operators can be found in Parikh and Boyd [2013, Section 1.2], while further details on Bregman stationarity and how it relates to other notions of convergence can be found in Zhang and He [2018, Section 2.3].

The following result shows that Algorithm 24 needs polynomially many iterations to finds a point $(\mathbf{w}, \boldsymbol{\theta})$ with ε-small Bregman stationarity in-expectation:

Theorem 9.2.1. Let $F = f(\mathbf{w}_1, \boldsymbol{\theta}_1)$ be the value of f at initialization. Under Assumption 9.2.1, if we run Algorithm 24 for $T = \frac{16\gamma F}{\varepsilon^2}(G_{\mathbf{w}}^2 + G_{\boldsymbol{\theta}}^2)$ iterations with step-size $\eta = \sqrt{\frac{4F}{\gamma(G_{\mathbf{w}}^2+G_{\boldsymbol{\theta}}^2)T}}$ then $\mathbb{E}\Delta_{\frac{1}{2\gamma}}(\mathbf{w}_r, \boldsymbol{\theta}_r) \leqslant \varepsilon$. Here the expectation is over the randomness of the algorithm and gradients.

The proof in Appendix 9.A.1 follows from single-block analysis [Zhang and He, 2018, Theorem 3.1] and in fact holds for the general case of any number of blocks associated to any set of strongly-convex DGFs. Although there are prior results for the multi-block case [Dang and Lan, 2015], they do not hold for nonsmooth Bregman divergences such as the KL divergence needed for exponentiated gradient.

Thus Algorithm 24 returns an ε-stationary-point given $T = \mathcal{O}(G_{\mathbf{w}}^2 + G_{\boldsymbol{\theta}}^2)/\varepsilon^2$ iterations, where $G_{\mathbf{w}}^2$ bounds the squared ℓ_2-norm of the shared-weights gradient $\tilde{\nabla}_{\mathbf{w}}$ and $G_{\boldsymbol{\theta}}^2$ bounds the squared magnitude of the architecture gradient $\tilde{\nabla}_{\boldsymbol{\theta}}$, as measured by the *dual norm* $\|\cdot\|_*$ of $\|\cdot\|$. Only the last term $G_{\boldsymbol{\theta}}$ is affected by our choice of DGF ϕ. The DGF of SGD is strongly-convex w.r.t. the ℓ_2-norm, which is its own dual, so $G_{\mathbf{w}}^2$ is defined via ℓ_2. However, for EG the DGF $\phi(\mathbf{u}) = \mathbf{u} \cdot \log \mathbf{u}$ is strongly-convex w.r.t. the ℓ_1-norm, whose dual is ℓ_∞. Since the ℓ_2-norm of a k-dimensional vector can be $\sqrt{k}$ times its ℓ_∞-norm, picking this DGF can lead to better bound on G_θ and thus on the number of iterations.

GAEA: A geometry-aware exponentiated algorithm

Equipped with these single-level guarantees, we turn to designing methods that can in-principle be applied to both the single-level and bilevel objectives, seeking parameterizations and algorithms that converge quickly and encourage favorable properties; in particular, we focus on returning architecture parameters that are *sparse* to reduce loss due to post-search discretization. EG is often considered to converge quickly to sparse solutions over the simplex [Bradley and Bagnell, 2008, Bubeck, 2019], which makes it a natural choice for the architecture update. We thus propose **GAEA**, a **Geometry-Aware Exponentiated Algorithm** in which operation

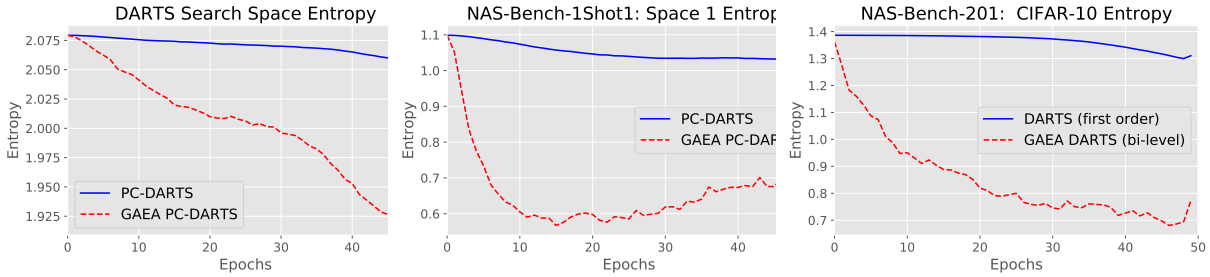

Figure 9.1: Evolution over search epochs of the average entropy of the operation weights when run on the DARTS search space (left), NAS-Bench-1Shot1 Search Space 1 (middle), and NASBench-201 on CIFAR-10 (right). GAEA reduces entropy much more quickly, allowing it to quickly obtain sparse architecture weights. This leads to both faster convergence to a single architecture and a lower loss when pruning at the end of search.

weights on each edge are constrained to the simplex and trained using EG; as in DARTS, the shared weights $\mathbf{w}$ are trained using SGD. GAEA can be used as a simple, principled modification to the many NAS methods that treat architecture parameters $\boldsymbol{\theta} \in \Theta = \mathbb{R}^{|E| \times |O|}$ as real-valued "logits" to be passed through a softmax to obtain mixture weights or probabilities for simplices over the operations O. Such methods include DARTS, PC-DARTS [Xu et al., 2020b], and GDAS [Dong and Yang, 2019]. To apply GAEA, first re-parameterize Θ to be the product set of $|E|$ simplices, each associated to an edge $(i,j) \in E$; thus $\boldsymbol{\theta}_{[i,j,o]}$ corresponds directly to the weight or probability of operation $o \in O$ for edge (i,j), not a logit. Then, given a stochastic gradient $\tilde{\nabla}_{\boldsymbol{\theta}} f(\mathbf{w}_t, \boldsymbol{\theta}_t)$ and step-size $\eta > 0$, replace the architecture update by EG:

$$\begin{aligned} \tilde{\boldsymbol{\theta}}_{t+1} &\leftarrow \boldsymbol{\theta}_t \odot \exp\left(-\eta \tilde{\nabla}_{\boldsymbol{\theta}} f(\mathbf{w}_t, \boldsymbol{\theta}_t)\right) && \text{(multiplicative update)} \\ \boldsymbol{\theta}_{t+1[i,j,o]} &\leftarrow \frac{\tilde{\boldsymbol{\theta}}_{t+1[i,j,o]}}{\sum_{o' \in O} \tilde{\boldsymbol{\theta}}_{t+1[i,j,o']}} \;\; \forall\, o \in O,\; \forall\, (i,j) \in E && \text{(simplex projection)} \end{aligned} \tag{9.8}$$

These two simple modifications, *re-parameterization* and *exponentiation*, suffice to obtain state-of-the-art results on several NAS benchmarks, as shown in Section 9.2.2. Note that to obtain a bilevel algorithm we simply replace the gradient w.r.t. $\boldsymbol{\theta}$ of the training loss with that of the validation loss.

GAEA is equivalent to Algorithm 24 with $\phi(\boldsymbol{\theta}) = \sum_{(i,j) \in E} \sum_{o \in O} \boldsymbol{\theta}_{[i,j,o]} \log \boldsymbol{\theta}_{[i,j,o]}$, which is strongly-convex w.r.t. $\|\cdot\|_1 / \sqrt{|E|}$ over the product of $|E|$ $|O|$-simplices. The dual is $\sqrt{|E|}\|\cdot\|_\infty$, so if $G_{\mathbf{w}}$ bounds the shared-weights gradient and we have an entry-wise bound on the architecture gradient then GAEA reach ε-stationarity in $\mathcal{O}(G_{\mathbf{w}}^2 + |E|)/\varepsilon^2$ iterations. This can be up to a factor $|O|$ improvement over SGD, either over the simplex or the logit space. In addition, GAEA encourages sparsity in the architecture weights by using a multiplicative update over simplices and not an additive update over $\mathbb{R}^{|E| \times |O|}$. Obtaining sparse architecture parameters is critical for good performance, both for the mixture relaxation, where it alleviates the effect of discretization on the validation loss, and for the stochastic relaxation, where it reduces noise when sampling architectures.

9.2.2 Empirical results using GAEA

We evaluate GAEA on three different computer vision benchmarks: the large and heavily studied search space from DARTS [Liu et al., 2019b] and two smaller oracle evaluation benchmarks, NAS-Bench-1Shot1 [Zela et al., 2020b], and NAS-Bench-201 [Dong and Yang, 2020]. NAS-Bench-1Shot1 differs from the others by applying operations per node instead of per edge, while NAS-Bench-201 differs by not requiring edge-pruning. Since GAEA can modify a variety of methods, e.g. DARTS, PC-DARTS [Xu et al., 2020b], and GDAS [Dong and Yang, 2019], on each benchmark we start by evaluating the GAEA variant of the current best method on that benchmark. We show that despite the diversity of search spaces, GAEA improves upon this state-of-the-art across all three. Note that we use the same step-size for GAEA variants of DARTS/PC-DARTS and do not require weight decay on architecture parameters. We defer experimental details and hyperparameter settings to Appendix 9.B.1 and release all code, hyperparameters, and random seeds for reproducibility.

Convergence and sparsity of GAEA

We first examine the impact of GAEA on convergence and sparsity. Figure 9.1 shows the entropy of the operation weights averaged across nodes for a GAEA-variant and its base method across the three benchmarks, demonstrating that it decreases much faster for GAEA-modified approaches. This validates our expectation that GAEA encourages sparse architecture parameters, which should alleviate the mismatch between the continuously relaxed architecture parameters and the discrete architecture returned. Indeed, we find that post-search discretization on the DARTS search space causes the validation accuracy of the PC-DARTS supernet to drop from 72.17% to 15.27%, while for GAEA PC-DARTS the drop is only 75.07% to 33.23%; note that this is shared-weights accuracy, obtained *without* retraining the final network. The numbers demonstrate that GAEA both (1) achieves better supernet optimization of the weight-sharing objective and (2) suffers less due to discretization.

GAEA on the DARTS search space

Here we evaluate GAEA on the task of designing CNN cells for CIFAR-10 [Krizhevksy, 2009] and ImageNet [Russakovsky et al., 2015] by using it to modify PC-DARTS [Xu et al., 2020b], the current state-of-the-art method. We follow the same three stage process used by both DARTS and RS-WS for search and evaluation. Table 9.1 displays results on both datasets and demonstrates that GAEA's parameterization and optimization scheme improves upon PC-DARTS. In fact, GAEA PC-DARTS outperforms all search methods except ProxylessNAS, which uses 1.5 times as many parameters on a different search space. Thus we improve the state-of-the-art on the DARTS search space. To meet a higher bar for reproducibility on CIFAR-10, in Appendix 9.B.1 we report "broad reproducibility" [Li and Talwalkar, 2019] by repeating our pipeline with new seeds. While GAEA PC-DARTS consistently finds good networks when selecting the best of four independent trials, multiple trials are required due to sensitivity to initialization, as is true for many approaches [Liu et al., 2019b, Xu et al., 2020b].

On ImageNet, we follow Xu et al. [2020b] by using subsamples containing 10% and 2.5% of the training images from ILSVRC-2012 [Russakovsky et al., 2015] as training and validation

Table 9.1: Comparison with SOTA NAS methods on the DARTS search space, plus three results on different search spaces with a similar number of parameters reported at the top for comparison. All evaluations and reported performances of models found on the DARTS search space use similar training routines; this includes auxiliary towers and cutout but no other modifications, e.g. label smoothing [Müller et al., 2019], AutoAugment [Cubuk et al., 2019], Swish [Ramachandran et al., 2017], Squeeze & Excite [Hu et al., 2018], etc. The specific training procedure we use is that of PC-DARTS, which differs slightly from the DARTS routine by a small change to the drop-path probability; PDARTS tunes both this and batch-size. Our results are averaged over 10 random seeds. Search cost is hardware-dependent; we used Tesla V100 GPUs. For more details see Tables 9.4 & 9.5.

	CIFAR-10 Error		**Search Cost**	**ImageNet Error**		**Search Cost**	
Search Method (source)	Best	Average	(GPU Days)	top-1	top-5	(GPU Days)	method
NASNet-A* [Zoph et al., 2018]	-	2.65	2000	26.0	8.4	1800	RL
AmoebaNet-B* [Real et al., 2019]	-	2.55 ± 0.05	3150	24.3	7.6	3150	evolution
ProxylessNAS* [Cai et al., 2019]	2.08	-	4	24.9	7.5	8.3	gradient (WS)
ENAS [Pham et al., 2018]	2.89	-	0.5	-	-	-	RL (WS)
RS-WS† [Li and Talwalkar, 2019]	2.71	2.85 ± 0.08	0.7	-	-	-	random (WS)
ASNG [Akimoto et al., 2019]	-	2.83 ± 0.14	0.1	-	-	-	gradient (WS)
SNAS [Xie et al., 2019]	-	2.85 ± 0.02	1.5	27.3	9.2	1.5	gradient (WS)
DARTS (1st)† [Liu et al., 2019b]	-	3.00 ± 0.14	0.4	-	-	-	gradient (WS)
DARTS (2nd)† [Liu et al., 2019b]	-	2.76 ± 0.09	1	26.7	8.7	4.0	gradient (WS)
PDARTS [Chen et al., 2019]	2.50	-	0.3	24.4	7.4	0.3	gradient (WS)
PC-DARTS† [Xu et al., 2020b]	-	2.57 ± 0.07	0.1	24.2	7.3	3.8	gradient (WS)
GAEA PC-DARTS† (ours)	2.39	2.50 ± 0.06	0.1	24.0	7.3	3.8	gradient (WS)
PC-DARTS† [Xu et al., 2020b]	(search on CIFAR-10, train on ImageNet)			25.1	7.8	0.1	gradient (WS)
GAEA PC-DARTS† (ours)	(search on CIFAR-10, train on ImageNet)			24.3	7.3	0.1	gradient (WS)

* Search space/backbone differ from the DARTS setting; we show results for networks with a comparable number of parameters.
† For fair comparison to other work, we show the search cost for training the shared-weights network with a single initialization.

sets, respectively. We fix architecture parameters for the first 35 epochs, then run GAEA PC-DARTS with step-size 0.1. All other hyperparameters match those of Xu et al. [2020b]. Table 9.1 shows the final performance of both the architecture found by GAEA PC-DARTS on CIFAR-10 and the one found directly on ImageNet when trained from scratch for 250 epochs using the same settings as Xu et al. [2020b]. GAEA PC-DARTS achieves a top-1 test error of 24.0%, which is state-of-the-art performance in the mobile setting when excluding additional training modifications, e.g. those in the caption. Additionally, the architecture found by GAEA PC-DARTS for CIFAR-10 and transferred achieves a test error of 24.2%, comparable to the 24.2% error of the one found by PC-DARTS directly on ImageNet. The top architectures found by GAEA PC-DARTS are depicted in Figure 9.5 in Appendix 9.B.1.

GAEA on NAS-Bench-1Shot1

NAS-Bench-1Shot1 [Zela et al., 2020b] is a subset of NAS-Bench-101 [Ying et al., 2019] that allows benchmarking weight-sharing methods on three search spaces over CIFAR-10 that differ in the number of nodes considered and the number of input edges per node. Of the weight-sharing methods benchmarked by Zela et al. [2020b], we found that PC-DARTS achieves the best

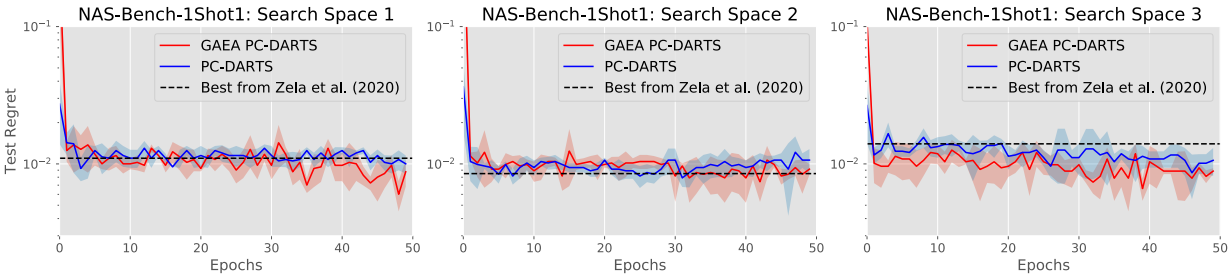

Figure 9.2: Online comparison of PC-DARTS and GAEA PC-DARTS in terms of the test regret at each epoch of shared-weights training on NAS-Bench-1Shot1, i.e. the difference between the ground truth test error of the proposed architecture and that of the best architecture in the search space. The dark lines indicate the mean of four random trials and the light colored bands $\pm$ one standard deviation. The dashed line is the final regret of the best weight-sharing method according to Zela et al. [2020b]; note that in our reproduction PC-DARTS performed better than their evaluation on spaces 1 and 3.

performance on 2 of 3 search spaces, so we again evaluate GAEA PC-DARTS here. Figure 9.2 shows that GAEA PC-DARTS consistently finds better architectures on average than PC-DARTS and thus exceeds the performance of the best method from Zela et al. [2020b] on 2 of 3 search spaces. We hypothesize that the benefits of GAEA are limited here due to the near-saturation of NAS methods. In particular, existing methods obtain within 1% test error of the top network in each space, while the latters' test errors when evaluated with different initializations are 0.37%, 0.23% and 0.19%, respectively.

GAEA on NAS-Bench-201

NAS-Bench-201 has one search space on three datasets—CIFAR-10, CIFAR-100, and ImageNet-16-120—that includes 4-node architectures with an operation from $O = \{$none, skip connect, 1x1 convolution, 3x3 convolution, 3x3 avg pool$\}$ on each edge, yielding 15625 possible networks. Dong and Yang [2020] report results for several algorithms in the *transfer NAS* setting, where search is conducted on CIFAR-10 and the resulting networks are trained on a possibly different target dataset. Table 9.2 reports a subset of these results alongside evaluations of our implementation of several existing and GAEA-modified NAS methods in both the transfer and direct setting. Both the results from Dong and Yang [2020] and our reproductions show that GDAS is the best previous weight-sharing method; we evaluate GAEA GDAS and find that it achieves better results on CIFAR-100 and similar results on the other two datasets.

Since we are interested in improving upon not only GAEA GDAS but also upon traditional hyperparameter optimization methods, we also investigate the performance of GAEA applied to first-order DARTS. We evaluate GAEA DARTS with both single-level (ERM) and bilevel optimization; recall that in the latter case we optimize architecture parameters w.r.t. the validation loss and the shared weights w.r.t. the training loss, whereas in ERM there is no data split. GAEA DARTS (ERM) achieves state-of-the-art performance on all three datasets in both the transfer and direct setting, exceeding the test accuracy of both weight-sharing and traditional hyperparameter tuning by a wide margin. GAEA DARTS (bilevel) performs worse but still exceeds all

Table 9.2: NAS-Bench-201 separated into traditional hyperparameter optimization algorithms with search run on CIFAR-10 (top block), weight-sharing methods with search run on CIFAR-10 (middle block), and weight-sharing methods run directly on the dataset used for training (bottom block). The use of transfer NAS follows the evaluations conducted by Dong and Yang [2020]; unless otherwise stated all non-GAEA results are from their paper. The best results in the transfer and direct settings on each dataset are **bolded**.

		Search* (seconds)	CIFAR-10 (test)	CIFAR-100 (test)	ImageNet-16-120 (test)
Regular HO, search on CIFAR-10	REA	N/A	93.92 ± 0.30	71.84 ± 0.99	45.54 ± 1.03
	RS	N/A	93.70 ± 0.36	71.04 ± 1.08	44.57 ± 1.25
	REINFORCE	N/A	93.85 ± 0.37	71.71 ± 1.09	45.25 ± 1.18
	BOHB	N/A	93.61 ± 0.52	70.85 ± 1.28	44.42 ± 1.49
Weight sharing, search on CIFAR-10	RSPS	7587	87.66 ± 1.69	58.33 ± 4.34	31.14 ± 3.88
	DARTS (bilevel)	35781	54.30 ± 0.00	15.61 ± 0.00	16.32 ± 0.00
	SETN	34139	87.64 ± 0.00	59.05 ± 0.24	32.52 ± 0.21
	GDAS	31609	93.61 ± 0.09	70.70 ± 0.30	41.71 ± 0.98
	DARTS‡ (bilevel)	10683†	54.30 ± 0.00	15.32 ± 0.00	16.38 ± 0.00
	GAEA DARTS (bilevel)	7930†	91.63 ± 2.57	68.39 ± 4.47	41.59 ± 4.20
	DARTS‡ (ERM)	18112†	84.39 ± 3.82	54.81 ± 7.08	31.82 ± 4.78
	GAEA DARTS (ERM)	9061†	$\mathbf{94.10 \pm 0.29}$	$\mathbf{72.60 \pm 0.89}$	$\mathbf{45.81 \pm 0.51}$
Weight sharing, direct search	GDAS‡	27923†	93.52 ± 0.15	67.52 ± 0.15	40.91 ± 0.12
	GAEA GDAS	16754†	93.55 ± 0.13	70.47 ± 0.47	40.91 ± 0.12
	DARTS‡ (bilevel)	10683†	54.30 ± 0.00	15.32 ± 0.00	28.96 ± 10.22
	GAEA DARTS (bilevel)	7930†	91.63 ± 2.57	71.87 ± 0.57	45.69 ± 0.56
	DARTS‡ (ERM)	18112†	84.39 ± 3.82	51.26 ± 6.14	31.35 ± 7.46
	GAEA DARTS (ERM)	9061†	$\mathbf{94.10 \pm 0.29}$	$\mathbf{73.43 \pm 0.13}$	$\mathbf{46.36 \pm 0.00}$
	ResNet	N/A	93.97	70.86	43.63
	Optimal	N/A	94.37	73.51	47.31

* Search cost reported for running the search algorithm on CIFAR-10.
† Search cost measured on NVIDIA P100 GPUs.
‡ Our reproduction or implementation of a non-GAEA method.

other methods on CIFAR-100 and ImageNet-16-120 in the direct search setting. The result thus also confirms the relevance of studying the single-level case to understand NAS; notably, the DARTS (ERM) baseline also improves substantially upon the DARTS (bilevel) baseline.

9.2.3 Conclusion

In this section we studied NAS as a single-level optimization problem, arguing that the design of good NAS algorithms is largely a matter of successfully optimizing and regularizing the supernet. In support of this, we develop GAEA, a simple modification of gradient-based NAS that attains state-of-the-art performance on several computer vision benchmarks while enjoying favorable speed and sparsity properties. The next section complicates this view, showing that from a statistical perspective there may be some evidence for the bilevel approach.

9.3 The benefits of bilevel optimization: Selecting feature maps using weight-sharing

While NAS is at least as hard a problem as deep learning, we might still hope to gain insight from understanding certain sub-problems, sub-routines, or simple settings. In this section focusing on the bilevel optimization problem, we take the latter approach and propose *feature map selection*—picking the best fixed data representations to use in a linear model—as a way to study NAS and weight-sharing. In this setting, we show empirically that weight-sharing provides a signal in the form of correlation between the performances of shared and standalone weights for individual configurations, just as in NAS. Using this, we design a simple method that outperforms hyperparameter tuning baselines on two tasks. Finally, we show how the simple setting also provides an explicit case where a bilevel objective improves over single-level optimization in terms of sample complexity and discuss insights and limitations of this analysis.

9.3.1 Random search with weight-sharing

A starting point for solving the bilevel optimization problem is random search [Bergstra and Bengio, 2012]: randomly sample a configuration $c \in \mathcal{C}$, train the inner loop problem to completion, and repeat. More sophisticated methods adapt the distribution [Hutter et al., 2011] or allocate fewer resources to less-promising configurations [Li et al., 2018a]. A major drawback of these methods is the need to train configurations separately, which means only a limited part of the search space can be explored under resource constraints. Hence, Pham et al. [2018] proposed simultaneously exploring both the weight-space $\mathcal{W}$ and the configuration space $\mathcal{C}$ by partially training the weights w with minibatch gradient steps w.r.t. architectures c sampled from a parameterized distribution. These weights are called *shared* because they are trained to optimize a distribution of architectures over $\mathcal{C}$.

Despite the noise due to updating w.r.t. different architectures, weight-sharing has become an incredibly successful tool. A simple example that illustrates its surprising power is random search with weight-sharing (RS-WS) [Bender et al., 2018, Li and Talwalkar, 2019], where weights are trained by taking minibatch gradient steps w.r.t. *uniformly* sampled architectures. The bilevel object solved by RS-WS is thus

$$\min_{c \in \mathcal{C}} \ell_V(\mathbf{w}^*, c) \quad \text{s.t.} \quad \mathbf{w}^* \in \arg\min_{\mathbf{u} \in \mathcal{W}} \mathop{\mathbb{E}}_{c \sim \text{Unif}(\mathcal{C})} \mathcal{L}_T(\mathbf{u}, c) \tag{9.9}$$

i.e. shared-weights are trained to optimize the expected loss of a uniformly sampled architecture. This is followed by evaluations of different configurations $c \in \mathcal{C}$ using the resulting shared weights $\mathbf{w}^*$. In the case of the two benchmarks studied by Li and Talwalkar [2019], highly performant architectures according to the shared-weights also had high ground truth performance. In Section 9.3.2 we observe analogous behavior for the feature map selection problem.

9.3.2 Feature map selection: A simple setting for understanding NAS

In this section, we introduce feature map selection and show how it can be viewed as a simple type of NAS with weight-sharing. Empirically, weight-sharing outperforms random search

Algorithm 25: Feature map selection using successive halving with weight-sharing.

Input: training set T, validation set V, convex loss ℓ, set of feature map ϕ_c configurations $\mathcal{C}$, regularization parameter $\lambda > 0$

for *round* $t = 1, \ldots, \log_2 |\mathcal{C}|$ **do**

  for *datapoint* $(x, y) \in T \cup V$ **do**

    assign $c_x \sim \mathrm{Unif}(\mathcal{C})$ i.i.d.

  $\mathbf{w}_t^* \leftarrow \arg\min_{\mathbf{w} \in \mathbb{R}^d} \ \lambda \|\mathbf{w}\|_2^2 + \sum_{(x,y) \in T} \ell(\langle \mathbf{w}, \phi_{c_x}(x) \rangle, y)$

  for *configuration* $c \in \mathcal{C}$ **do**

    $V_c \leftarrow \{(x, y) \in V : c_x = c\}$

    $s_c \leftarrow \frac{1}{|V_c|} \sum_{(x,y) \in V_c} \ell(\langle \mathbf{w}_t^*, \phi_{c_x}(x) \rangle, y)$

  $\mathcal{C} = \{c \in \mathcal{C} : s_c \leqslant \mathrm{Median}(\{s_c : c \in \mathcal{C}\})\}$

Result: Singleton set of configurations $\mathcal{C}$.

and Hyperband [Li et al., 2018a] both when selecting random Fourier maps on CIFAR-10 and configuring Bag-of-n-Grams representations on IMDb, motivating further analysis.

In feature map selection each configuration $c \in \mathcal{C}$ corresponds to a feature map $\boldsymbol{\phi}_c : \mathcal{X} \mapsto \mathbb{R}^d$ of the input to be passed to a linear classifier in $\mathcal{W} = \mathbb{R}^d$; the hypothesis space is then $H(\mathbb{R}^d, \mathcal{C}) = \{\langle \mathbf{w}, \boldsymbol{\phi}_c(\cdot) \rangle : \mathbf{w} \in \mathbb{R}^d, c \in \mathcal{C}\}$. This can be viewed as a simple NAS problem, with the difference that in neural nets the maps $\boldsymbol{\phi}_c$ also depend on parameter-weights $\mathbf{w}$, whereas here we only parameterize the last layer. We can write the standard bilevel optimization for feature map selection in the form of (9.2) for regularization parameter $\lambda > 0$:

$$\min_{c \in \mathcal{C}} \ \sum_{(x,y) \in V} \ell(\langle \mathbf{w}_c^*, \boldsymbol{\phi}_c(x) \rangle, y) \quad \text{s.t.} \quad \mathbf{w}_c^* = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \ \lambda \|\mathbf{w}\|_2^2 + \sum_{(x,y) \in T} \ell(\langle \mathbf{w}, \boldsymbol{\phi}_c(x) \rangle, y) \tag{9.10}$$

Note that as a non-architectural hyperparameter, λ is not tuned by weight-sharing.

How can we use weight-sharing to approximate (9.10) without solving for $\mathbf{w}_c^*$ for each configuration $c \in \mathcal{C}$? We take inspiration from the RS-WS algorithm described in Section 9.3.1, which allocates a resource—a minibatch of training examples—to a configuration at each iteration. Analogously, in Algorithm 25, we propose to allocate training examples to feature maps: at each iteration t we solve an ERM problem with each point featurized by a random map $\boldsymbol{\phi}_c, c \sim \mathrm{Unif}(\mathcal{C})$. The result $\mathbf{w}_t$ is thus *shared* among the feature maps rather than being the minimizer for any single one; as with RS-WS we find that, despite being trained using the uniform distribution over configurations, as showin in Figure 9.3 the shared-weights $\mathbf{w}_t$ are much better classifiers for data featurized using the best maps. We use this as validation signal in a successive halving procedure approximating (9.10) in $\log_2 |\mathcal{C}|$ regression solves.

We evaluate Algorithm 25 on two problems: kernel ridge regression over random Fourier features [Rahimi and Recht, 2008] on CIFAR-10 and regularized SVM classification over hashed Bag-of-n-Gram representations [Wang and Manning, 2012] on IMDb. The hyperparameters we tune are described in Appendix 9.B.2. In Figure 9.3, we show that the performance of the shared weights found at the first stage of Algorithm 25 on CIFAR-10 is strongly correlated with the standalone validation accuracy; thus, as in NAS we can use weight-sharing to obtain a useful

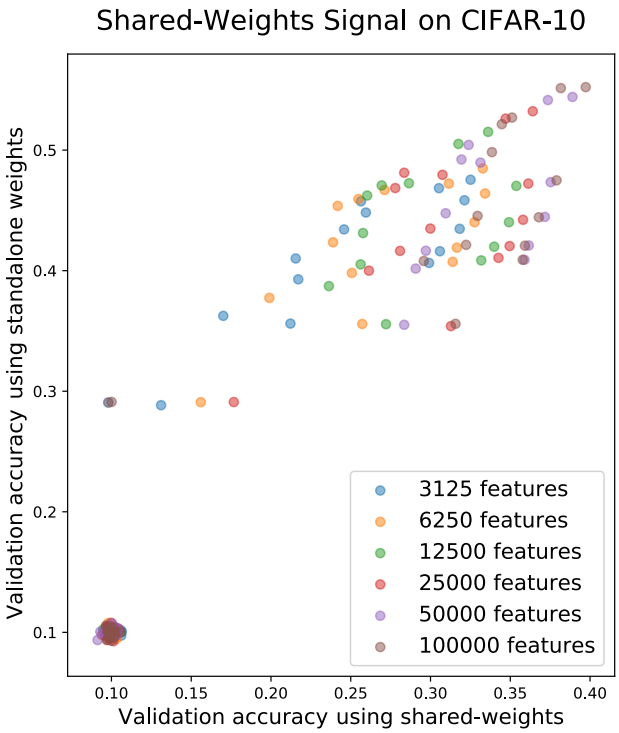

Figure 9.3: Validation accuracy of individual feature maps using shared weights compared to individual training.

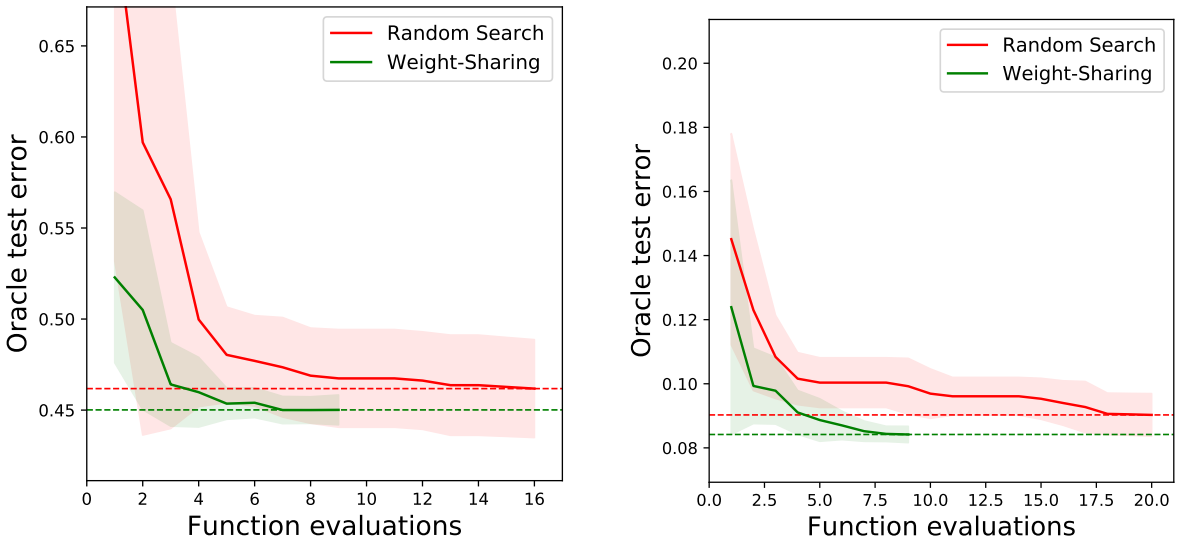

Figure 9.4: Oracle test-error on CIFAR-10 (left) and IMDb (right) as a function of number of solver calls. Here, *oracle* test-error refers to evaluation of a separately trained, non-weight-shared, classifier on the best config at any given round according to weight-sharing. All curves are averaged over 10 independent trials.

noisy indication of the performance of many configurations while training only one set of parameters. We can exploit this fact in Figure 9.4, where we see that Algorithm 25 obtains a better test accuracy in fewer calls to the ridge regression or SVM solver than random search. Interestingly, in terms of wallclock time the average weight-sharing solve is slower than the average single-configuration solve, but in Appendix 9.B.2 we describe how we can modify Algorithm 25 to outperform both random search and Hyperband [Li et al., 2018a] in terms of both accuracy and time. Note that we do not compare to Hyperband in the above plot because it computes many more solutions to lower-dimensional problems and so the evaluations are incomparable.

9.3.3 Generalization guarantees for the bilevel problem

While in our experiments we have used the bilevel formulation, it is not immediately clear, in NAS or feature map selection, that the joint ERM problem

$$\min_{c\in\mathcal{C},\mathbf{w}\in\mathcal{W}} \quad \lambda\|\mathbf{w}\|_2^2 \quad + \sum_{(x,y)\in T\cup V} \ell(\langle \mathbf{w}, \boldsymbol{\phi}_c(x)\rangle, y) \tag{9.11}$$

over the combined data would not also work. This question is interesting due to the widespread use of the bilevel formulation in NAS with weight-sharing. Especially when continuous relaxation [Liu et al., 2019b] is applied, architecture parameters in NAS appear more similar to regular model parameters rather than controls on the model complexity, so it is reasonable to wonder why most NAS practitioners have used the bilevel formulation. In this section we give an analysis suggesting that decomposing the objective can improve generalization by adapting to the sample complexity of the best configuration in $\mathcal{C}$, whereas ERM suffers that of the worst. For feature map selection our theory gives concrete excess risk bounds.

We start with a key fact about the weights that optimize the bilevel problem: they are optima $\arg\min_{\mathbf{w}\in\mathcal{W}} \mathcal{L}_T(\mathbf{w}, c)$ of the inner objective, i.e. elements of the *version space* $H_{c,T} = \{h_{\mathbf{w},c} : \mathbf{w} \in \arg\min_{\mathbf{u}\in\mathcal{W}} \mathcal{L}_T(\mathbf{u}, c)\}$ [Kearns et al., 1997, Equation 6]. We use the following data-dependent quantification of how much the hypotheses are restricted by the inner optimization for $N(F, \varepsilon)$ the L^∞*-covering-number* of a set of functions F at scale $\varepsilon > 0$, i.e. the number of L^∞ balls in an ε-cover of F [Mohri et al., 2012, Equation 3.60]:

Definition 9.3.1. The **version entropy** of $H(\mathcal{C}, \mathcal{W})$ at scale $\varepsilon > 0$ induced by the objective $\mathcal{L}_T$ over training data T is $\Lambda(H, \varepsilon, T) = \log N\left(\bigcup_{c\in\mathcal{C}} H_{c,T}, \varepsilon\right)$.

For finite $\mathcal{C}$, the version entropy is less than $\log|\mathcal{C}| + \max_{c\in\mathcal{C}} \log N(H_{c,T}, \varepsilon)$, so that the second term measures the worst-case complexity of the global minimizers of $\mathcal{L}_T$. In the feature selection problem, $\mathcal{L}_T$ is usually a strongly-convex loss due to regularization and so all version spaces are singleton sets, making the version entropy $\log|\mathcal{C}|$. In the other extreme case of nested model selection the version entropy reduces to the complexity of the version space of the largest model and so may not be informative. However, in practical problems such as NAS an inductive bias is often imposed via constraints on the number of input edges.

To bound the excess risk in terms of the version entropy, we first discuss an important assumption describing cases when we expect the bilevel approach to perform well:

Assumption 9.3.1. There exists a $c^* \in \mathcal{C}$ s.t. $(\mathbf{w}^*, c^*) \in \arg\min_{\mathcal{W}\times\mathcal{C}} \ell_{\mathcal{D}}(\mathbf{w}, c)$ for some $\mathbf{w}^* \in \mathcal{W}$ and s.t. w.h.p. over the training sample T at least one of the minima of the optimization induced by c^* and T has low excess risk, i.e. w.p. $1-\delta$ there exists $\mathbf{w} \in \arg\min_{\mathbf{u}\in\mathcal{W}} \mathcal{L}_T(\mathbf{u}, c^*)$ s.t. $\ell_{\mathcal{D}}(h_{\mathbf{w},c^*}) - \ell_{\mathcal{D}}(h^*) \leqslant \varepsilon^*(|T|, \delta)$ for excess risk ε^* and all $h^* \in H(\mathcal{W}, \mathcal{C})$.

This assumption requires that the inner optimization objective does not exclude all good classifiers for the optimal configuration. It asks nothing of the other configurations in $\mathcal{C}$, which may be arbitrarily bad, nor of the hypotheses found by the procedure, but prevents the case where even minimizing the objective $\mathcal{L}_T(\cdot, c^*)$ does not provide a set of good weights. Note that if the inner optimization is simply ERM over the training set T, i.e. $\mathcal{L}_T = \ell_T$, then standard learning-theoretic guarantees will give $\varepsilon^*(|T|, \delta)$ decreasing in the size $|T|$ of the training set and increasing at most poly-logarithmically in $\frac{1}{\delta}$. With this assumption, we can show the following guarantee for solutions to the bilevel optimization (9.2):

Theorem 9.3.1. If Assumption 9.3.1 holds, ℓ is B-bounded, and $(\mathbf{w}, c) \in \mathcal{W} \times \mathcal{C}$ solves the bilevel objective (9.2) for space $H(\mathcal{W}, \mathcal{C})$ and data V, T, as in Section 9.3.1 then w.p. $1 - 3\delta$

$$\ell_{\mathcal{D}}(h_{\mathbf{w},c}) \leqslant \min_{h\in H(\mathcal{W},\mathcal{C})} \ell_{\mathcal{D}}(h) + \varepsilon^*(|T|, \delta) + \inf_{\varepsilon>0} 3\varepsilon + \frac{3B}{2}\sqrt{\frac{2}{|V|}\left(\Lambda(H, \varepsilon, T) + \log\frac{1}{\delta}\right)} \tag{9.12}$$

Excess risk of feature map selection

To use this theorem we must bound the version entropy. In feature map selection, strong-convexity induces a unique minimum for each each ϕ_c and thus a singleton version space, so the bound is $\log|\mathcal{C}|$. Then we can show the following for Lipschitz (e.g. hinge) losses:

Corollary 9.3.1. For feature map selection with Lipschitz loss ℓ there exists $\lambda > 0$ s.t. bilevel optimization yields a hypothesis with excess risk less than $\mathcal{O}\left(\sqrt{\frac{\|\mathbf{w}^*\|_2^2+1}{|T|}\log\frac{1}{\delta}} + \sqrt{\frac{1}{|V|}\log\frac{|\mathcal{C}|}{\delta}}\right)$.

In the case of selection random Fourier approximations of kernels, we can show that we can compete with the optimal RKHS from among those associated with one of the configurations:

Corollary 9.3.2. In feature map selection suppose each map $\boldsymbol{\phi}_c, c \in \mathcal{C}$ is associated with a random Fourier feature approximation of a continuous shift-invariant kernel approximating an RKHS $\mathcal{H}_c$ and ℓ is the square loss. Then for sufficiently large d there exists $\lambda > 0$ s.t. w.p. $1-\delta$ solving (9.10) yields a hypothesis with excess risk w.r.t. $\mathcal{H}_c$ less than $\mathcal{O}\left(\frac{\log^2\frac{1}{\delta}}{\sqrt{|T|}} + \sqrt{\frac{1}{|V|}\log\frac{|\mathcal{C}|}{\delta}}\right)$.

In both cases we get bounds almost identical to the excess risk achievable by knowing the best configuration beforehand, up to a term depending weakly on the number of configurations. This improves upon solving the regular ERM objective, where we have to contend with the possibly high complexity of the hypothesis space induced by the worst configuration.

Version entropy and NAS

In simple settings Theorem 9.3.1 can guarantee excess risk almost as good as that of the (unknown) optimal configuration without assuming anything about the complexity or behavior of

sub-optimal configurations. However, for NAS we do not have a bound on the version entropy, which now depends on all of $\mathcal{C}$. Whether the version space of deep networks is small compared to the number of samples is unclear, although we gather some evidence below. The question amounts to how many (functional) global optima are induced by a training set of size $|T|$. In an idealized spin-glass model, Choromanska et al. [2015, Theorem 11.1] suggest that the number of critical points is exponential *only* in the number of layers, which would yield a small version entropy. It is conceivable that the quantity may be further bounded by the complexity of solutions explored by the algorithm when optimizing $\mathcal{L}_T$ [Nagarajan and Kolter, 2017, Bartlett et al., 2017]. On the other hand, Nagarajan and Kolter [2019] argue, with evidence in restricted settings, that even the most stringent implicit regularization cannot lead to a non-vacuous uniform convergence bound; if true more generally this would imply that the NAS version entropy is quite large.

9.4 Conclusion

This chapter presented our investigations into the weight-sharing paradigm in architecture search, approaching the challenge by studying optimization objectives—single-level and bilevel—and how they affected different aspects of learning: optimization and generalization. From the perspective of optimization we found that the single-level problem was both tractable to analyze and effective in practice, influencing the development of GAEA. However, our study of feature map selection suggests that bilevel optimization may be preferable for generalization, providing a possible explanation for its prevalence in the NAS literature. As a setting that both empirically demonstrates the usefulness of weight-sharing while being simple to stay and analyze, we expect feature map selection to spur further theoretical study of architecture search.

9.A Proofs

9.A.1 Optimization

This section contains proofs and generalizations of the nonconvex optimization results in Section 9.2.1. Throughout this section, $\mathbb{V}$ denotes a finite-dimensional real vector space with Euclidean inner product $\langle \cdot, \cdot \rangle$.

Preliminaries

Definition 9.A.1. [Zhang and He, 2018, Definition 2.1] Consider a closed and convex subset $\mathcal{X} \subset \mathbb{V}$. For any $\gamma > 0$ and $\phi : \mathcal{X} \mapsto \overline{\mathbb{R}}$ an everywhere-subdifferentiable function $f : \mathcal{X} \mapsto \mathbb{R}$ is called γ**-relatively-weakly-convex (**γ**-RWC)** w.r.t. ϕ if $f(\cdot) + \gamma\phi(\cdot)$ is convex on $\mathcal{X}$.

Definition 9.A.2. [Zhang and He, 2018, Definition 2.3] Consider a closed and convex subset $\mathcal{X} \subset \mathbb{V}$. For any $\lambda > 0$, function $f : \mathcal{X} \mapsto \mathbb{R}$, and DGF $\phi : \mathcal{X} \mapsto \overline{\mathbb{R}}$ the **Bregman proximal operator** of f is

$$\operatorname{prox}_\lambda(\mathbf{x}) = \arg\min_{\mathbf{u} \in \mathcal{X}} \lambda f(\mathbf{u}) + \mathcal{B}_\phi(\mathbf{u}||\mathbf{x}) \tag{9.13}$$

Definition 9.A.3. [Zhang and He, 2018, Equation 2.11] Consider a closed and convex subset $\mathcal{X} \subset \mathbb{V}$. For any $\lambda > 0$, function $f : \mathcal{X} \mapsto \mathbb{R}$, and DGF $\phi : \mathcal{X} \mapsto \overline{\mathbb{R}}$ the **Bregman stationarity** of f at any point $\mathbf{x} \in \mathcal{X}$ is

$$\Delta_\lambda(\mathbf{x}) = \frac{\mathcal{B}_\phi(\mathbf{x}||\operatorname{prox}_\lambda(\mathbf{x})) + \mathcal{B}_\phi(\operatorname{prox}_\lambda(\mathbf{x})||\mathbf{x})}{\lambda^2} \tag{9.14}$$

Results

Throughout this section let $\mathbb{V} = \bigtimes_{i=1}^b \mathbb{V}_i$ be a product space of b finite-dimensional real vector spaces $\mathbb{V}_i$, each with an associated norm $\|\cdot\|_i : \mathbb{V}_i \mapsto \mathbb{R}_{>0}$, and $\mathcal{X} = \bigtimes_{i=1}^b \mathcal{X}_i$ be a product set of b subsets $\mathcal{X}_i \subset \mathbb{V}_i$, each with an associated 1-strongly-convex DGF $\phi_i : \mathcal{X}_i \mapsto \overline{\mathbb{R}}$ w.r.t. $\|\cdot\|_i$. For each $i \in [b]$ will use $\|\cdot\|_{i,*}$ to denote the dual norm of $\|\cdot\|_i$ and for any element $\mathbf{x} \in \mathcal{X}$ we will use $\mathbf{x}_{[i]}$ to denote its component in block i and $\mathbf{x}_{[-i]}$ to denote the component across all blocks other than i. Define the functions $\|\cdot\| : \mathbb{V} \mapsto \mathbb{R}_{>0}$ and $\|\cdot\|_* \mathbb{V} \mapsto \mathbb{R}_{>0}$ for any $\mathbf{x} \in \mathbb{V}$ by $\|\mathbf{x}\|^2 = \sum_{i=1}^b \|\mathbf{x}_{[i]}\|_i^2$ and $\|\mathbf{x}\|_*^2 = \sum_{i=1}^b \|\mathbf{x}_{[i]}\|_{i,*}^2$, respectively, and the function $\phi : \mathcal{X} \mapsto \overline{\mathbb{R}}$ for any $\mathbf{x} \in \mathcal{X}$ by $\phi(\mathbf{x}) = \sum_{i=1}^b \phi_i(\mathbf{x})$. Finally, for any $n \in \mathcal{N}$ we will use $[n]$ to denote the set $\{1, \ldots, n\}$.

Setting 9.A.1. For some fixed constants $\gamma_i, L_i > 0$ for each $i \in [b]$ we have the following:

1. $f : \mathcal{X} \mapsto \mathbb{R}$ is everywhere-subdifferentiable with minimum $f^* > -\infty$ and for all $\mathbf{x} \in \mathcal{X}$ and each $i \in [b]$ the restriction $f(\cdot, \mathbf{x}_{[-i]})$ is γ_i-RWC w.r.t. ϕ_i.
2. For each $i \in [b]$ there exists a stochastic oracle G_i that for input $\mathbf{x} \in \mathcal{X}$ outputs a random vector $G_i(\mathbf{x}, \xi)$ s.t. $\mathbb{E}_\xi G_i(\mathbf{x}, \xi) \in \partial_i f(\mathbf{x})$, where $\partial_i f(\mathbf{x})$ is the subdifferential set of the restriction $f(\cdot, \mathbf{x}_{[-i]})$ at $\mathbf{x}_{[i]}$. Moreover, $\mathbb{E}_\xi \|G_i(\mathbf{x}, \xi)\|_{i,*}^2 \leqslant L_i^2$.

Define $\gamma = \max_{i \in [b]} \gamma_i$ and $L^2 = \sum_{i=1}^b L_i^2$.

Claim 9.A.1. $\|\cdot\|$ is a norm on $\mathbb{V}$.

Proof. Positivity and homogeneity are trivial. For the triangle inequality, note that for any $\lambda \in [0,1]$ and any $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ we have that

$$\begin{aligned}
\|\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\| &= \sqrt{\sum_{i=1}^{b} \|\lambda\mathbf{x}_{[i]} + (1-\lambda)\mathbf{y}_{[i]}\|_i^2} \\
&\leqslant \sqrt{\sum_{i=1}^{b} (\lambda\|\mathbf{x}_{[i]}\|_i + (1-\lambda)\|\mathbf{y}_{[i]}\|_i)^2} \\
&\leqslant \lambda\sqrt{\sum_{i=1}^{b} \|\mathbf{x}_{[i]}\|_i^2} + (1-\lambda)\sqrt{\sum_{i=1}^{b} \|\mathbf{y}_{[i]}\|_i^2} \\
&= \lambda\|\mathbf{x}\| + (1-\lambda)\|\mathbf{y}\|
\end{aligned} \tag{9.15}$$

where the first inequality follows by convexity of the norms $\|\cdot\|_i \; \forall \; i \in [b]$ and the fact that the Euclidean norm on $\mathbb{R}^b$ is nondecreasing in each argument, while the second inequality follows by convexity of the Euclidean norm on $\mathbb{R}^b$. Setting $\lambda = \frac{1}{2}$ and multiplying both sides by 2 yields the triangle inequality. □

Claim 9.A.2. $\frac{1}{2}\|\cdot\|_*^2$ is the convex conjugate of $\frac{1}{2}\|\cdot\|^2$.

Proof. Consider any $\mathbf{u} \in \mathbb{V}$. To upper-bound the convex conjugate note that

$$\begin{aligned}
\sup_{\mathbf{x}\in\mathbb{V}} \langle \mathbf{u}, \mathbf{x}\rangle - \frac{\|\mathbf{x}\|^2}{2} &= \sup_{\mathbf{x}\in\mathbb{V}} \sum_{i=1}^{b} \langle \mathbf{u}_{[i]}, \mathbf{x}_{[i]}\rangle - \frac{\|\mathbf{x}_{[i]}\|_i^2}{2} \\
&\leqslant \sup_{\mathbf{x}\in\mathbb{V}} \sum_{i=1}^{b} \|\mathbf{u}_{[i]}\|_{i,*}\|\mathbf{x}_{[i]}\|_i - \frac{\|\mathbf{x}_{[i]}\|_i^2}{2} \\
&= \frac{1}{2}\sum_{i=1}^{b} \|\mathbf{u}_{[i]}\|_{i,*}^2 \\
&= \frac{\|\mathbf{u}\|_*^2}{2}
\end{aligned} \tag{9.16}$$

where the first inequality follows by definition of a dual norm and the second by maximizing each term w.r.t. $\|\mathbf{x}_{[i]}\|_i$. For the lower bound, pick $\mathbf{x} \in \mathbb{V}$ s.t. $\langle \mathbf{u}_{[i]}, \mathbf{x}_{[i]}\rangle = \|\mathbf{u}_{[i]}\|_{i,*}\|\mathbf{x}_{[i]}\|_i$ and $\|\mathbf{x}_{[i]}\|_i = \|\mathbf{u}_{[i]}\|_{i,*} \; \forall \; i \in [b]$, which must exist by the definition of a dual norm. Then

$$\langle \mathbf{u}, \mathbf{x}\rangle - \frac{\|\mathbf{x}\|^2}{2} = \sum_{i=1}^{b} \langle \mathbf{u}_{[i]}, \mathbf{x}_{[i]}\rangle - \frac{\|\mathbf{x}_{[i]}\|_i^2}{2} = \frac{1}{2}\sum_{i=1}^{b} \frac{\|\mathbf{u}_{[i]}\|_{i,*}^2}{2} = \frac{\|\mathbf{u}\|_*^2}{2} \tag{9.17}$$

so $\sup_{\mathbf{x}\in\mathbb{V}} \langle \mathbf{u}, \mathbf{x}\rangle - \frac{1}{2}\|\mathbf{x}\|^2 \geqslant \frac{1}{2}\|\mathbf{u}\|_*^2$, completing the proof. □

Algorithm 26: Block-stochastic mirror descent over the product domain $\mathcal{X} = \bigtimes_{i=1}^{b} \mathcal{X}_i$ given DGFs ϕ_i associated with each $\mathcal{X}_i$.

Input: initialization $\mathbf{x}_1 \in \mathcal{X}$, number of steps $T \geqslant 1$, step-size sequence $\{\eta_t\}_{t=1}^T$
for iteration $t \in [T]$ **do**
  sample $i \sim \mathrm{Unif}[b]$
  set $\mathbf{x}_{t+1[-i]} = \mathbf{x}_{t[-i]}$
  get $\mathbf{g} = G_i(\mathbf{x}_t, \xi_t)$
  set $\mathbf{x}_{t+1[i]} = \arg\min_{\mathbf{u} \in \mathcal{X}_i} \eta_t \langle \mathbf{g}, \mathbf{u} \rangle + \mathcal{B}_{\phi_i}(\mathbf{u} || \mathbf{x}_{t[i]})$
Output: $\hat{\mathbf{x}} = \mathbf{x}_t$ w.p. $\frac{\eta_t}{\sum_{t=1}^T \eta_t}$.

Theorem 9.A.1. Let $\hat{\mathbf{x}}$ be the output of Algorithm 26 after T iterations with non-increasing step-size sequence $\{\eta_t\}_{t=1}^T$. Then under Setting 9.A.1, for any $\hat{\gamma} > \gamma$ we have that

$$\mathbb{E}\Delta_{\frac{1}{\hat{\gamma}}}(\hat{\mathbf{x}}) \leqslant \frac{\hat{\gamma} b}{\hat{\gamma} - \gamma} \frac{\min_{\mathbf{u} \in \mathcal{X}} f(\mathbf{u}) + \hat{\gamma} \mathcal{B}_\phi(\mathbf{u} || \mathbf{x}_1) - f^* + \frac{\hat{\gamma} L^2}{2b} \sum_{t=1}^T \eta_t^2}{\sum_{t=1}^T \eta_t} \tag{9.18}$$

where the expectation is w.r.t. ξ_t and the randomness of the algorithm.

Proof. Define transforms $\mathbf{U}_i, i \in [b]$ s.t. $\mathbf{U}_i^\top \mathbf{x} = \mathbf{x}_{[i]}$ and $\mathbf{x} = \sum_{i=1}^b \mathbf{U}_i \mathbf{x}_{[i]} \ \forall \ \mathbf{x} \in \mathcal{X}$. Let G be a stochastic oracle that for input $\mathbf{x} \in \mathcal{X}$ outputs $G(\mathbf{x}, i, \xi) = b \mathbf{U}_i G_i(\mathbf{x}, \xi)$. This implies $\mathbb{E}_{i,\xi} G(\mathbf{x}, i, \xi) = \frac{1}{b} \sum_{i=1}^b b \mathbf{U}_i \mathbb{E}_\xi G_i(\mathbf{x}, \xi) \in \sum_{i=1}^b \mathbf{U}_i \partial_i f(\mathbf{x}) = \partial f(\mathbf{x})$ and $\mathbb{E}_{i,\xi} \|G(\mathbf{x}, i, \xi)\|_*^2 = \frac{1}{b} \sum_{i=1}^b b^2 \mathbb{E}_\xi \|\mathbf{U}_i G_i(\mathbf{x}, \xi)\|_{i,*}^2 \leqslant b \sum_{i=1}^b L_i^2 = b L^2$. Then

$$\begin{aligned}
\mathbf{x}_{t+1} &= \mathbf{U}_i \left[\arg\min_{\mathbf{u} \in \mathcal{X}_i} \eta_t \langle g, u \rangle + \mathcal{B}_{\phi_i}(\mathbf{u} || \mathbf{x}_{t[i]}) \right] \\
&= \mathbf{U}_i \mathbf{U}_i^\top \left[\arg\min_{\mathbf{u} \in \mathcal{X}} \eta_t \langle \mathbf{U}_i G_i(\mathbf{x}_t, \xi_t), \mathbf{u} \rangle + \sum_{i=1}^b \mathcal{B}_{\phi_i}(\mathbf{u}_{[i]} || \mathbf{x}_{t[i]}) \right] \\
&= \arg\min_{\mathbf{u} \in \mathcal{X}} \frac{\eta_t}{b} \langle G(\mathbf{x}, i, \xi_t), \mathbf{u} \rangle + \mathcal{B}_\phi(\mathbf{u} || \mathbf{x}_t)
\end{aligned} \tag{9.19}$$

Thus Algorithm 26 is equivalent to Zhang and He [2018, Algorithm 1] with stochastic oracle $G(\mathbf{x}, i, \xi)$, step-size sequence $\{\eta_t / b\}_{t=1}^T$, and no regularizer. Note that ϕ is 1-strongly-convex w.r.t. $\|\cdot\|$ and f is γ-RWC w.r.t. ϕ, so in light of Claims 9.A.1 and 9.A.2 our setup satisfies Assumption 3.1 of Zhang and He [2018]. The result then follows from Theorem 3.1 of the same. □

Corollary 9.A.1. Under Setting 9.A.1 let $\hat{\mathbf{x}}$ be the output of Algorithm 26 with constant step-size $\eta_t = \sqrt{\frac{2b(f(\mathbf{x}_1) - f^*)}{\gamma L^2 T}} \ \forall \ t \in [T]$. Then we have

$$\mathbb{E}\Delta_{\frac{1}{2\gamma}}(\hat{\mathbf{x}}) \leqslant 2L \sqrt{\frac{2b\gamma(f(\mathbf{x}_1) - f^*)}{T}} \tag{9.20}$$

where the expectation is w.r.t. ξ_t and the randomness of the algorithm. Equivalently, we can reach a point $\hat{\mathbf{x}}$ satisfying $\mathbb{E}\Delta_{\frac{1}{2\gamma}}(\hat{\mathbf{x}}) \leqslant \varepsilon$ in $\frac{8\gamma b L^2 (f(\mathbf{x}_1) - f^*)}{\varepsilon^2}$ stochastic oracle calls.

A single-level analysis of ENAS and DARTS

In this section we apply our analysis to understanding two existing NAS algorithms, ENAS [Pham et al., 2018] and DARTS [Liu et al., 2019b]. For simplicity, we assume objectives induced by architectures in the relaxed search space are γ-smooth, which excludes components such as ReLU. However, such cases can be smoothed via Gaussian convolution, i.e. adding noise to every gradient; thus given the noisiness of SGD training we believe the following analysis is still informative [Kleinberg et al., 2018].

ENAS continuously relaxes $\mathcal{A}$ via a neural controller that samples architectures $a \in \mathcal{A}$, so $\Theta = \mathbb{R}^{\mathcal{O}(h^2)}$, where h is the number of hidden units. The controller is trained with Monte Carlo gradients. On the other hand, first-order DARTS uses a mixture relaxation but applies a softmax instead of constraining parameters to the simplex. Thus $\Theta = \mathbb{R}^{|E| \times |O|}$ for E the set of learnable edges and O the set of possible operations. If we assume that both algorithms use SGD for the architecture parameters then to compare them we are interested in their respective values of G_θ, which we will refer to as G_{ENAS} and G_{DARTS}. Before proceeding, we note again that our theory holds only for the single-level objective and when using SGD as the architecture optimizer, whereas both algorithms specify the bilevel objective and Adam [Kingma and Ba, 2015], respectively.

At a very high level, the Monte Carlo gradients used by ENAS are known to be high-variance, so G_{ENAS} may be much larger than G_{DARTS}, yielding faster convergence for DARTS, which is reflected in practice [Liu et al., 2019b]. We can also do a simple low-level analysis under the assumption that all architecture gradients are bounded entry-wise, i.e. in ℓ_∞-norm, by some constant; then since the squared ℓ_2-norm is bounded by the product of the dimension and the squared ℓ_∞-norm we have $G^2_{\text{ENAS}} = \mathcal{O}(h^2)$ while $G^2_{\text{DARTS}} = \mathcal{O}(|E||O|)$. Since ENAS uses a hidden state size of $h = 100$ and the DARTS search space has $|E| = 14$ edges and $|O| = 7$ operations, this also points to DARTS needing fewer iterations to converge.

9.A.2 Generalization

This section contains proofs of the generalization results in Section 9.3.3.

Settings and main assumption

We first describe the setting for which we prove our general result.

Setting 9.A.2. Let $\mathcal{C}$ be a set of possible architecture/configurations of finite size such that each $c \in \mathcal{C}$ is associated with a parameterized hypothesis class H_c=$\{h_{\mathbf{w},c} : \mathcal{X} \mapsto \mathcal{Y}' : \mathbf{w} \in \mathcal{W}\}$ for input space $\mathcal{X}$, output space $\mathcal{Y}'$, and fixed set of possible weights $\mathcal{W}$. We will measure the performance of a hypothesis $h_{\mathbf{w},c}$ on an input $x, y \in \mathcal{X} \times \mathcal{Y}$ for some output space $\mathcal{Y}$ using a B-bounded loss function $\ell : \mathcal{Y}' \times \mathcal{Y} \mapsto [0, B]$. Note that while the examples below have unbounded loss functions, in practice they are explicitly or implicitly bounded by explicit or implicit regularization.

We are given a training sample $T \sim \mathcal{D}^{|T|}$ and a validation sample $V \sim \mathcal{D}^{|V|}$, where $\mathcal{D}$ is some distribution over $\mathcal{X} \times \mathcal{Y}$. We will denote the the population risk by $\ell_{\mathcal{D}}(h_{\mathbf{w},c}) = \ell_{\mathcal{D}}(\mathbf{w}, c) =$

$\mathbb{E}_{(x,y)\sim\mathcal{D}}\ell(h_{\mathbf{w},c}(x),y)$ and for any finite subset $S\subset\mathcal{X}\times\mathcal{Y}$ we will denote the empirical risk over S by $\ell_S(h_{\mathbf{w},c})=\ell_S(\mathbf{w},c)=\frac{1}{|S|}\sum_{(x,y)\in S}\ell(h_{\mathbf{w},c}(x),y)$.

Finally, we will consider solutions of optimization problems that depend on the training data and architecture. Specifically, for any configuration $c\in\mathcal{C}$ and finite subset $S\subset\mathcal{X}\times\mathcal{Y}$ let $\mathcal{W}_c(S)\subset\mathcal{W}$ be the set of global minima of some optimization problem induced by S and c and let the associated version space [Kearns et al., 1997] be $H_c(S)=\{h_{\mathbf{w},c}:\mathbf{w}\in\mathcal{W}_c(S)\}$.

We next give as examples two specific settings encompassed by Setting 9.A.2.

Setting 9.A.3. For **feature map selection**, in Setting 9.A.2 the configuration space $\mathcal{C}$ is a set of feature maps $\boldsymbol{\phi}_c:\mathcal{X}\mapsto\mathbb{R}^d$, the set of weights $\mathcal{W}\subset\mathbb{R}^d$ consists of linear classifiers, for inputs $x\in\mathcal{X}$ the hypotheses are $h_{\mathbf{w},c}(x)=\langle\mathbf{w},\boldsymbol{\phi}_c(x)\rangle$ for $\mathbf{w}\in\mathcal{W}$, and so $\mathcal{W}_c(S)$ is the singleton set of solutions to the regularized ERM problem

$$\underset{\mathbf{w}\in\mathcal{W}}{\arg\min}\qquad\lambda\|\mathbf{w}\|_2^2\quad+\sum_{(x,y)\in S}\ell(\langle\mathbf{w},\boldsymbol{\phi}_c(x)\rangle,y)\tag{9.21}$$

for square loss $\ell:\mathcal{Y}'\times\mathcal{Y}\mapsto\mathbb{R}_{>0}$ and some coefficient $\lambda>0$.

Setting 9.A.4. For **neural architecture search**, in Setting 9.A.2 the configuration space consists of all possible choices of edges on a DAG of N nodes and a choice from one of K operations at each edge, for a total number of configurations bounded by K^{N^2}. The hypothesis $h_{\mathbf{w},c}:\mathcal{X}\mapsto\mathcal{Y}'$ is determined by a choice of architecture $c\in\mathcal{C}$ and a set of network weights $\mathbf{w}\in\mathcal{W}$ and the loss $\ell:\mathcal{Y}'\times\mathcal{Y}\mapsto\mathbb{R}_{>0}$ is the cross-entropy loss. In the simplest case $\mathcal{W}_c(S)$ is the set of global minima of the ERM problem

$$\min_{\mathbf{w}\in\mathcal{W}}\quad\sum_{(x,y)\in S}\ell(h_{\mathbf{w},c}(x),y)\tag{9.22}$$

We now state the main assumption we require.

Assumption 9.A.1. In Setting 9.A.2 there exists a good architecture $c^*\in\mathcal{C}$, i.e. one satisfying $(\mathbf{w}^*,c^*)\in\arg\min_{\mathcal{W}\times\mathcal{C}}\ell_{\mathcal{D}}(\mathbf{w},c)$ for some weights $\mathbf{w}^*\in\mathcal{W}$, such that w.p. $1-\delta$ over the drawing of training set $T\sim\mathcal{D}^{|T|}$ at least one of the minima of the optimization problem induced by c^* and T has low excess risk, i.e. $\exists\,\mathbf{w}\in\mathcal{W}_{c^*}(T)$ s.t.

$$\ell_{\mathcal{D}}(\mathbf{w},c^*)-\ell_{\mathcal{D}}(\mathbf{w}^*,c^*)\leqslant\varepsilon^*(|T|,\delta)\tag{9.23}$$

for some error function ε^*.

Clearly, we prefer error functions ε^* that are decreasing in the number of training samples $|T|$ and increasing at most poly-logarithmically in $\frac{1}{\delta}$. This assumption requires that if we knew the optimal configuration *a priori*, then the provided optimization problem will find a good set of weights for it. We will show how, under reasonable assumptions, Assumption 9.A.1 can be formally shown to hold in Settings 9.A.3 and 9.A.4.

Main result

Our general result will be stated in terms of covering numbers of certain function classes.

Definition 9.A.4. Let H be a class of functions from $\mathcal{X}$ to $\mathcal{Y}'$. For any $\varepsilon > 0$ the associated L^∞ **covering number** $N(H, \varepsilon)$ of H is the minimal positive integer k such that H can be covered by k balls of L^∞-radius ε.

The following is then a standard result in statistical learning theory (see e.g. Lafferty et al. [2010, Theorem 7.82]):

Theorem 9.A.2. Let H be a class of functions from $\mathcal{X}$ to $\mathcal{Y}$ and let $\ell : \mathcal{Y}' \times \mathcal{Y} \mapsto [0, B]$ be an L-Lipschitz, B-bounded loss function. Then for any distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ we have

$$\Pr_{S\sim\mathcal{D}^m}\left(\sup_{h\in H}|\ell_{\mathcal{D}}(h) - \ell_S(h)| \geqslant 3\varepsilon\right) \leqslant 2N\left(H, \varepsilon\right)\exp\left(-\frac{m\varepsilon^2}{2B^2}\right) \tag{9.24}$$

where we use the loss notation from Setting 9.A.2.

Before stating our theorem, we define a final quantity, which measures the log covering number of the version spaces induced by the optimization procedure over a given training set.

Definition 9.A.5. In Setting 9.A.2, for any sample $S \subset \mathcal{X} \times \mathcal{Y}$ define the **version entropy** to be $\Lambda(H, \varepsilon, S) = \log N\left(\bigcup_{c\in\mathcal{C}} H_c(S), \varepsilon\right)$.

Theorem 9.A.3. In Setting 9.A.2 let $(\hat{\mathbf{w}}, \hat{c}) \in \mathcal{W} \times \mathcal{C}$ be obtained as a solution to the following optimization problem:

$$\underset{\mathbf{w}\in\mathcal{W}, c\in\mathcal{C}}{\arg\min} \quad \ell_V(\mathbf{w}, c) \qquad \text{s.t.} \qquad \mathbf{w} \in \mathcal{W}_c(T) \tag{9.25}$$

Then under Assumption 9.A.1 we have w.p. $1 - 3\delta$ that

$$\begin{aligned}\ell_{\mathcal{D}}(\hat{\mathbf{w}}, \hat{c}) \leqslant\ & \ell_{\mathcal{D}}(\mathbf{w}^*, c^*) \\ & + \varepsilon^*(|T|, \delta) + B\sqrt{\frac{1}{2|V|}\log\frac{1}{\delta}} + \inf_{\varepsilon>0} 3\varepsilon + B\sqrt{\frac{2}{|V|}\left(\Lambda(H, \varepsilon, T) + \log\frac{1}{\delta}\right)}\end{aligned} \tag{9.26}$$

Proof. We have for any $\mathbf{w} \in \mathcal{W}_{c^*}(T)$ satisfying Equation 9.23, whose existence holds by Assumption 9.A.1, that

$$\begin{aligned}\ell_{\mathcal{D}}(\hat{\mathbf{w}}, \hat{c}) - \ell_{\mathcal{D}}(\mathbf{w}^*, c^*) \quad \leqslant \quad & \underbrace{\ell_{\mathcal{D}}(\hat{\mathbf{w}}, \hat{c}) - \ell_V(\hat{\mathbf{w}}, \hat{c})}_{1} \quad + \quad \underbrace{\ell_V(\hat{\mathbf{w}}, \hat{c}) - \ell_V(\mathbf{w}, c^*)}_{2} \\ + \quad & \underbrace{\ell_V(\mathbf{w}, c^*) - \ell_{\mathcal{D}}(\mathbf{w}, c^*)}_{3} \quad + \quad \underbrace{\ell_{\mathcal{D}}(\mathbf{w}, c^*) - \ell_{\mathcal{D}}(\mathbf{w}^*, c^*)}_{4}\end{aligned} \tag{9.27}$$

each term of which can be bounded as follows:

1. Since $\hat{\mathbf{w}} \in \mathcal{W}_{\hat{c}}(T)$ for some $\hat{c} \in \mathcal{C}$ the hypothesis space can be covered by the union of the coverings of $H_c(T)$ over $c \in \mathcal{C}$, so by Theorem 9.A.2 we have that w.p. $1 - \delta$

$$\ell_{\mathcal{D}}(\hat{\mathbf{w}}, \hat{c}) - \ell_V(\hat{\mathbf{w}}, \hat{c}) \leqslant \inf_{\varepsilon>0} 3\varepsilon + B\sqrt{\frac{2}{|V|}\left(\Lambda(H, \varepsilon, T) + \log\frac{1}{\delta}\right)} \tag{9.28}$$

2. By optimality of the pair $(\hat{\mathbf{w}}, \hat{c})$ and the fact that $\mathbf{w} \in \mathcal{W}_{c^*}(T)$ we have

$$\ell_V(\hat{\mathbf{w}}, \hat{c}) = \min_{c \in \mathcal{C}, \mathbf{w}' \in \mathcal{W}_c(T)} \ell_V(\mathbf{w}', \hat{c}) \leqslant \min_{\mathbf{w}' \in \mathcal{W}_{c^*}(T)} \ell_V(\mathbf{w}', c^*) \leqslant \ell_V(\mathbf{w}, c^*) \tag{9.29}$$

3. Hoeffding's inequality yields $\ell_V(\mathbf{w}, c^*) - \ell_{\mathcal{D}}(\mathbf{w}, c^*) \leqslant B\sqrt{\frac{1}{2|V|} \log \frac{1}{\delta}}$ w.p. $1 - \delta$
4. Assumption 9.A.1 states that $\ell_{\mathcal{D}}(\mathbf{w}, c^*) - \ell_{\mathcal{D}}(\mathbf{w}^*, c^*) \leqslant \varepsilon^*(|T|\delta)$ w.p. $1 - \delta$.

□

Applications

For the feature map selection problem, Assumption 9.A.1 holds by standard results for ℓ_2-regularized ERM for linear classification (e.g. Sridharan et al. [2008]):

Corollary 9.A.2. In Setting 9.A.3, suppose the loss function ℓ is Lipschitz. Then for regularization parameter $\lambda = \sqrt{\frac{1}{|T|} \log \frac{1}{\delta}}$ we have

$$\ell_{\mathcal{D}}(\mathbf{w}, c^*) - \ell_{\mathcal{D}}(\mathbf{w}^*, c^*) \leqslant \mathcal{O}\left(\sqrt{\frac{\|\mathbf{w}^*\|_2^2 + 1}{|T|} \log \frac{1}{\delta}}\right) \tag{9.30}$$

We can then directly apply Theorem 9.A.3 and the fact that the version entropy is bounded by $\log |\mathcal{C}|$ because the minimizer over the training set is always unique to get the following:

Corollary 9.A.3. In Setting 9.A.3 let $(\hat{\mathbf{w}}, \hat{c}) \in \mathcal{W} \times \mathcal{C}$ be obtained as a solution to the following optimization problem:

$$\operatorname*{arg\,min}_{\mathbf{w} \in \mathcal{W}, c \in \mathcal{C}} \quad \ell_V(\mathbf{w}, c) \qquad \text{s.t.} \qquad \mathbf{w} = \operatorname*{arg\,min}_{\mathbf{w} \in \mathcal{W}} \quad \lambda \|\mathbf{w}\|_2^2 + \sum_{(x,y) \in T} \ell(\langle \mathbf{w}, \boldsymbol{\phi}_c(x) \rangle, y) \tag{9.31}$$

Then

$$\ell_{\mathcal{D}}(\hat{\mathbf{w}}, \hat{c}) - \ell_{\mathcal{D}}(\mathbf{w}^*, c^*) \leqslant \mathcal{O}\left(\sqrt{\frac{\|\mathbf{w}^*\|_2^2 + 1}{|T|} \log \frac{1}{\delta}} + \sqrt{\frac{1}{|V|} \log \frac{|\mathcal{C}| + 1}{\delta}}\right) \tag{9.32}$$

In the special case of kernel selection we can apply generalization results for learning with random features to show that we can compete with the optimal RKHS from among those associated with one of the configurations [Rudi and Rosasco, 2017, Theorem 1]:

Corollary 9.A.4. In Setting 9.A.3, suppose each configuration $c \in \mathcal{C}$ is associated with a random Fourier feature approximation of a continuous shift-invariant kernel that approximates an RKHS $\mathcal{H}_c$. Suppose ℓ is the squared loss so that $(\hat{\mathbf{w}}, \hat{c}) \in \mathcal{W} \times \mathcal{C}$ is obtained as a solution to the following optimization problem:

$$\operatorname*{arg\,min}_{\mathbf{w} \in \mathcal{W}, c \in \mathcal{C}} \quad \ell_V(\mathbf{w}, c) \qquad \text{s.t.} \qquad \mathbf{w} = \operatorname*{arg\,min}_{\mathbf{w} \in \mathcal{W}} \quad \lambda \|w\|_2^2 + \sum_{(x,y) \in T} (\langle \mathbf{w}, \boldsymbol{\phi}_c(x) \rangle - y)^2 \tag{9.33}$$

If the number of random features $d = \mathcal{O}(\sqrt{|T|}\log\sqrt{|T|}/\delta)$ and $\lambda = 1/\sqrt{|T|}$ then w.p. $1-\delta$ we have

$$\ell_{\mathcal{D}}(h_{\hat{\mathbf{w}},\hat{c}}) - \min_{c\in\mathcal{C}}\min_{h\in\mathcal{H}_c}\ell_{\mathcal{D}}(h) \leqslant \mathcal{O}\left(\frac{\log^2\frac{1}{\delta}}{\sqrt{|T|}} + \sqrt{\frac{1}{|V|}\log\frac{|C|+1}{\delta}}\right) \tag{9.34}$$

In the case of neural architecture search we are often solving (unregularized) ERM in the inner optimization problem. In this case we can make an assumption weaker than Assumption 9.A.1, namely that the set of empirical risk minimizers contains a solution that, rather than having low excess risk, simply has low generalization error; then applying Hoeffding's inequality yields the following:

Corollary 9.A.5. In Setting 9.A.2 let $(\hat{\mathbf{w}},\hat{c}) \in \mathcal{W}\times\mathcal{C}$ be obtained as a solution to the following optimization problem:

$$\operatorname*{arg\,min}_{\mathbf{w}\in\mathcal{W},c\in\mathcal{C}} \quad \ell_V(\mathbf{w},c) \qquad \text{s.t.} \qquad \mathbf{w} \in \operatorname*{arg\,min}_{\mathbf{w}'\in\mathcal{W}} \ell_T(\mathbf{w}',c) \tag{9.35}$$

Suppose there exists $c^* \in \mathcal{C}$ satisfying $(\mathbf{w}^*,c^*) \in \arg\min_{\mathcal{W}\times\mathcal{C}}\ell_{\mathcal{D}}(\mathbf{w},c)$ for some weights $\mathbf{w}^* \in \mathcal{W}$ such that w.p. $1-\delta$ over the drawing of training set $T \sim \mathcal{D}^{|T|}$ at least one of the minima of the optimization problem induced by c^* and T has low generalization error, i.e. $\exists\ \mathbf{w} \in \arg\min_{\mathbf{w}'\in\mathcal{W}}\ell_T(\mathbf{w}',c^*)$ s.t.

$$\ell_{\mathcal{D}}(\mathbf{w},c^*) - \ell_T(\mathbf{w}^*,c^*) \leqslant \varepsilon^*(|T|,\delta) \tag{9.36}$$

for some error function ε^*. Then we have w.p. $1-4\delta$ that

$$\begin{aligned}\ell_{\mathcal{D}}(\hat{\mathbf{w}},\hat{c}) \leqslant \ell_{\mathcal{D}}(\mathbf{w}^*,c^*) + \varepsilon^*(|T|,\delta) + B\sqrt{\frac{1}{2|V|}\log\frac{1}{\delta}} + B\sqrt{\frac{1}{2|T|}\log\frac{1}{\delta}} \\ + \inf_{\varepsilon>0} 3\varepsilon + B\sqrt{\frac{2}{|V|}\left(\Lambda(H,\varepsilon,T) + \log\frac{1}{\delta}\right)}\end{aligned} \tag{9.37}$$

9.B Experimental details

9.B.1 GAEA

We provide additional detail on the experimental setup and hyperparameter settings used for each benchmark studied in Section 9.2.2. Code to reproduce the results for GAEA is available here: `https://github.com/liamcli/gaea_release`.

DARTS search space

We consider the same search space as DARTS [Liu et al., 2019b], which has become one of the standard search spaces for CNN cell search [Xie et al., 2019, Chen et al., 2019, Noy et al., 2019, Liang et al., 2019]. Following the evaluation procedure used in Liu et al. [2019b] and Xu et al. [2020b] , our evaluation of GAEA PC-DARTS consists of three stages:

- **Stage 1:** In the search phase, we run GAEA PC-DARTS with 5 random seeds to reduce variance from different initialization of the shared-weights network.[2]
- **Stage 2:** We evaluate the best architecture identified by each search run by training from scratch.
- **Stage 3:** We perform a more thorough evaluation of the best architecture from stage 2 by training with ten different random seed initializations.

For completeness, we describe the convolutional neural network search space considered. A cell consists of 2 input nodes and 4 intermediate nodes for a total of 6 nodes. The nodes are ordered and subsequent nodes can receive the output of prior nodes as input so for a given node k, there are $k-1$ possible input edges to node k. Therefore, there are a total of $2+3+4+5=14$ edges in the weight-sharing network.

An architecture is defined by selecting 2 input edges per intermediate node and also selecting a single operation per edge from the following 8 operations: (1) 3×3 separable convolution, (2) 5×5 separable convolution, (3) 3×3 dilated convolution, (4) 5×5 dilated convolution, (5) max pooling, (6) average pooling, (7) identity (8) zero. We use the same search space to design a "normal" cell and a "reduction" cell; the normal cells have stride 1 operations that do not change the dimension of the input, while the reduction cells have stride 2 operations that half the length and width dimensions of the input. In the experiments, for both cell types, , after which the output of all intermediate nodes are concatenated to form the output of the cell.

Stage 1: Architecture Search For stage 1, as is done by DARTS and PC-DARTS, we use GAEA PC-DARTS to update architecture parameters for a smaller shared-weights network in the search phase with 8 layers and 16 initial channels. All hyperparameters for training the weight-sharing network are the same as that used by PC-DARTS:

```
train:
    scheduler: cosine
    lr_anneal_cycles: 1
    smooth_cross_entropy: false
    batch_size: 256
    learning_rate: 0.1
    learning_rate_min: 0.0
    momentum: 0.9
    weight_decay: 0.0003
    init_channels: 16
    layers: 8
    autoaugment: false
    cutout: false
    auxiliary: false
    drop_path_prob: 0
    grad_clip: 5
```

[2]Note [Liu et al., 2019b] trains the weight-sharing network with 4 random seeds. However, since PC-DARTS is significantly faster than DARTS, the cost of an additional seed is negligible.

For GAEA PC-DARTS, we initialize the architecture parameters with equal weight across all options (equal weight across all operations per edge and equal weight across all input edges per node). Then, we train the shared-weights network for 10 epochs without performing any architecture updates to warmup the weights. Then, we use a learning rate of 0.1 for the exponentiated gradient update for GAEA PC-DARTS.

Stage 2 and 3: Architecture Evaluation For stages 2 and 3, we train each architecture for 600 epochs with the same hyperparameter settings as PC-DARTS:

```
train:
    scheduler: cosine
    lr_anneal_cycles: 1
    smooth_cross_entropy: false
    batch_size: 128
    learning_rate: 0.025
    learning_rate_min: 0.0
    momentum: 0.9
    weight_decay: 0.0003
    init_channels: 36
    layers: 20
    autoaugment: false
    cutout: true
    cutout_length: 16
    auxiliary: true
    auxiliary_weight: 0.4
    drop_path_prob: 0.3
    grad_clip: 5
```

Broad Reproducibility Our 'broad reproducibility' results in Table 9.3 show the final stage 3 evaluation performance of GAEA PC-DARTS for 2 additional sets of random seeds from stage 1 search. The performance of GAEA PC-DARTS for one set is similar to that reported in Table 9.1, while the other is on par with the performance reported for PC-DARTS in Xu et al. [2020b]. We do observe non-negligible variance in the performance of the architecture found by different random seed initializations of the shared-weights network, necessitating running multiple searches before selecting an architecture. We also found that it was possible to identify and eliminate poor performing architectures in just 20 epochs of training during stage 2 intermediate evaluation, thereby reducing the total training cost by over 75% (we only trained 3 out of 10 architectures for the entire 600 epochs).

We depict the top architectures found by GAEA PC-DARTS for CIFAR-10 and ImageNet in Figure 9.5 and detailed results in Tables 9.4 and 9.5.

Table 9.3: GAEA PC-DARTS Stage 3 Evaluation for 3 sets of random seeds.

Stage 3 Evaluation		
Set 1 (Reported)	Set 2	Set 3
2.50 ± 0.07	2.50 ± 0.09	2.60 ± 0.09

Table 9.4: CIFAR-10 performance comparisons with manually designed networks and those found by SOTA NAS methods, mainly on the DARTS search space [Liu et al., 2019b]. Results grouped by the type of search method: manually designed, full-evaluation NAS, and weight-sharing NAS. All test errors are for models trained with auxiliary towers and cutout (parameter counts exclude auxiliary weights). Test errors we report are averaged over 10 seeds. "-" indicates that the field does not apply while "N/A" indicates unknown. Note that search cost is hardware-dependent; our results used Tesla V100 GPUs.

Architecture	**Test Error** Best	Average	**Params** (M)	**Search Cost** (GPU Days)	**Comparable Search Space**	**Search Method**
Shake-Shake [DeVries and Taylor, 2017]	N/A	2.56	26.2	-	-	manual
PyramidNet [Yamada et al., 2018]	2.31	N/A	26	-	-	manual
NASNet-A* [Zoph et al., 2018]	N/A	2.65	3.3	2000	N	RL
AmoebaNet-B* [Real et al., 2019]	N/A	2.55 ± 0.05	2.8	3150	N	evolution
ProxylessNAS‡ [Cai et al., 2019]	2.08	N/A	5.7	4	N	gradient
ENAS [Pham et al., 2018]	2.89	N/A	4.6	0.5	Y	RL
Random search WS† [Li and Talwalkar, 2019]	2.71	2.85 ± 0.08	3.8	0.7	Y	random
ASNG-NAS [Akimoto et al., 2019]	N/A	2.83 ± 0.14	3.9	0.1	Y	gradient
SNAS [Xie et al., 2019]	N/A	2.85 ± 0.02	2.8	1.5	Y	gradient
DARTS (1st-order)† [Liu et al., 2019b]	N/A	3.00 ± 0.14	3.3	0.4	Y	gradient
DARTS (2nd-order)† [Liu et al., 2019b]	N/A	2.76 ± 0.09	3.3	1	Y	gradient
PDARTS# [Chen et al., 2019]	2.50	N/A	3.4	0.3	Y	gradient
PC-DARTS† [Xu et al., 2020b]	N/A	2.57 ± 0.07	3.6	0.1	Y	gradient
GAEA PC-DARTS† (Ours)	2.39	2.50 ± 0.06	3.7	0.1	Y	gradient

* We show results for networks with a comparable number of parameters.

† For fair comparison to other work, we show the search cost for training the shared-weights network with a single initialization.

‡ Search space and backbone architecture (PyramidNet) differ from the DARTS setting.

PDARTS results not reported for multiple seeds. Additionally, PDARTS uses deeper weight-sharing networks during search, on which PC-DARTS has also been shown to improve performance [Xu et al., 2020b], so we GAEA PC-DARTS to further improve if modified similarly.

NAS-Bench-1Shot1

The NAS-Bench-1Shot1 benchmark [Zela et al., 2020b] contains 3 different search spaces that are subsets of the NAS-Bench-101 search space. The search spaces differ in the number of nodes and the number of input edges selected per node. We refer the reader to [Zela et al., 2020b] for details about each individual search space.

Of the NAS methods evaluated in Zela et al. [2020b], PC-DARTS had the most robust performance across the three search spaces and converged to the best architecture in search spaces 1 and 3. GDAS, a probabilistic gradient NAS method, achieved the best performance on search

Table 9.5: ImageNet performance comparisons of manually designed networks and those found by SOTA NAS methods, mainly on the DARTS search space [Liu et al., 2019b]. Results are grouped by the type of search method: manually designed, full-evaluation NAS, and weight-sharing NAS. All test errors are for models trained with auxiliary towers and cutout but no other modifications, e.g. label smoothing [Müller et al., 2019], AutoAugment [Cubuk et al., 2019], Swish [Ramachandran et al., 2017], squeeze and excite modules [Hu et al., 2018], etc. "-" indicates that the field does not apply while "N/A" indicates unknown. Note that search cost is hardware-dependent; our results used Tesla V100 GPUs.

Architecture	**Source**	**Test Error** top-1	top-5	**Params** (M)	**Search Cost** (GPU Days)	**Search Method**
MobileNet	[Howard et al., 2017]	29.4	10.5	4.2	-	manual
ShuffleNet V2 2x	[Ma et al., 2018]	25.1	N/A	~ 5	-	manual
NASNet-A*	[Zoph et al., 2018]	26.0	8.4	5.3	1800	RL
AmoebaNet-C*	[Real et al., 2019]	24.3	7.6	6.4	3150	evolution
DARTS†	[Liu et al., 2019b]	26.7	8.7	4.7	4.0	gradient
SNAS	[Xie et al., 2019]	27.3	9.2	4.3	1.5	gradient
ProxylessNAS‡	[Cai et al., 2019]	24.9	7.5	7.1	8.3	gradient
PDARTS$^\#$	[Chen et al., 2019]	24.4	7.4	4.9	0.3	gradient
PC-DARTS (CIFAR-10)†	[Xu et al., 2020b]	25.1	7.8	5.3	0.1	gradient
PC-DARTS (ImageNet)†	[Xu et al., 2020b]	24.2	7.3	5.3	3.8	gradient
GAEA PC-DARTS (CIFAR-10)†	Ours	24.3	7.3	5.3	0.1	gradient
GAEA PC-DARTS (ImageNet)†	Ours	24.0	7.3	5.6	3.8	gradient

$^{*,\dagger,\ddagger,\#}$ See notes in Table 9.4.

space 2. Hence, we focused on applying a geometry-aware approach to PC-DARTS. We implemented GAEA PC-DARTS within the repository provided by the authors of Zela et al. [2020b] available at `https://github.com/automl/nasbench-1shot1`. We used the same hyperparameter settings for training the weight-sharing network as that used by Zela et al. [2020b] for PC-DARTS. Similar to the previous benchmark, we initialize architecture parameters to allocate equal weight to all options. For the architecture updates, the only hyperparameter for GAEA PC-DARTS is the learning rate for exponentiated gradient, which we set to 0.1.

As mentioned in Section 9.2.2, the search spaces considered in this benchmark differ in that operations are applied after aggregating all edge inputs to a node instead of per edge input as in the DARTS and NAS-Bench-201 search spaces. This structure inherently limits the size of the weight-sharing network to scale with the number of nodes instead of the number of edges ($\mathcal{O}(|\text{nodes}|^2)$), thereby limiting the degree of overparameterization. Understanding the impact of overparameterization on the performance of weight-sharing NAS methods is a direction for future study.

NAS-Bench-201

The NAS-Bench-201 benchmark [Dong and Yang, 2020] evaluates a single search space across 3 datasets: CIFAR-10, CIFAR-100, and a miniature version of ImageNet (ImageNet-16-120). ImageNet-16-120 is a downsampled version of ImageNet with 16×16 images and 120 classes

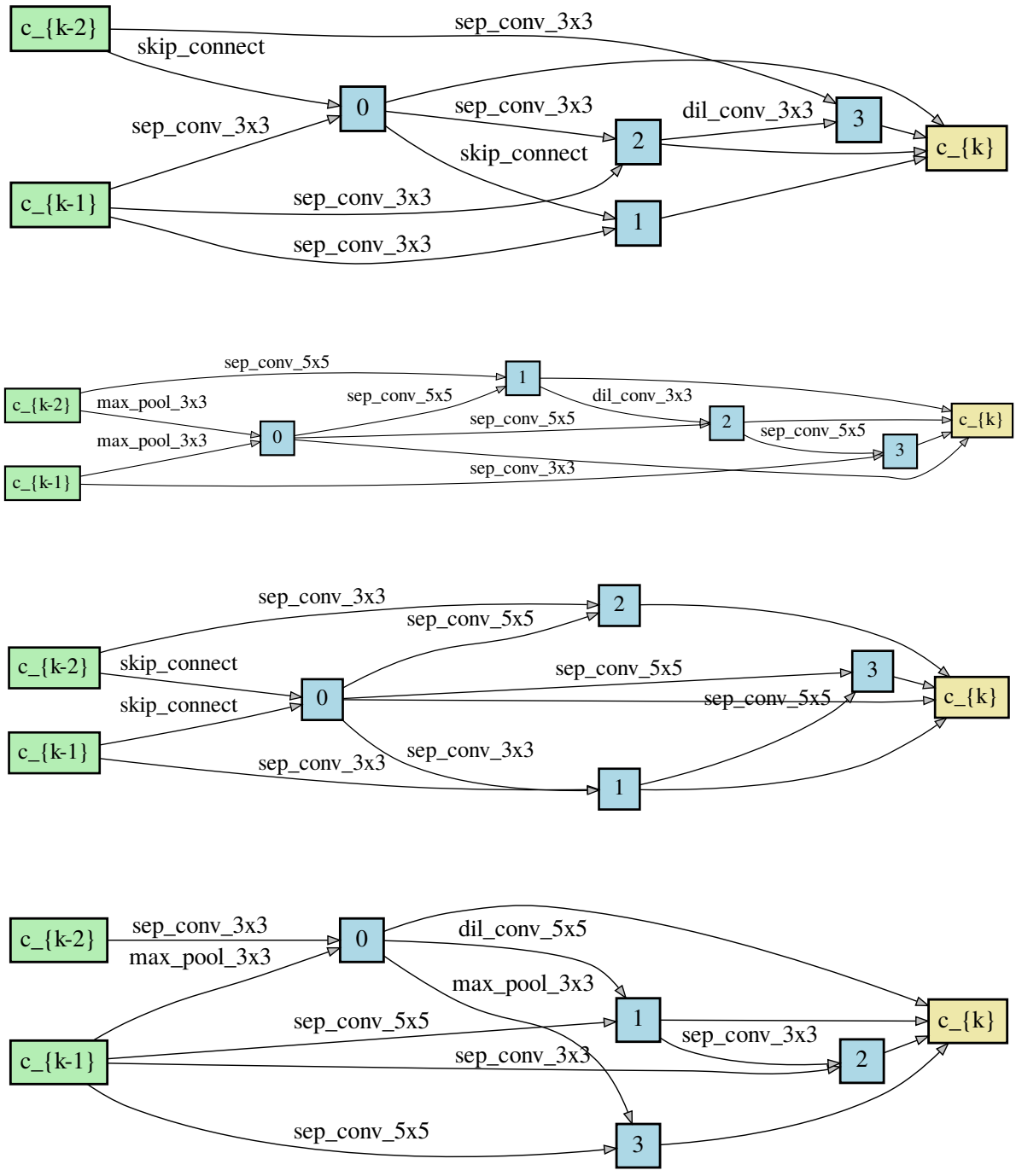

Figure 9.5: The best normal and reduction cells found by GAEA PC-DARTS on CIFAR-10 (top) and ImageNet (bottom). From top to bottom we show: CIFAR-10: Normal Cell, CIFAR-10: Reduction Cell, ImageNet: Normal Cell, ImageNet: Reduction Cell.

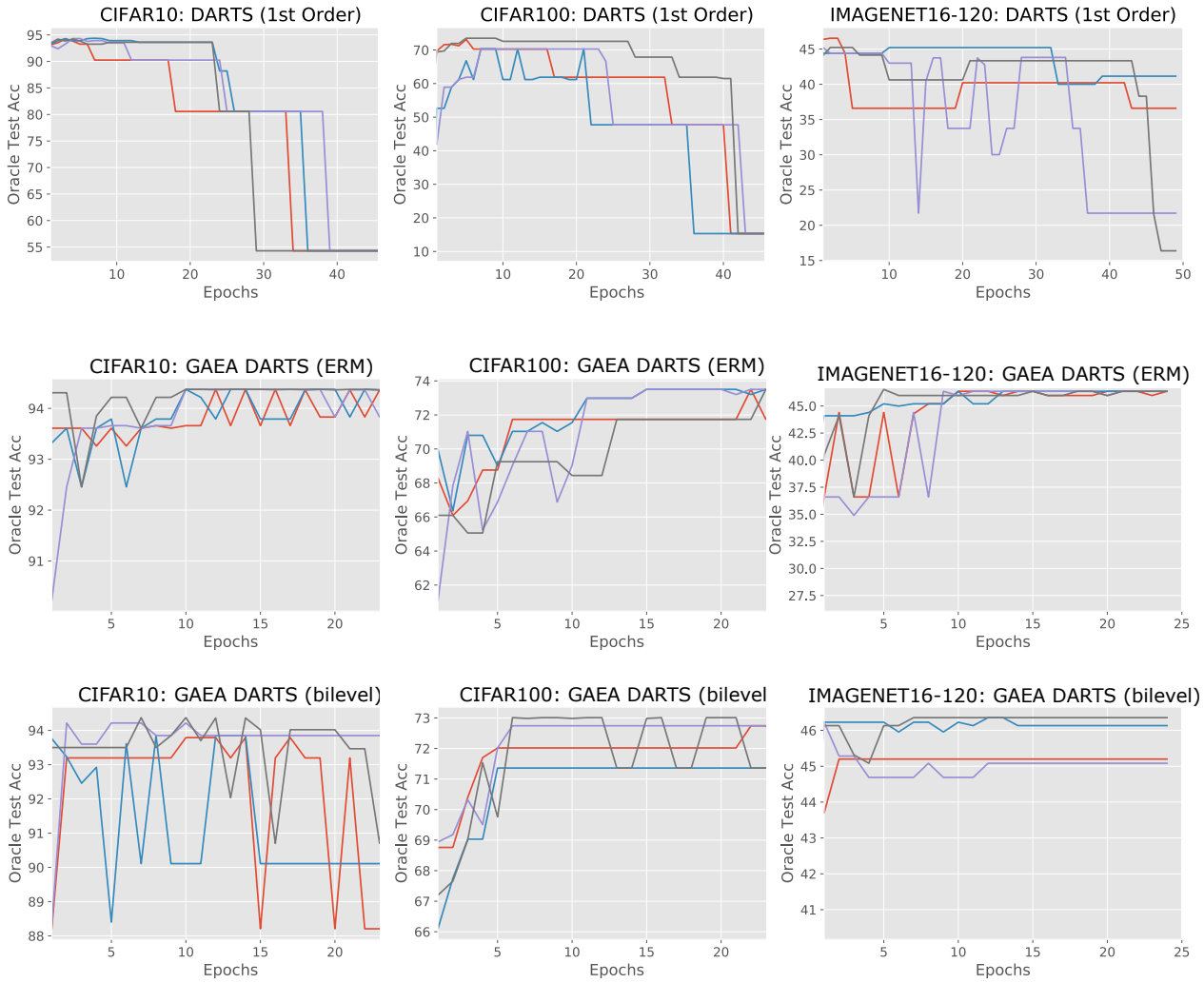

Figure 9.6: Evolution over search phase epochs of the best architecture according to the NAS method on NAS-Bench-201. DARTS (first-order) converges to nearly all skip connections while GAEA is able to suppress overfitting to the mixture relaxation by encouraging sparsity in operation weights.

for a total of 151.7k training images, 3k validation images, and 3k test images. The authors of Dong and Yang [2020] evaluated the architecture search performance of multiple weight-sharing methods and traditional hyperparameter optimization methods on all three datasets. According to the results from Dong and Yang [2020], GDAS outperformed other weight-sharing methods by a large margin. Hence, we first evaluated the performance of GAEA GDAS on each of the three datasets. Our implementation of GAEA GDAS uses an architecture learning rate of 0.1, which matches the learning rate used for GAEA approaches in the previous two benchmarks. Additionally, we run GAEA GDAS for 150 epochs instead of 250 epochs used for GDAS in the original benchmarked results; this is why the search cost is lower for GAEA GDAS. All other hyperparameter settings are the same. Our results for GAEA GDAS is comparable to

the reported results for GDAS on CIFAR-10 and CIFAR-100 but slightly lower on ImageNet-16-120. Compared to our reproduced results for GDAS, GAEA GDAS outperforms GDAS on CIFAR-100 and matches it on CIFAR-10 and ImageNet-16-120.

Next, to see if we can use GAEA to further improve the performance of weight-sharing methods, we evaluated GAEA DARTS (first order) applied to both the single-level (ERM) and bi-level optimization problems. Again, we used a learning rate of 0.1 and trained GAEA DARTS for 25 epochs on each dataset. The one additional modification we made was to exclude the `zero` operation, which limits GAEA DARTS to a subset of the search space. To isolate the impact of this modification, we also evaluated first-order DARTS with this modification. Similar to [Dong and Yang, 2020], we observe DARTS with this modification to also converge to architectures with nearly all skip connections, resulting in similar performance as that reported in Dong and Yang [2020]. We present the learning curves of the oracle architecture recommended by DARTS and GAEA DARTS (when excluding zero operation) over the training horizon for 4 different runs in Figure 9.6. For GAEA GDAS and GAEA DARTS, we train the weight-sharing network with the following hyperparameters:

```
train:
    scheduler: cosine
    lr_anneal_cycles: 1
    smooth_cross_entropy: false
    batch_size: 64
    learning_rate: 0.025
    learning_rate_min: 0.001
    momentum: 0.9
    weight_decay: 0.0003
    init_channels: 16
    layers: 5
    autoaugment: false
    cutout: false
    auxiliary: false
    auxiliary_weight: 0.4
    drop_path_prob: 0
    grad_clip: 5
```

Surprisingly, we observe single-level optimization to yield better performance than solving the bi-level problem with GAEA DARTS on this search space. In fact, the performance of GAEA DARTS (ERM) not only exceeds that of GDAS, but also outperforms traditional hyper-parameter optimization approaches on all three datasets, nearly reaching the optimal accuracy on all three datasets. In contrast, GAEA DARTS (bi-level) outperforms GDAS on CIFAR-100 and ImageNet-16-120 but underperforms slightly on CIFAR-10. The single-level results on this benchmark provides concrete support for our convergence analysis, which only applies to the ERM problem. As noted in Section 9.2.2, the search space considered in this benchmark differs from the prior two in that there is no subsequent edge pruning. Additionally, the search space is fairly small with only 3 nodes for which architecture decisions must be made. The success of GAEA DARTS (ERM) on this benchmark indicate the need for a better understanding of when

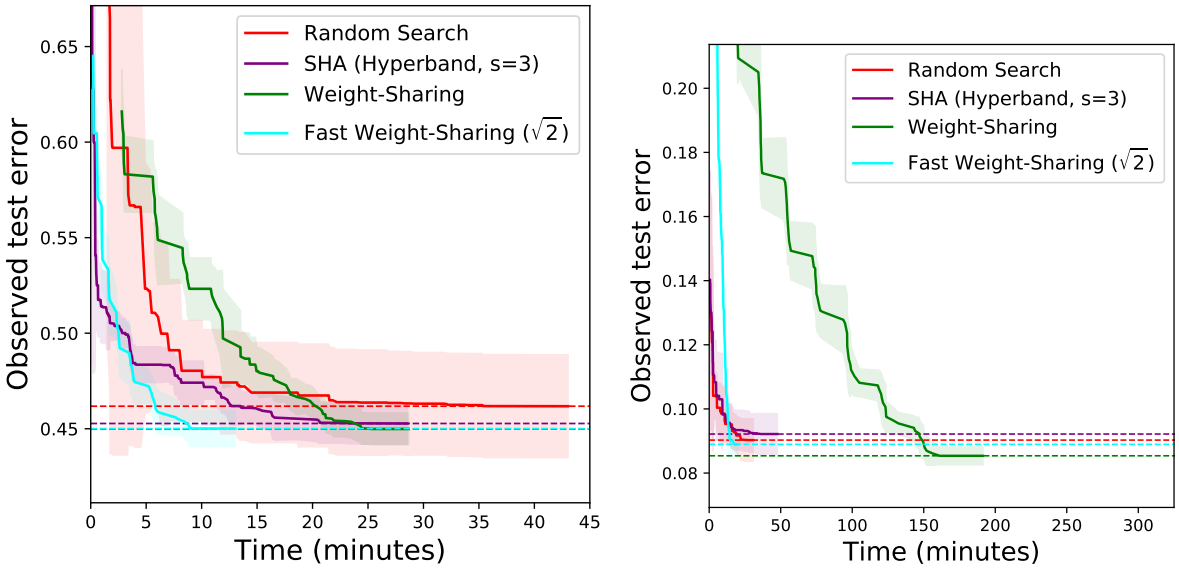

Figure 9.7: Observed test-error on CIFAR-10 (left) and IMDb (right) as a function of number of time. All curves are averaged over 10 independent trials.

single-level optimization should be used in favor of the default bi-level optimization problem and how the search space impacts the decision.

9.B.2 Feature map selection

Fast weight-sharing algorithm

In addition to the weight-sharing approach described in Algorithm 25, we evaluate a variant we call *Fast Weight-Sharing* in which at each round t the feature dimension used is a constant multiplicative factor greater than that used on the previous round (we use $\sqrt{2}$, finding doubling to be too aggressive), with the final dimension d reached only at the last round. This is reminiscent of the resource allocation scheme of Li et al. [2018a], who after each round of successive elimination give the promoted arms multiplicatively more features. The results, showing that Fast Weight-Sharing outperforms the strong baseline of successive halving and random search, are displayed in Figure 9.7.

All results are for 10 independent trials of each algorithm with the maximum number of features used by all competing algorithms set to 100K. For both the Weight-Sharing and Fast Weight-Sharing algorithms we started with 256 different configurations. For Figure 9.3 we varied the feature dimension as shown and considered the correlation between shared-weights performance and standalone performance fo 32 different configurations. The remaining settings are in the provided code: `https://github.com/mkhodak/weight-sharing`.

Hyperparameter tuning setup

For selecting feature maps on CIFAR-10 and IMDb we used the Ridge regression and SVM solvers from `scikit-learn` [Pedregosa et al., 2011] to solve the inner ℓ_2-regularized ERM problems. The regularization was fixed to $\lambda = \frac{1}{2}$ since weight-sharing does not tune non-architectural hyperparameters; the same λ was used for all search algorithms. We tested whether including λ in the search space helped random search and found that it did not, or even caused random search to do worse; this is likely due to the bandwidth parameter playing a similar role.

Random Fourier features on CIFAR-10 For the search space we used the same kernel configuration settings as Li et al. [2018a, Table 5] but replacing the regularization parameter by the option to use the Laplace kernel instead of the Gaussian kernel. The data split used was the standard 40K/10K/10K for training/validation/testing.

Hashed bag-of-n-grams on IMDb For the search space we tuned whether to just tokenize on spaces, split on punctuation, or use the NLTK [Loper and Bird, 2002] tokenizer; whether to remove stopwords; whether to lowercase; the n-gram order between 1-3; whether to binarize bins; whether to weight using Naive-Bayes [Wang and Manning, 2012] or SIF [Arora et al., 2017]; the constant α for these weighting schemes between $[10^{-5}, 10^{1}]$ on a logarithmic scale; and whether to use nothing, normlizing, or averaging as preprocessing. The data split was 25K/12.5K/12.5K for training/validation/testing.

Chapter 10

Finding neural operations for diverse tasks

Automated machine learning (AutoML) and neural architecture search (NAS) are often motivated by a vision of democratizing ML by reducing the need for expert design on a variety of tasks. While NAS has grown rapidly with developments such as weight-sharing [Pham et al., 2018] and "NAS-benches" [Ying et al., 2019, Zela et al., 2020b], most efforts focus on search spaces that glue together established primitives for well-studied tasks like vision and text [Liu et al., 2019b, Li and Talwalkar, 2019, Xu et al., 2020b] or on issues such as latency [Cai et al., 2020, Fang et al., 2020]. In this chapter, we revisit the broader vision of NAS and propose to move towards much more general search spaces while still exploiting successful network topologies. To do so we focus on expanding the set of operations, which is usually fairly small; for example, that of the well-studied DARTS space has eight elements: a few types of convolution and pooling layers [Liu et al., 2019b]. The baseline approach for expanding this set—adding operations one-by-one—scales poorly and will not result in new operations when faced with new types of data.

Our contributions are two different search spaces for increasing the size of the operation space while extending the well-known benefits of CNNs. The first is one called *XD-Operations* that mimic the inductive bias of standard multi-channel convolutions while being much more expressive: we prove that it includes many named operations across multiple application areas. Starting with any standard backbone such as ResNet, we show how to transform it into a search space over XD-operations and how to traverse the space using a simple weight-sharing scheme. On a diverse set of tasks—solving PDEs, distance prediction for protein folding, and music modeling—our approach consistently yields models with lower error than baseline networks and often even lower error than expert-designed domain-specific approaches.

Seeking an approach that is faster, simpler, and broadly applicable, in our second search space we again fix a standard CNN topology but propose to search for the right kernel sizes and dilations its operations should take on. This dramatically expands the model's capacity to extract features at multiple resolutions for different types of data while only requiring search over the operation space. To overcome the efficiency challenges of naive weight-sharing in this search space, we introduce DASH, a differentiable NAS algorithm that computes the mixture-of-operations using the Fourier diagonalization of convolution, achieving both a better asymptotic complexity and

[0]The work presented in this chapter first appeared in Roberts et al. [2021] and Shen et al. [2022].

an up-to-10x search time speedup in practice. We evaluate DASH on ten tasks spanning a variety of application domains such as PDE solving, protein folding, and heart disease detection. DASH outperforms state-of-the-art AutoML methods in aggregate, attaining the best-known automated performance on seven tasks. Meanwhile, on six of the ten tasks, the combined search and retraining time is less than 2x slower than simply training a CNN backbone that is far less accurate.

10.1 Related work

Most work in AutoML has focused on either small hyperparameter spaces [Bergstra and Bengio, 2012, Li et al., 2018a] or on NAS [Elsken et al., 2019a]; search spaces for the latter usually only contain a few operations such as convolutions [Liu et al., 2019b, Mei et al., 2020, Zela et al., 2020b, Dong and Yang, 2020], which may not be useful for domains where CNNs are ineffective. Applications of NAS outside vision largely follow the same pattern of combining human-designed operations [Nekrasov et al., 2019, Wang et al., 2020b]. On the other extreme, AutoML-Zero [Real et al., 2020] demonstrates the possibility of evolving all aspects of ML from scratch. We seek to establish a middle ground with large and domain-agnostic search spaces that still allow the use of well-tested methods, e.g. stochastic gradient descent (SGD).

We introduce two approaches for improving the applicability of NAS by a morphism-based approach [Wei et al., 2016, Chen et al., 2016] that adapts existing CNN backbones to target tasks. The first, XD-Operations, aims to generalize convolutions by generalizing the DFT matrices in their diagonalization. Several papers have generalized the DFT to replace layers in deep nets [Dao et al., 2019, Alizadeh vahid et al., 2020, Ailon et al., 2020, Dao et al., 2020] in order to speed up or add structure to models while *reducing* expressivity. In contrast, we can replace *convolutions* and other layers while *increasing* expressivity by extending their diagonalization via K-matrices. As discussed in Section 10.2.1, using K-matrices for this directly is inefficient for input dimension > 1.

Although this new search space includes all types of convolutions, the search process is unacceptably long even for simple benchmarking tasks such as CIFAR-100 (Figure 10.5), let alone the more complex set of diverse problems that we would like to consider, e.g. those in Tu et al. [2022]. In addition, the output architectures of XD are as inefficient as the supernet due to the absence of a discretization step. We thus move to a second approach, DASH, which specifically focuses on expanding the operation space to multiple types of convolutions, simultaneously varying the kernel size and the dilation factor. Past work in related directions has at most studied the easier problem of altering dilation alone, and only for vision tasks [Chen et al., 2018b]. Therefore, although convolution has been an integral part of NAS, how to search over a large set of convolutional operators remains an open problem. In the following, we identify three types of solutions from existing work and illustrate their limitations.

1. **Differentiable Architecture Search (DARTS):** We can treat convolutions as ordinary operators and apply a scalable NAS algorithm. In particular, DARTS [Liu et al., 2019b] introduces continuous relaxation to the weight-sharing paradigm [Pham et al., 2018] and allows us to gain information about many networks efficiently by training a combined supernet. The algorithm relaxes the discrete set of operations at each edge in a computational graph as a softmax so the search process is end-to-end differentiable and amenable to reg-

ular optimizers. After search, it discretizes the weights to output a valid architecture. The original DARTS search space contains only four convolutions with kernel sizes no larger than 5 and dilation rates no larger than 2. While this small search space might be enough for low-resolution image input, it is insufficient for diverse tasks such as high-dimensional time series problems [Tu et al., 2022]. Although one could add more convolutions one-by-one to the operator set to augment performance, this approach scales poorly, as reflected in the limited search spaces of similar methods like AMBER [Zhang et al., 2021]. In fact, AMBER has to shift the kernel size up to achieve good performance on long-sequence genomic data.

2. **MergeNAS and RepVGG:** An alternative way to explore the convolutional search space is to take advantage of the operator's linearity. That is, we can first mix the kernels and then apply convolution once, unlike DARTS which computes each convolution separately and outputs the aggregated result to the next layer. This kernel-mixing strategy, which we call *mixed-weights* and will formally define in Section 10.3.2, has been employed by MergeNAS [Wang et al., 2020a] and RepVGG [Ding et al., 2021] to improve architecture search robustness and VGG inference speed, respectively. It works well for *a few small* kernels. However, we will show later that similar to DARTS, *mixed-weights* on its own is also insufficient for searching over *a diverse set of large kernels* which is crucial to solving a wide range of problems.
3. **Single-Path NAS:** This approach defines a large filter and uses its subsets for smaller filters [Stamoulis et al., 2020]. This DARTS-based method compensates operator heterogeneity for efficiency during search. It does not handle search spaces with many large kernels and is not evaluated on diverse tasks.

Outside the field of AutoML, there is also emerging interest in designing general-purpose models such as Perceiver IO [Jaegle et al., 2022] and Frozen Pretrained Transformer [Lu et al., 2022]. However, these Transformer-based models do not adapt the network to the target tasks and are generally harder to train compared with CNNs. In Table 10.5, we evaluate Perceiver IO and show that its performance is not ideal.

10.2 XD-operations

In this section we formalize XD-operations, a large, general search space of neural operations, and design a search procedure for finding good architectures. We then evaluate this approach on multiple understudied applications.

10.2.1 The expressive diagonalization relaxation

In this section we overview our main contribution:

Formally, we view an architecture as a *parameterizable* object—a mapping from model weights to functions—described by a *labeled* directed acyclic graph (DAG) $\mathcal{G}(V, E)$. Each edge in E has the form $(u, v, \mathbf{Op})$, where $u, v \in V$ are nodes and $\mathbf{Op}$ is an operation that can be parameterized to define some transformation of the representation at node u; node v aggregates

the outputs of its incoming edges into a new representation. For example, the popular ResNet architecture [He et al., 2016] has many nodes with two incoming edges, one labeled by the convolution operation $\mathbf{Conv}$ and one by the identity (skip-connect) $\mathbf{Id}$, whose outputs it sums and passes to outgoing edges with the same labels. Each architecture has a source node taking in input data and an output node returning a prediction.

Neural architecture search is the problem of automatically selecting an operation for each edge of $\mathcal{G}$ to optimize an objective.[1] For each edge $e \in E$ a NAS algorithm must pick one element of a *search space* $\mathcal{S} = \{\mathbf{Op}_a \,|a \in \mathcal{A}\}$ of operations specified by architecture parameters $a \in \mathcal{A}$ to assign to e; in past work, $\mathcal{A}$ usually indexes a small set of operations. As an example, we will refer to a variant[2] $\mathcal{S}_{\mathbf{discrete}}$ of the DARTS search space with parameters $\mathcal{A}_{\mathbf{discrete}} = \{1, \ldots, 8\}$ where each operation is one of $\mathbf{Zero}$, $\mathbf{Id}$, $\mathbf{MaxPool}_{3\times3}$, $\mathbf{AvgPool}_{3\times3}$, $\mathbf{Conv}_{3\times3 \text{ or } 5\times5}$, or $\mathbf{DilatedConv}_{3\times3,2 \text{ or } 5\times5,2}$ [Liu et al., 2019b].

Our main contribution is a novel family of operations that comprise a search space containing almost all these operations, in addition to many others that have been found useful on different types of data. The starting point of our construction of these XD-operations is the simple observation that all the operations $\mathbf{Op} \in \mathcal{S}_{\mathbf{discrete}}$ listed above except $\mathbf{MaxPool}_{3\times3}$ are *linear*, i.e. for any model weights $\mathbf{w}$ there exists a matrix $\mathbf{A}_{\mathbf{w}}$ such that for all inputs $\mathbf{x}$ we have $\mathbf{Op}(\mathbf{w})(\mathbf{x}) = \mathbf{A}_{\mathbf{w}}\mathbf{x}$. More specifically, all seven of them return convolutions: to see this note that $\mathbf{Zero}$, $\mathbf{Id}$, and $\mathbf{AvgPool}_{3\times3}$ each apply a convolution with filter $\mathbf{0}_{1\times1}$, $\mathbf{1}_{1\times1}$, and $\mathbf{1}_{3\times3}/9$, respectively. This means that most of the operations in the DARTS search space—which is representative of NAS operation spaces in computer vision—share the convolution's diagonalization by the discrete Fourier transform (DFT). Formally, if $\mathbf{A}_{\mathbf{w}} \in \mathbb{R}^{n^2 \times n^2}$ is the matrix representing a 2D convolution with filter $\mathbf{w} \in \mathbb{R}^{\mathbf{k}}$ of kernel size $\mathbf{k} \in [n]^2$, then for any 2D input $\mathbf{x} \in \mathbb{R}^{n^2}$ we have

$$\mathbf{Conv}(\mathbf{w})(\mathbf{x}) = \mathbf{A}_{\mathbf{w}}\mathbf{x} = \mathbf{F}^{-1} \operatorname{diag}(\mathbf{F}\underline{\mathbf{w}})\, \mathbf{F}\mathbf{x} \tag{10.1}$$

Here $\operatorname{diag}(\mathbf{z})$ denotes the diagonal matrix with entries $\mathbf{z}$, $\underline{\mathbf{w}} \in \mathbb{R}^{n^2}$ is an appropriate zero-padding of $\mathbf{w} \in \mathbb{R}^{\mathbf{k}}$, and $\mathbf{F} \in \mathcal{C}^{n^2 \times n^2}$ is the 2D DFT (a Kronecker product of two 1D DFTs).

This diagonalization explicates both the computational and representational efficiency of the DARTS operations, as the DFT and its inverse can be applied in time $\mathcal{O}(n \log n)$ and stored with $\mathcal{O}(n \log n)$ bits. It also suggests a natural way to dramatically expand the operation space while preserving these efficiencies: just replace matrices $\mathbf{F}$ and $\mathbf{F}^{-1}$ in (10.1) by any one of a general family of efficient matrices. Doing so yields the single-channel version of our *expressive diagonalization* (XD) operations:

$$\mathbf{XD}^{\mathbf{1}}_{\alpha}(\mathbf{w})(\mathbf{x}) = \operatorname{Real}(\mathbf{K} \operatorname{diag}(\mathbf{L}\underline{\mathbf{w}})\, \mathbf{M}\mathbf{x}) \tag{10.2}$$

Here architecture parameter $\alpha = (\mathbf{K}, \mathbf{L}, \mathbf{M})$ specifies the matrices that replace $\mathbf{F}$ and $\mathbf{F}^{-1}$ in Equation 10.1.

The main remaining question is the family of efficient matrices to use, i.e. the domain of the architecture parameters $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$. For this we turn to the Kaleidoscope matrices, or *K-matrices* [Dao et al., 2020], which generalize $\mathbf{F}$ and $\mathbf{F}^{-1}$ to include all computationally efficient

[1] It is often defined as selecting both operations and a graph topology [Zoph et al., 2018], but if the set of operations contains the zero-operation $\mathbf{Zero}$ then the former subsumes the latter.

[2] For memory-efficiency, all convolutions in the original DARTS search space are separable [Liu et al., 2019b].

linear transforms with short description length, including important examples such as sparse matrices and permutations. To obtain this general family, K-matrices allow the DFT's butterfly factors—matrices whose products yield its efficient implementation—to take on different values. While a detailed construction of K-matrices can be found in the original paper, we need only the following useful properties: they are as (asymptotically) efficient to apply as DFTs, are differentiable and can thus be updated using gradient-based methods, and can be composed (made "deeper") to make more expressive K-matrices.

Specifying that $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$ in Equation 10.2 are K-matrices largely completes our core contribution: a new search space $\mathcal{S}_{\mathbf{XD}}$ of XD-operations with K-matrix architecture parameters. We give a full multi-channel formalization in N dimensions, as well as an overview of its expressivity, in Section 10.2.2. First, we note some key aspects of this new search space:

- **Complexity:** $\mathbf{XD}^{\mathbf{1}}_{\alpha}(\mathbf{w})$ requires three K-matrices and $\mathcal{O}(1)$ filter weights to represent, i.e. description length $\mathcal{O}(n \log n)$; this is larger than a regular convolution (which has no architecture parameters) but is not quadratic in the input size like a linear layer. Applying $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ requires multiplication by three K-matrices, yielding a theoretical per-channel time complexity of $\mathcal{O}(n \log n)$, matching the efficiency of convolutions. However, as XD-operations strictly generalize convolutions they are more expensive to apply in practice; we detail these costs both in the application sections and in Table 10.7, and we view improving upon them as an important future direction.
- **Initialization:** a crucial advantage of XD-operations is that we can initialize or *warm-start* search using operations with known constructions. In particular, since we can recover convolutions (10.1) by setting architecture parameters $\mathbf{K} = \mathbf{F}^{-1}$, $\mathbf{L} = \mathbf{F}$, and $\mathbf{M} = \mathbf{F}$ in Equation 10.2, we can always start search with any CNN backbone. We use this extensively in experiments.
- **K-matrices:** as they contain all efficient linear transforms, K-matrices can represent all functions returned by XD-operations, including convolutions. However, for input dimension and filter size > 1 the only known way is to apply K-matrices directly to flattened inputs $\mathbf{x} \in \mathbb{R}^{n^N}$, yielding much worse description length $\mathcal{O}(n^N \log n)$. In contrast, as detailed in Section 10.2.2, our diagonalization approach uses Kronecker products to apply DFTs to each dimension separately, yielding description length $\mathcal{O}(n \log n)$. It is thus the first (and in some sense, "right") method to use such matrices to replace convolutions. Furthermore, diagonalization allows us to separate model weights $\mathbf{w}$ from architecture parameters α, letting the former vary across channels while fixing the latter.

Finally, we address the fact that the architecture parameters of $\mathcal{S}_{\mathbf{XD}}$ are continuous, not discrete, contrasting with much of the NAS literature. This can be viewed as a natural extension of the weight-sharing paradigm [Pham et al., 2018], in which continuous relaxation enables updating architecture parameters with gradient methods. For example, many algorithms traverse the relaxed DARTS search space $\tilde{\mathcal{S}}_{\mathbf{discrete}} = \left\{\sum_{i=1}^{8} \lambda_i \, \mathbf{Op}_i \,|\lambda_i \geqslant 0, \sum_{i=1}^{8} \lambda_i = 1\right\}$, defined via DARTS operations $\mathbf{Op}_i \in \mathcal{S}_{\mathbf{discrete}}$ and architecture parameters λ_i in the 8-simplex; most search spaces then require discretizing after search via a rounding procedure that maps from the simplex to $\mathcal{A}_{\mathbf{discrete}}$. Note that the fully continuous nature of XD-operations means that we will only evaluate the final network returned by search. In particular, while some weight-sharing papers also report the correlation between true architecture performance and that indicated by the shared

weights [Yang et al., 2020], there is no obvious way to define a ranking or sampling distribution over XD-operations in order to do so. This also means that our final architecture will not be more efficient than the supernet, unlike other weight-sharing methods that do discretize.

10.2.2 XD-operations and their expressivity

Here we formalize XD-operations and what operations they include. We first define operations:

Definition 10.2.1. A **parameterizable operation** is a mapping $\mathbf{Op} : \mathcal{W} \mapsto \mathcal{F}$ from parameter space $\mathcal{W}$ to a space $\mathcal{F} = \{\mathbf{Op}(\mathbf{w}) : \mathcal{X} \mapsto \mathcal{Y} | \mathbf{w} \in \mathcal{W}\}$ of **parameterized functions** from input space $\mathcal{X}$ to output space $\mathcal{Y}$. A **search space** is a set of operations with the same $\mathcal{W}$, $\mathcal{X}$, and $\mathcal{Y}$.

For example, if $\mathcal{X} = \mathcal{Y} = \mathbb{R}^n$ and $\mathcal{W} = \mathbb{R}^{n \times n}$ then each $\mathbf{W} \in \mathcal{W}$ defines a parameterized linear layer that for each $\mathbf{x} \in \mathcal{X}$ returns $\mathbf{Lin}(\mathbf{W})(\mathbf{x}) = \mathbf{W}\mathbf{x}$. Here $\mathbf{Lin}$ is the parameterizable operation and for each $\mathbf{W}$ the linear map $\mathbf{Lin}(\mathbf{W})$ is the parameterized function.

From Definition 10.2.1, we say a search space can *express* a specific operation if it contains it. Crucially, the ability of a parameterizable operation $\mathbf{Op}_1$ to express a parameterized function $\mathbf{Op}_2(\mathbf{w})$ output from another operation $\mathbf{Op}_2$ given the right set of weights $\mathbf{w}$ does *not* imply that a search space containing $\mathbf{Op}_1$ can express $\mathbf{Op}_2$. For example, $\mathbf{Lin}(\mathbf{I}_n) = \mathbf{Id}(\mathbf{W}) \; \forall \; \mathbf{W} \in \mathbb{R}^{n \times n}$ but $\mathbf{Lin}(\mathbf{W}) \neq \mathbf{Id}(\mathbf{W}) \; \forall \; \mathbf{W} \neq \mathbf{I}_n$, so a search space containing the linear operation $\mathbf{Lin}$ cannot express the skip-connection $\mathbf{Id}$, despite the fact that the operation $\mathbf{Lin}$ can itself be parameterized by a specific $\mathbf{W}$ to compute the identity.

Formalizing multi-channel XD-operations

Recall the single-channel XD-operation $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ in Equation 10.2 specified by three-matrix architecture parameter $\alpha = (\mathbf{K}, \mathbf{L}, \mathbf{M})$. For input dimension $N \geqslant 1$, every matrix $\mathbf{B} \in \alpha$ is a Kronecker product of N K-matrices of depth $\mathbf{d} \in \mathbb{Z}^3_{>0}$, i.e. $\mathbf{B} = \bigotimes_{i=1}^{N} \mathbf{B}_i$ for K-matrices $\mathbf{B}_i \in \mathcal{C}^{n \times n}$ of depth $\mathbf{d}_{[1]}$, $\mathbf{d}_{[2]}$, or $\mathbf{d}_{[3]}$ for $\mathbf{B} = \mathbf{K}$, $\mathbf{L}$, or $\mathbf{M}$, respectively.[3] Roughly speaking, $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ can return any linear operation that is diagonalized by K-matrices and is thus efficient to compute and represent, e.g. any convolution (recall we recover the diagonalization of $\mathbf{Conv}(\mathbf{w})$ in Equation 10.1 by setting $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$ appropriately in Equation 10.2). However, $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ cannot represent efficient *parameter-free* operations such as skip-connections and average-pooling, both common in NAS. In particular, the only way to always ignore the model weights $\mathbf{w}$ is to set one of the K-matrices to zero, producing the zero-operation. We avoid this by adding a bias $\mathbf{b} \in \mathcal{C}^{n^N}$ as an architecture parameter, yielding the *biased* single-channel XD-operation:[4]

$$\mathbf{XD}^{\mathbf{1}}_{\alpha,\mathbf{b}}(\mathbf{w})(\mathbf{x}) = \mathrm{Real}\left(\mathbf{K}\,\mathrm{diag}(\mathbf{L}\underline{\mathbf{w}} + \mathbf{b})\mathbf{M}\underline{\mathbf{x}}\right) \tag{10.3}$$

This lets us define skip-connections ($\mathbf{K} = \mathbf{M} = \mathbf{I}_{n^N}$, $\mathbf{L} = \mathbf{0}_{n^N \times n^N}$, and $\mathbf{b} = \mathbf{1}_{n^N}$) and average-pooling ($\mathbf{K} = \mathbf{F}^{-1}$, $\mathbf{L} = \mathbf{0}_{n^N \times n^N}$, $\mathbf{M} = \mathbf{F}$, and $\mathbf{b}$ is $\mathbf{F}$ multiplied by a pooling filter).

Lastly, we use $\mathbf{XD}^{\mathbf{1}}_{\alpha,\mathbf{b}}$ to construct multi-channel "layers" that pass input features through multiple channels and re-combine them as multiple output features. This follows the primary

[3] A depth-d K-matrix is a product of d depth-1 K-matrices.

[4] Zero-padding $\mathbf{x}$ as well lets the input to be smaller than the output if needed, e.g. for transposed convolutions.

way of using convolutions in deep nets. The key insight here is that we will share the same parameterizable operation (specified by α and $\mathbf{b}$) across all channels, just as in convolutional layers.

Definition 10.2.2. Let $a = (\alpha, \mathbf{b}, \mathbf{C})$ be an architecture parameter containing a triple $\alpha = (\mathbf{K}, \mathbf{L}, \mathbf{M})$ of Kronecker products of N K-matrices with depths $\mathbf{d} \in \mathbb{Z}^3_{>0}$, a bias $\mathbf{b} \in \mathcal{C}^{n^N}$, and channel gates $\mathbf{C} \in \mathcal{C}^{c \times c}$.[5] Using "$\bigoplus$" to denote concatenation, the **XD-operation** $\mathbf{XD}_a$ of depth $\mathbf{d}$ specified by a is a parameterizable operation on parameter space $\mathcal{W} = \mathbb{R}^{c \times c \times \mathbf{k}}$ consisting of c^2 filters of size $\mathbf{k} \in [n]^N$ that outputs parameterized functions on $\mathcal{X} = \mathbb{R}^{c \times m^N}$ for $m \leqslant n$ mapping every $\mathbf{x} \in \mathcal{X}$ to

$$\mathbf{XD}_a(\mathbf{w})(\mathbf{x}) = \bigoplus_{i=1}^{c} \sum_{j=1}^{c} \mathbf{C}_{[i,j]} \mathbf{XD}^{\mathbf{1}}_{\alpha,\mathbf{b}}(\mathbf{w}_{[i,j]})(\mathbf{x}_{[j]}) \tag{10.4}$$

The last architecture parameter $\mathbf{C}$ allows interpolation between all-to-all layers ($\mathbf{C} = \mathbf{1}_{c \times c}$), e.g. multi-channel convolutions, and layers where each channel is connected to one other channel ($\mathbf{C} = \mathbf{I}_c$), e.g. skip-connections and average-pooling. We note that we use $\mathcal{S}_{\mathbf{XD}}$ to describe the set of operations covered by Definition 10.2.2 and conclude our construction by discussing two properties:

- **Kernel size:** the weight-space available to an XD-operation is $\mathbb{R}^{c \times c \times n^N}$; however, since we will initialize search with existing CNNs, we will zero-pad to have the same weight-space $\mathbb{R}^{c \times c \times k^N}$ as the convolutions with filter size $k \leqslant n$ that they replace. This preserves the weight count but also means that if the backbone has 3×3 filters our search space will *not* contain 5×5 convolutions. Experimentally, we find that relaxing the constraint to allow this does not significantly affect results on image tasks, so we do not do so in subsequent applications to avoid increasing the weight count.
- **Depth:** an XD-operation's depth is a triple describing the depths of its K-matrices $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$. Increasing it trades off efficiency for expressivity; for example, in the next section we describe operations that we can show are contained in $\mathcal{S}_{\mathbf{XD}}$ if $\mathbf{L}$ or $\mathbf{M}$ have depth > 1. By default we will set the depth to be the minimum needed to initialize search with the backbone operation.

Expressivity of XD-operations

For many papers that replace deep net layers with efficient linear transforms [Moczulski et al., 2015, Dao et al., 2020], the question of expressivity comes down to the transform capacity. For example, layers with a K-matrix in every channel can represent a different transform in each, thus allowing the output to be any combination of efficient linear operations. Our case is less straightforward since we care about expressivity of the search space, not of parameterized functions, and our approach is less-expressive *by design* as all channels share K-matrices $\mathbf{K}$, $\mathbf{L}$, and $\mathbf{M}$. The latter can be thought of as a useful inductive bias on NAS: the set of XD-operations is still much broader than the set of convolutions, but the way in which model weights are applied is the same across all channels.

[5] For simplicity we formalize the case where all N dimensions have the same input size and there is an identical number c of input and output channels; both are straightforward to extend.

Expressivity results are a way to see if this bias is useful or constraining. Here we summarize some important operations that are 1D XD-operations; proofs can be found in Appendix 10.A.1 and are straightforward to extend to multi-dimensional inputs. Formally, there exists $\mathbf{d} \in \mathbb{Z}^3_{>0}$ such that the set of XD-operations of depth $\mathbf{d}$ over weights $\mathcal{W} = \mathbb{R}^{c \times c \times k}$ and inputs $\mathcal{X} = \mathbb{R}^m$ for $m \leqslant n$ contains

1. convolutions with filter size $\leqslant k$, dilation rate $\leqslant \lfloor \frac{n-1}{k-1} \rfloor$, stride length $\leqslant n-1$, and arbitrary channel groups.
2. parameter-free operations $\mathbf{Id}$, $\mathbf{Zero}$, and $\mathbf{AvgPool}_s$ for any kernel size $s \leqslant n$.
3. composing 1 or 2 operations with multiplication of all input or output channels by a bounded-depth K-matrix.

Note this does not account for *all* important XD-operations, e.g. we show in Appendix 10.A.1 that they also express Fourier Neural Operators [Li et al., 2021c] with $\leqslant \lfloor k/2 \rfloor$ modes and any transposed convolutions whose stride equals the dilated kernel size.[6] Still, the first two items account for non-separable variants of most operations considered in past NAS work in computer vision, excluding the nonlinear $\mathbf{MaxPool}$ [Ying et al., 2019, Dong and Yang, 2020]. Note depthwise-separable convolutions *are* contained in the set of compositions of XD-operations. The third item implies that XD-operations can express the basic and diffusion graph convolutions over fixed graphs [Kipf and Welling, 2017, Li et al., 2018b]: both are point-wise convolutions composed with sparse multiplication by a modified adjacency matrix, which K-matrices can represent efficiently.

As a concrete example, consider dilated convolutions, which for $k > 1$ and dilation factor $d \geqslant 1$ apply filters of effective size $(k-1)d+1$ with nonzero entries separated by $d-1$ zeros. One could hope to express the application of $\mathbf{DilatedConv}_{k,d}$ to an input $\mathbf{x} \in \mathbb{R}^n$ in the single-channel setting as $\mathbf{F}^{-1} \operatorname{diag}(\mathbf{F} \operatorname{diag}(\mathbf{p}_{k,d}) \underline{\mathbf{w}}) \mathbf{F}\mathbf{x}$, where $\mathbf{p}_{k,d} \in \{0,1\}^n$ zeroes out appropriate entries of $\underline{\mathbf{w}}$, but this requires filter size $(k-1)d+1 > k$, increasing the number of weights. Instead, we can use a permutation $\mathbf{P}_{k,d} \in \{0,1\}^{n \times n}$ before the DFT to place the k entries of $\underline{\mathbf{w}}$ into dilated positions:

$$\mathbf{DilatedConv}_{k,d}(\mathbf{w})(\mathbf{x}) = \mathbf{F}^{-1} \operatorname{diag}(\mathbf{F}\mathbf{P}_{k,d}\underline{\mathbf{w}}) \mathbf{F}\mathbf{x} \tag{10.5}$$

As permutations are depth-2 K-matrices [Dao et al., 2020], we can express $\mathbf{DilatedConv}_{k,d}$ with an XD-operation of depth $(1,3,1)$, with $\mathbf{K} = \mathbf{F}^{-1}$, $\mathbf{L} = \mathbf{F}\mathbf{P}_{k,d}$, and $\mathbf{M} = \mathbf{F}$.

10.2.3 Finding and evaluating XD-operations

This section outlines a simple procedure that we use to evaluate XD-operations. Recall that NAS methods specify architectures by assigning operations to each edge $(u, v, \mathbf{Op})$ of a computational graph. We aim to simultaneously find good operations and model weights, a goal distinct from the classic *two-stage* NAS formulation, which finds assignments in an initial search phase before training the resulting architecture from scratch [Ying et al., 2019]. However, the use of weight-sharing [Pham et al., 2018] extends NAS to *one-shot* objectives where weights and architectures

[6] This includes those transposed convolutions used in popular architectures such as U-Net [Ronneberger et al., 2015].

are jointly optimized. Under weight-sharing, architecture parameters become weights in a larger "supernet," extending the hypothesis class (c.f. Chapter 9).

To assess XD-operations directly we assume the user provides a starter network with existing edge labels $\mathbf{Op}_{u,v}$ as a backbone. We transform this into a weight-sharing supernet by reparameterizing each operation $\mathbf{Op}_{u,v}$ as an XD-operation $\mathbf{XD}_{a_{u,v}}$ with architecture parameter $a_{u,v}$. Then we simultaneously train both $a_{u,v}$ and the model weights $\mathbf{w}_{u,v}$ associated with each edge as follows:

- **Architecture parameters** $a_{u,v}$ are initialized using the original operation used by the CNN backbone by setting $\mathbf{Op}_{u,v} = \mathbf{XD}_{a_{u,v}}$; $a_{u,v}$ is then updated via SGD or Adam [Kingma and Ba, 2015]. We tune step-size, momentum, and the number of "warmup" epochs: initial epochs during which only model weights $\mathbf{w}_{u,v}$ are updated. This can be viewed as a specialized step-size schedule.
- **Model weights** $\mathbf{w}_{u,v}$ are initialized and updated using the backbone's original routine.

This approach allows us to use established topologies and optimizers while searching for new operations, thus aligning with the goal for Sections 10.2.4, 10.2.4, and 10.2.4: to improve upon the CNN backbones that practitioners often use as a first attempt. As a simple example, we start by applying the procedure to image classification. Since this is not the main objective of our work, we treat it as a warmup and consider two datasets: CIFAR-10 and a variant where the images' rows and columns are permuted. On CIFAR-10 we do *not* expect to see much improvement from XD-operations over the CNN backbone used to initialize search, as convolutions are already the "right" operation for images. On the other hand, the "right" operation on permuted data, at least in layer one, is an inverse permutation followed by convolution; as this is an XD-operation[7], here we do hope to see improvement.

Using LeNet [LeCun et al., 1999] and ResNet-20 [He et al., 2016] as backbones, we compare applying our algorithm to XD-operations with two baselines: (1) using just the backbone CNN and (2) applying a similar method to the relaxed set $\tilde{\mathcal{S}}_{\mathbf{discrete}}$ of DARTS operations from Section 10.2.1. To optimize over $\tilde{\mathcal{S}}_{\mathbf{discrete}}$ we take an approach similar to DARTS: parameterize the simplex using a softmax and apply Adam. We experiment with both a uniform initialization and one biased towards the backbone's operation. While both $\mathcal{S}_{\mathbf{XD}}$ and $\mathcal{S}_{\mathbf{discrete}}$ contain LeNet's $\mathbf{Conv}_{5\times5}$ and ResNet's $\mathbf{Conv}_{3\times3}$ and $\mathbf{Id}$, for LeNet's $\mathbf{MaxPool}_{3\times3}$ layer we initialize with the closest operation. For direct comparison, both search spaces employ weights with maximum filter size 5×5 and for both we evaluate the shared weights rather than retraining, which we find hurts $\tilde{\mathcal{S}}_{\mathbf{discrete}}$. We set the XD-operations' depth to $\mathbf{d} = \mathbf{3}_3$ to express the dilated convolutions in $\mathcal{S}_{\mathbf{discrete}}$ and convolutions composed with permutations.

In Table 10.1, we see that while both the relaxed discrete NAS operations and XD-operations perform comparably on regular images, XD-operations achieve around 15% better accuracy with both backbones when the images are permuted.[8] Note that even networks obtained by running state-of-the-art NAS procedures such as GAEA PC-DARTS (c.f. Section 9.2.2) and DenseNAS [Fang et al., 2020] on permuted CIFAR-10 achieve only 66.3% and 61.6% accuracy, respectively, despite using millions more parameters than ResNet-20. While it is not straightforward to understand the recovered XD-operations that perform so well, we can use the relative

[7]Recall $\mathcal{S}_{\mathbf{XD}}$ includes compositions of convolutions with multiplication by a K-matrix, e.g. a permutation.

[8]Full accuracy can be recovered via an auxiliary loss encouraging permutation-like K-matrices [Dao et al., 2020].

Table 10.1: Search space comparison on CIFAR-10. Validation accuracies are averages of three trials. While we use small CNNs for exploration, XD-operations can also be used with high-performance backbones to obtain $> 95\%$ accuracy (c.f. Appendix 10.C.1).

Backbone search space	CIFAR-10	Permuted CIFAR-10*	Cost (hours†)
LeNet	75.5 ± 0.1	43.7 ± 0.5	0.3
$\tilde{\mathcal{S}}_{\mathbf{discrete}}$	75.6 ± 3.4	47.7 ± 1.0	1.0
$\mathcal{S}_{\mathbf{XD}}$	77.7 ± 0.7	63.0 ± 1.0	0.9
ResNet-20	91.7 ± 0.2	58.6 ± 0.7	0.6
$\tilde{\mathcal{S}}_{\mathbf{discrete}}$	92.7 ± 0.2	58.0 ± 1.0	5.3
$\mathcal{S}_{\mathbf{XD}}$	92.4 ± 0.2	73.5 ± 1.6	5.6

* No data augmentation used in the permuted case.

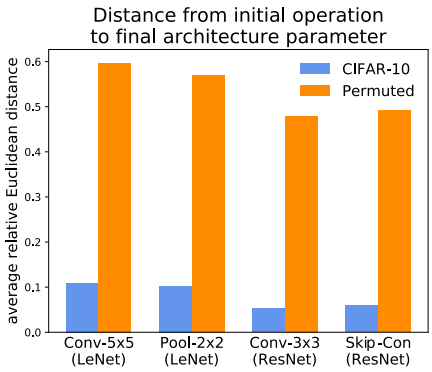

Figure 10.1: On permuted images, where convolutions are not the "right" operation, we find XD-operations that are farther away from the operations of the initial CNN backbone.

Euclidean distance of their architecture parameters from initialization as a proxy for novelty; in Figure 10.1 we see that on regular images our procedure finds operations that are quite similar to convolutions, but on permuted data they are much further away. These results show that to enable NAS on diverse data, we will need a search space that contains truly novel operations, not just combinations of existing ones. In the remainder of this section we study more diverse and realistic tasks that show further evidence that $\mathcal{S}_{\mathbf{XD}}$ is a strong candidate for this.

10.2.4 Diverse applications

In selecting applications to consider beyond vision, we focused on domains with structured data that is distinct from natural images or text, with an emphasis on scientific topics; of course, availability of data and competitive hand-designed architectures as baselines was also a consideration. After our initial experiments with XD, Tu et al. [2022] released NAS-Bench-360, a ten-task benchmark that significantly simplifies the types of evaluations we are interested in performing; as a result, our experiments in the next section focus on that benchmark. However, for now we turn to the three domains we chose for XD.

Learning to solve PDEs

As our first non-vision application, we consider the task of solving PDEs, an important application area of ML in the natural sciences [Li et al., 2015, 2018c, Sirignano and Spiliopoulos, 2018]. In our setup, data generated by classical PDE solvers is used to learn functions from some initial condition or setting to the corresponding PDE solution, with the goal of replacing the solver by a deep net forward pass; the latter can be orders of magnitude faster. A recent state-of-the-art approach for this introduces Fourier Neural Operators [Li et al., 2021c], operations that

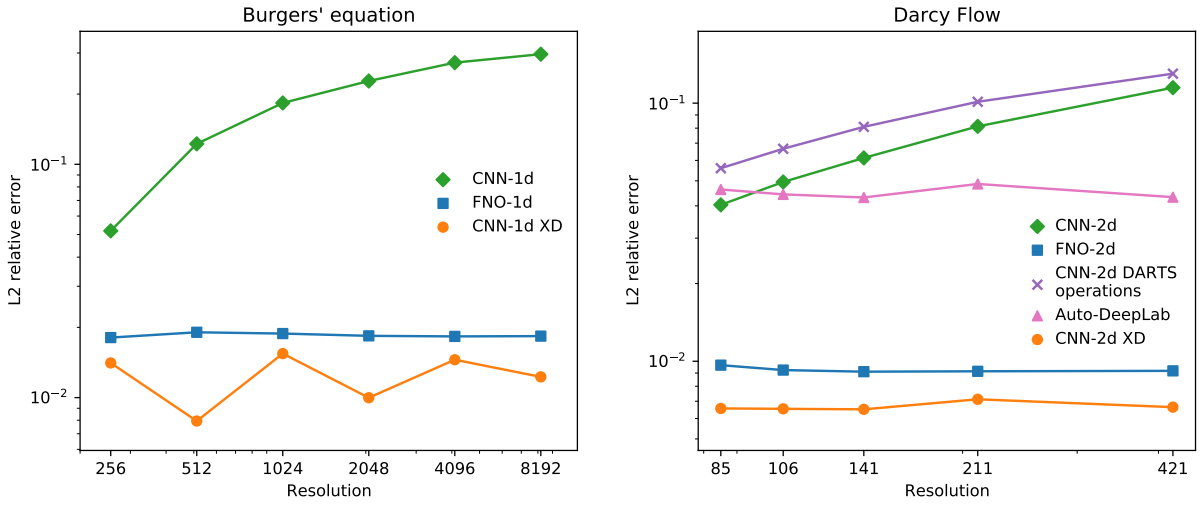

Figure 10.2: Relative error on Burgers' equation (left) and Darcy Flow (right).

significantly improve upon previous neural approaches across three different PDE settings. To evaluate the ability of XD-operations to compete with such custom-designed operations starting from simple CNN backbones, we will investigate the same three PDEs that they study: Burgers' equation, Darcy Flow, and the 2D Navier-Stokes equations, which involve 1D, 2D, and 3D data, respectively. The first two are studied across multiple resolutions, while the last one is studied at different viscosities.

As before, we start with a simple CNN backbone—the type a scientist might use in a first attempt at a solution—and replace all convolutions by XD-operations. We initially hope to do better than this backbone, but ambitiously also hope to compete with the custom-designed FNO. The specific CNN we use is simply the FNO architecture of the appropriate dimension N but with all N-dimensional FNOs replaced by N-dimensional convolutions; this performs similarly to their CNN baselines [Li et al., 2021c]. In all cases we compare mainly to the CNN backbone and our reproduction of the FNO results, as the latter exceeds all other neural methods; a complete results table is provided in Appendix 10.C.2. Our reproduction of FNO is slightly worse than their reported numbers for Burgers' equation and slightly better in the other two settings. Note that on the Navier-Stokes equations we only compare to the 3D FNO on the two settings in which we were able to reproduce their approach; moreover, we do *not* compare to their use of a 2D FNO plus a recurrent net in time, but in-principle XD-operations can also be substituted there. In the 2D Darcy Flow case we also include comparisons to DARTS operations in the simple CNN backbone, as in Section 10.2.3, and to Auto-DeepLab (AutoDL) [Liu et al., 2019a], a well-known NAS method for dense prediction. For evaluating XD-operations we again follow the procedure in Section 10.2.3, in which we tune only the architecture optimizer; notably, we do this only at the lowest resolutions. At all dimensions we use XD-operations of depth $\mathbf{d} = \mathbf{1}_3$; in addition, in dimensions $N > 1$ we fix the architecture biases $\mathbf{b}$ and channel gates $\mathbf{C}$ to $\mathbf{0}$ and $\mathbf{1}$, respectively, to conserve memory at higher resolutions. At lower ones we find that the performance difference is negligible.

Table 10.2: Relative test error on the 2D Navier-Stokes equations at different settings of the viscosity ν and time steps T. Best results in each setting are **bolded**.

	$\nu = 10^{-4}, T = 30$	$\nu = 10^{-5}, T = 20$
CNN-3d (our baseline)	0.325	0.278
FNO-3d (reproduced)	0.182	0.177
CNN-3d XD (ours)	**0.172**	**0.168**

We report our results for the Burger's equation and Darcy Flow in Figure 10.2; for 2D Navier-Stokes the results are in Table 10.2. In all cases we dramatically outperform the CNN backbone used to initialize XD-operations; furthermore, we also achieve better error than FNO, despite it being custom-made for this problem. In particular, we find that XD-operations have higher *training error* but generalize better (c.f. Figure 10.10). Figure 10.2 also shows that XD-operations perform consistently well across resolutions, a major advantage of FNOs over previous methods, whose performance was tightly coupled to the discretization [Li et al., 2021c]. Notably, CNN performance worsens with higher resolution, unlike that of XD and FNO. Finally, we also substantially outperform DARTS operations and AutoDL in 2D, although the latter is at least consistent across resolutions. These results provide strong evidence that XD-operations are a useful search space for discovering neural operations, even in domains where the convolutions used to initialize them perform much worse than state-of-the-art. Note that these results do come at a cost of slower training and inference: XD-operations are roughly an order of magnitude slower than FNOs, despite having fewer parameters in 2D and 3D. This still yields solvers one-to-two orders of magnitude faster than classical solvers, maintaining usefulness for the problem.

Real-valued distance prediction for protein folding

As a second scientific application, we consider the task of inferring the 3D "folded" structure of a polypeptide chain, which yields important insights into the function of the resulting protein [Adhikari, 2020]. This problem is a high-priority challenge in biology and has recently seen significant ML-driven advances from deep learning methods such as AlphaFold [Senior et al., 2020, Jumper et al., 2021] and PDNET [Adhikari, 2020]. These typically involve training a network to predict pairwise physical distances between residues in the chain. We work with the PDNET benchmark, which consists of a training set of 3,356 proteins, a validation set of 100 of proteins, and the PSICOV [Adhikari, 2020] test set of 150 proteins. PDNET is designed to be more accessible than datasets used by large-scale methods such as AlphaFold, which are not always publicly available and/or require massive compute [Senior et al., 2020, Jumper et al., 2021]. We follow the PDNET training procedure [Adhikari, 2020] and evaluate test set performance using their MAE_8 metric for assessing long-range distances.

As before we start with simple CNN backbones—in this case ResNets. We choose this to compare most directly to the custom-designed architecture used by PDNET, consisting of a Dilated ResNet characterized by its use of a cyclically increasing dilation rate across ResNet blocks [Adhikari, 2020]. At a sufficient depth, the Dilated ResNet is shown to outperform a

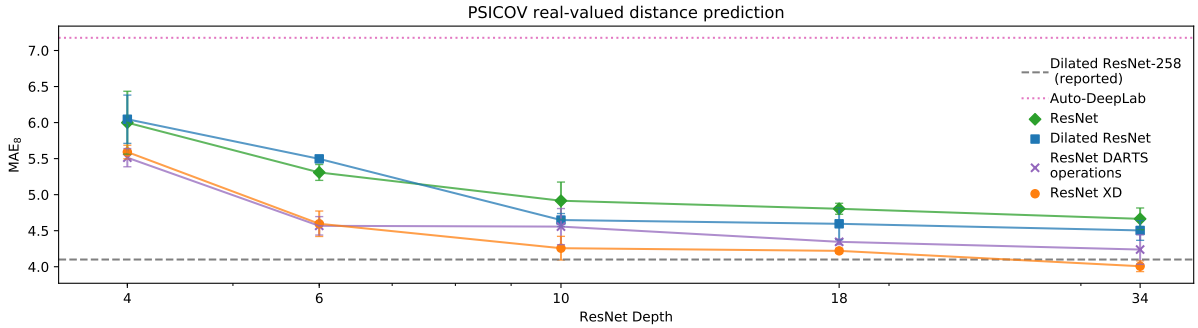

Figure 10.3: ResNet XD outperforms both baseline and dilated ResNets on PSICOV. At the highest depth we also obtain a better MAE_8 than the one reported for the much deeper Dilated ResNet-258 CNN [Adhikari, 2019].

standard pre-activation ResNet adapted to this task [Adhikari, 2020]. Our goal will be to see whether we can start with the vanilla ResNet and use XD to outperform both it and the specialized Dilated ResNet. We also aim to outperform the DARTS operations baseline from the previous two sections as well as the AutoDL NAS approach for dense prediction. We use XD-operations of depth $\mathbf{d} = \mathbf{1}_3$ and fix the architecture biases and channel gates as before to conserve memory. We evaluate architectures of different depths—4, 6, 10, 18, and 34—by varying the number of ResNet blocks used in the backbone architecture and baseline.

We report the results as averages across three trials for each depth in Figure 10.3. Notably, while Dilated ResNet slightly outperforms ResNet, ResNet XD outperforms both dilated and standard ResNets at all depths. This provides further evidence that XD-operations can outperform specialized operations for diverse domains, even when initialized naively as standard convolutions. XD also outperforms AutoDL, which does poorly, and DARTS operations, except at the two smaller depths where performance is similar. Moreover, our ResNet-34 XD's MAE_8 of 4.0 also improves upon PDNET's reported MAE_8 of 4.1 attained by the much deeper Dilated ResNet-258 [Adhikari, 2020]; however, in our reproduction Dilated ResNet-258 achieved an MAE_8 of 3.5. Given the trend in Figure 10.3, where XD-operations consistently improve the backbone architecture of the same depth, we conjecture that ResNet-258 XD could further improve upon this result. We leave scaling XD-operations to such deeper networks to future work.

Music modeling

Our last application is music modeling: learning to predict the next note from sheet music [Allan and Williams, 2005]. The dominant approaches here are recurrent nets [Hochreiter and Schmidhuber, 1997] and Transformers [Vaswani et al., 2017], but recent work shows that specially-designed CNNs are also competitive [Bai et al., 2018, 2019]. We will consider the temporal convolutional network (TCN) [Bai et al., 2018], which improves upon a regular CNN by having the dilations grow exponentially across layers. The tasks we study are on the JSB Chorales and Nottingham corpora, used in the original evaluation of TCNs [Bai et al., 2018]. As the baseline we take the TCN and set all dilation factors to one (undilated); our goal will be to start with this undilated network and match or outperform the custom dilation design of the TCN.

Table 10.3: XD-operations compared to recent results in music modeling. We report average loss across three trials. The best result on each task is **bolded**.

Method (source)	JSB Chorales	Nottingham
Best recurrent [Bai et al., 2018]	8.43	3.29
TCN [Bai et al., 2018]	8.10	3.07
Transformer [Wang et al., 2020c]	-	3.34
R-Transformer [Wang et al., 2020c]	-	**2.37**
Undilated TCN (our baseline)	8.16 ± 0.04	3.23 ± 0.02
TCN (reproduced)	8.17 ± 0.01	2.97 ± 0.01
Undilated TCN XD (ours)	$\mathbf{8.07 \pm 0.01}$	2.84 ± 0.02

The results presented in Table 10.3 show that we achieve this goal, as we outperform both the undilated baseline and the TCN on both tasks. While the simple undilated backbone that we initialize with turns out to already match the TCN on JSB Chorales, on Nottingham our approach demonstrates that XD-operations can be used to outperform hand-designed architectures starting from vanilla CNNs.[9] Where possible we also compare to other known results; XD-operations outperforms all of these except the R-Transformer [Wang et al., 2020c], a model combining recurrent nets and self-attention, on Nottingham.

Together with our results on PDEs and proteins, our study of music modeling provides further evidence that XD-operations can effectively find good operations using standard backbones on diverse tasks. One notable difficulty here is causality enforcement: making sure the input data does not contain the target when predicting the next entry. While TCNs can efficiently do so via temporal shifts, we do it in a brute-force manner by treating sequences of length n as $n-1$ data-points with masked targets. This is expensive and thus limits our evaluation to small music tasks. A fruitful direction for future work is thus to examine whether it is possibly to directly enforce causality in XD-operations, e.g. by forcing architecture parameters $\mathbf{K}$ and $\mathbf{M}$ to be lower triangular; since a product of lower triangular matrices is again lower triangular, the entire operation is then a multiplication of the input sequence by a lower triangular matrix, which suffices to prevent causality violations.

10.2.5 Conclusion

This chapter aims to transition NAS from combining existing operations designed for vision and text to finding novel and effective operations in many domains. Our first approach introduced a new search space of XD-operations and demonstrated its effectiveness on diverse tasks. Combining XD-operations with standard topology-search NAS, warm-starting search from non-standard

[9] In Appendix 10.C.4 we report similar improvements on two other tasks on which TCNs were evaluated— permuted MNIST and Penn TreeBank—that we do not discuss in detail as our focus is on under-explored tasks.

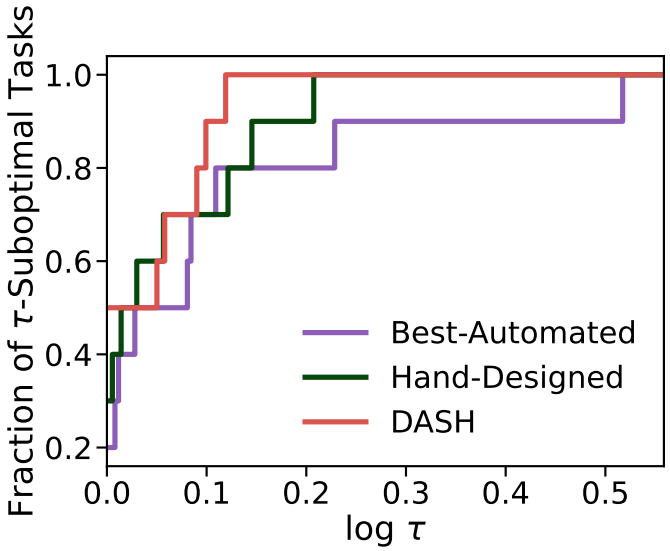

Figure 10.4: Comparing the aggregate performance of the best AutoML methods (task-wise), hand-designed models, and DASH on ten diverse tasks via performance profiles (defined in Section 10.3.4); larger values (larger fractions of tasks on which a method is within τ-factor of the best) are better.

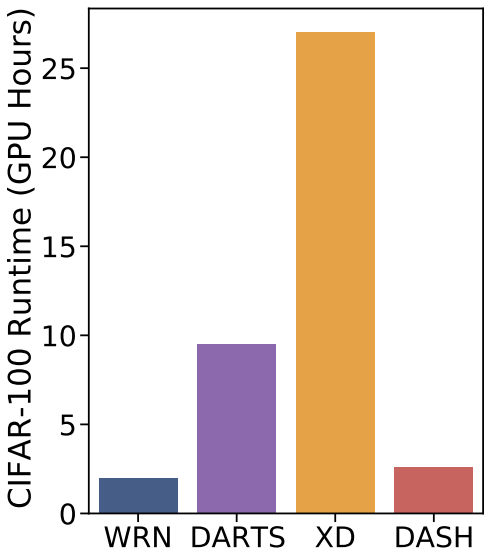

Figure 10.5: Runtime for Wide ResNet, DARTS, XD, and DASH on CIFAR-100; XD is too expensive to be applied to other tasks considered in this section [Tu et al., 2022].

operations such as graph convolutions and FNOs,[10] resolving the computational limitations described earlier, and constructing spaces containing missing operations such as BatchNorm [Ioffe and Szegedy, 2015] and self-attention [Vaswani et al., 2017] are all promising future directions.

10.3 DASH: Efficient architecture search for diverse tasks

XD improves the applicability of AutoML methods by relaxing inductive biases encoded in standard search spaces, but it is too expensive to execute even for simple problems like CIFAR-100 (Figure 10.5). Is there an approach that can provide sufficient expressivity to yield high accuracy across multiple domains, while still retaining the faster search and the more efficient final models of discrete architecture search? In this section, we show that an approach called DASH, which simply searches for the right kernel sizes and dilation rates to use in an existing CNN backbone, can successfully do so on a diverse array of tasks.

In the sequel, we first explain how DASH—detailed in Algorithm 27—leverages existing networks to initialize the supernet and generate different models for diverse tasks. Then, we formally define the multi-scale convolution search space and propose a fast way to search this space

[10] In this direction, we found that initializing XD with FNO did *worse* than initializing with convolutions on Burgers' equation and Darcy Flow, a surprising result given how much better FNO is than the baseline CNN. Similarly, initializing XD with convolutions dilated as in the original TCN did not lead to significant improvement, except in one setting, over undilated initialization. See Appendix 10.C for more details and results.

Algorithm 27: DASH

Input: training data Z, loss function l, the set of kernel sizes K, the set of dilation rates D, and subsampling ratio p

Initialize the backbone and replace each $\mathbf{Conv}$ layer with the mixed operation $\mathbf{AggConv}_{K,D}$

while *not converged* **do**

- Subsample $p|Z|$ training points uniformly at random
- Compute forward pass using Equation 10.9
- Descend the architecture parameters α by $\nabla_\alpha l(\mathbf{w}, \alpha)$ and the model weights $\mathbf{w}$ by $\nabla_{\mathbf{w}} l(\mathbf{w}, \alpha)$

Select $\arg\max_{k \in K, d \in D} \alpha_{k,d}$ for each $\mathbf{AggConv}$ layer

Tune retraining hyperparameters on a validation subset of the training data

Retrain the discretized model with all training data

using the three efficiency-motivated techniques mentioned earlier. Next, we outline the procedure for discretizing the search space and retraining the searched model. Lastly, we evaluate DASH on a benchmark of ten diverse tasks spanning multiple input and output dimensionalities.

10.3.1 Decoupling topology and operations

Every architecture is a mapping from model weights to functions and can be described by a directed acyclic graph $G(V, E)$. Each edge in E is characterized by $(u, v, \mathbf{Op})$, where $u, v \in V$ are nodes and $\mathbf{Op}$ is an operation applied to u. Node v aggregates the outputs of its incoming edges. NAS aims to automatically select the edge operations and the graph topology to optimize some objective. For each edge, $\mathbf{Op}$ is chosen from a search space $S = \{\mathbf{Op}_a \,|a \in A\}$ where $a \in A$ are architecture parameters. In past work, A usually indexes a small set of operations. For instance, the DARTS search space specifies $A_{discrete} = \{1, \ldots, 8\}$ with $S = \{\mathbf{Zero}, \mathbf{Id}, \mathbf{MaxPool}_{3\times3}, \mathbf{AvgPool}_{3\times3}, \mathbf{Conv}_{3\times3}, \mathbf{Conv}_{5\times5}, \mathbf{DilatedConv}_{3\times3,2}, \mathbf{DilatedConv}_{5\times5,2}\}$. However, A can also be defined systematically to identify operator properties, e.g. $\{$(kernel size k, dilation d)$\}$ for convolutions.

A common way to determine the network topology is to search for blocks of operations and stack several blocks together. As in the rest of this chapter, we take a different, morphism-based approach [Wei et al., 2016, Chen et al., 2016]: we use existing networks as backbones and replace certain layers in the backbone with the searched operations. Specifically, we select a set of architectures to accommodate both 2D and 1D datasets. Convolutional layers with different kernels can then be plugged into these networks. An advantage of decoupling topology and operation search is flexibility: the searched operators can vary from the beginning to the end of a network, so features at different granularities can be processed differently.

We pick Wide ResNets (WRNs) [Zagoruyko and Komodakis, 2016, Ismail Fawaz et al., 2020] as the backbone networks due to their simplicity and effectiveness in image and sequence modeling. Before search, the supernet is initialized to the backbone. Then, all $\mathbf{Conv}$ layers are substituted with an operator $\mathbf{AggConv}_{K,D}$ (short for aggregated convolution) that represents the new search space which we now define. For simplicity, our mathematical discussion will stick to the 1D case, though our experiments are on both 1D and 2D data.

10.3.2 Efficiently searching for multi-scale convolutions

A convolution filter is specified by the kernel size k and the dilation rate d (we do not consider stride which does not change the filter shape). The effective filter size is $(k-1)d+1$ with nonzero entries separated by $d-1$ zeros. Let $\mathbf{Conv}_{k,d}$ be the convolution with kernel size k, dilation rate d, c_{in} input channels, and c_{out} output channels. Given input data with shape n, let K be our interested set of kernel sizes, D the set of dilations. We define the $\mathbf{AggConv}_{K,D}$ search space as

$$S_{\mathbf{AggConv}_{K,D}} = \{\mathbf{Conv}_{k,d} \,|k \in K, d \in D\}. \tag{10.6}$$

Hence, $A = K \times D$ in previous notations. $S_{\mathbf{AggConv}_{K,D}}$ contains a collection of convolutions with receptive field size ranging from k_{min} to $d_{\text{max}}(k_{\text{max}}-1)+1$, which allows us to discover models that process the input with varying resolution at each layer.

To retain the efficiency of discrete NAS, we apply the continuous relaxation scheme of DARTS to $S_{\mathbf{AggConv}_{K,D}}$, which mixes all operations in the space using architecture parameters $\{\alpha_{k,d} \in \triangle_{|K||D|} | (k,d) \in K \times D\}$[11], so the output of each edge in the computational graph is

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) = \sum_{k \in K} \sum_{d \in D} \alpha_{k,d} \cdot \mathbf{Conv}(\mathbf{w}_{k,d})(\mathbf{x}). \tag{10.7}$$

Here $\{\mathbf{w}_{k,d} | (k,d) \in K \times D\}$ are the kernel weights. The resulting supernet can be trained end-to-end, and our hope is that after search, the most important operation is assigned the highest weight. However, the complexity of computing the above summation directly, a baseline algorithmic approach we call ***mixed-results***, is $\mathcal{O}(c_{in}c_{out}(|K||D| + \bar{K})n)$, where $\bar{K} = |D| \sum_{k \in K} k$. *Mixed-results* can be expensive when we increase the maximum element in K or D with larger input size n. To improve upon it, we propose three techniques which build up to the efficiency-oriented DASH.

Technique 1: Mixed-weights

Since convolution is linear, instead of computing $|K||D|$ convolutions, we can combine the kernels and compute convolution once. We call this approach ***mixed-weights***:

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) = \mathbf{Conv}\left(\sum_{k \in K} \sum_{d \in D} \alpha_{k,d} \cdot \mathbf{w}'_{k,d}\right)(\mathbf{x}). \tag{10.8}$$

Here $\mathbf{w}'$ is a padded version of $\mathbf{w}$ (appending 0's at the end of each dimension) that allows filters of different sizes to be added. The aggregated kernel has size $\bar{D} = \max_{k,d}(k-1)d+1$ and the n-dependent term of the complexity of *mixed-weights* is $c_{in}c_{out}\bar{D}n$. Hence, it removes the direct dependence of the leading-order term on $|K||D|$, the search space size, that *mixed-results* had.

Technique 2: Fourier convolution

If we wish to increase the kernel size and dilation with the input size, the complexity of *mixed-weights* will still grow *implicitly* with the search space size through the dependence on $\bar{D}$. To

[11] $\triangle$ denotes the probability simplex.

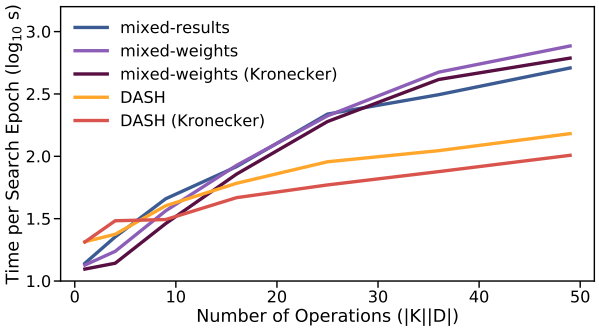

Figure 10.6: $\log_{10}$ time for one search epoch vs. numbers of operations in $S_{\mathbf{AggConv}_{K,D}}$. We use $K = \{2p+1 | 1 \leqslant p \leqslant c\}$ and $D = \{2^q - 1 | 1 \leqslant q \leqslant c\}$ while increasing c from 1 to 7.

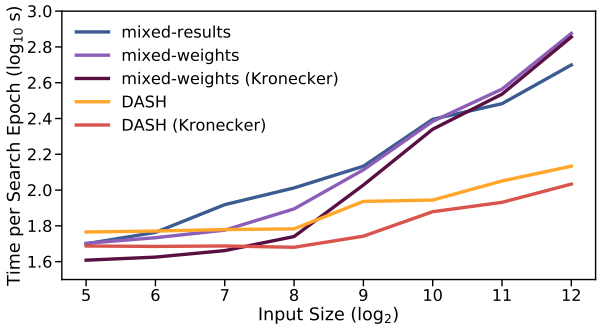

Figure 10.7: $\log_{10}$ time for one search epoch vs. input length of single-channel 1D data. We fix $K = \{3, 5, 7, 9, 11\}$, $D = \{1, 3, 7, 15, 31\}$ and test $n \in \{2^5, \ldots, 2^{12}\}$.

address this issue, we combine the kernel re-weighting idea with another technique motivated by the convolution theorem. Given a kernel $\mathbf{w}$, recall that $\mathbf{Conv}(\mathbf{w})(\mathbf{x}) = \mathbf{F}^{-1} \operatorname{diag}(\mathbf{F}\mathbf{w}')\mathbf{F}\mathbf{x}$, where $\mathbf{F}$ is the discrete Fourier transform (DFT) and $\operatorname{diag}(\mathbf{z})$ is a diagonal matrix with entries $\mathbf{z}$. In other words, convolution involves multiplying the DFT of the kernel by that of the input. Since the DFT can be applied in time $\mathcal{O}(n \log n)$ using the Fast Fourier Transform (FFT) and apart from that we only need element-wise multiplication, this yields an efficient approach to reducing dependence on the combined kernel size $\bar{D}$. While in XD we replaced the DFTs with a *continuous* set of matrices for XD-operations, DASH can be viewed as replacing the middle DFT with a *discrete* set of matrices for efficiency. Accordingly, DASH computes $\mathbf{AggConv}_{K,D}$ as follows:

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) = \mathbf{F}^{-1} \operatorname{diag}\left(\mathbf{F}\left(\sum_{k \in K}\sum_{d \in D} \alpha_{k,d} \cdot \mathbf{w}'_{k,d}\right)\right)\mathbf{F}\mathbf{x}. \tag{10.9}$$

Note that while the kernel changes for each $\mathbf{Conv}_{k,d}$, the input does not. Hence, we also save time by transforming the input to the frequency domain only once.

In Table 10.4, we report the theoretical complexities of the baselines and DASH, the latter leveraging both Technique 1 and 2. It is easy to obtain the operation complexities of *mixed-weights* and *mixed-results*. For DASH, the number of multiplications and additions can be attributed to the inner weight sum and multi-channel product (the first term) as well as three FFTs (the second). A detailed analysis is provided in Appendix 10.A.2. We see that *mixed-weights* is favorable to *mixed-results* when $\bar{D} < \bar{K} = \mathcal{O}(|D|k_{\text{max}}^2)$, which occurs with large kernels and a few dilations. On the other hand, only DASH completely separates the dominant terms containing $c_{in}c_{out}n$ from the size of the search space and its elements, replacing them by $\mathcal{O}(\log n)$, which is small for any realistic n. As we increase k_{max} and d_{max} for larger inputs, this will also lead to a slower asymptotic increase in complexity, making DASH an attractive choice for the multi-scale search space where $\bar{D}$ is large by design to extract possible long-range dependencies in the data.

Table 10.4: Complexity of different methods for computing $\mathbf{AggConv}$, denoting $\bar{K} = |D| \cdot \sum_{k \in K} k$ and $\bar{D} = \max_{k,d}(k-1)d + 1$. For details see Appendix 10.A.2.

Method	MULTs	ADDs
mixed-results (10.7)	$(c_{in}c_{out}\bar{K} + c_{out}\|K\|\|D\|)n$	$(c_{in}c_{out}\bar{K} + c_{out}\|K\|\|D\|)n$
mixed-weights (10.8)	$c_{in}c_{out}(\bar{K} + \bar{D}n)$	$c_{in}c_{out}\bar{D}(\|K\|\|D\| + n)$
DASH (10.9)	$c_{in}c_{out}(\bar{K} + n) + \mathcal{O}(c_{in}c_{out}n\log n)$	$c_{in}c_{out}(\|K\|\|D\|\bar{D} + n) + \mathcal{O}(c_{in}c_{out}n\log n)$

Technique 3: Kronecker dilation

To efficiently implement the kernel summation in *mixed-weights* and DASH on a GPU, we introduce our final technique: after initializing $\mathbf{w}_{k,d}$ for each $\mathbf{Conv}_{k,d}$ separately, we use a Kronecker product $\otimes$ to transform the undilated kernels into dilated forms. For example, to compute a 2D convolution with dilation d, we introduce the sparse pattern matrix $\mathbf{P} \in \mathbb{R}^{d \times d}$ whose entries are all 0's except for the upper-left entry $\mathbf{P}_{1,1} = 1$. Then, $\mathbf{w}_{k,d} = \mathbf{w}_{k,1} \otimes \mathbf{P}$. Beyond the theoretical gains shown in Wu et al. [2019], this dilation strategy is empirically faster than the standard way of padding 0's into $\mathbf{w}_{k,1}$ (Figure 10.6 and 10.7). After dilating the kernels, we sum them together, zero-pad to match the input size, and apply the FFTs.

Ablation study for the proposed techniques

To check that our asymptotic analysis leads to actual speedups and perform an ablation study on the proposed techniques, we evaluate the three methods on single-channel 1D inputs (experimental details can be found in Appendix 10.B.2). Since both *mixed-weights* and DASH require kernel summation, which can be implemented with Kronecker dilation (Technique 3), we compare five methods in total.

Figure 10.6 illustrates the combined forward- and backward-pass time in log scale for one search epoch vs. the size of $S_{\mathbf{AggConv}_{K,D}}$ when $n = 1000$. For small $\bar{D}$, the FFT overhead makes DASH runtime slightly longer but the difference is negligible. However, as $\bar{D}$ increases, the DASH curves grow much slower whereas the runtimes for the other methods scale with the number of operations. In Figure 10.7, we fix K and D to study how runtime is affected by input size n. Both *mixed-results* and *mixed-weights* become extremely inefficient for large n's which commonly occur in time-series or signal processing. Surprisingly, DASH's runtime does not increase much with n. We hypothesize that this is due to wallclock-time being dominated by data-passing at that speed.

In general, Technique 1 on its own scales poorly for the considered search space. This is why methods like MergeNAS [Wang et al., 2020a] cannot be used in our setting. Though XD makes use of Technique 2, it considers a parametrized space with infinitely many operations that need to be continuously evolved and is too expensive to be applied to tasks beyond CIFAR-100 (c.f. Figure 10.5). Technique 3 contributes to $2\times$ speedups for both *mixed-weights* and DASH. Overall, DASH (Kronecker) leads to about $10\times$ search-time speedups compared to the *mixed-results* scheme of DARTS for both the large operation space and large input size regimes. Hence, we use this version of DASH in later experiments.

10.3.3 Full pipeline: Search, hyperparameter tuning, and retraining

Having shown the main techniques for searching a large space of kernel patterns, we now specify the full search and model development pipeline. Given a dataset to train on, we set $K = \{3 + d(p-1) | 1 \leqslant p \leqslant p_{\text{max}}\}$ for kernel sizes and $D = \{2^q - 1 | 1 \leqslant q \leqslant q_{\text{max}}\}$ for dilations, with d, p_{max}, and q_{max} depending on the task's dimensionality. For 2D input, we set d to 2, p_{max} to 4, and q_{max} to 4, while for longer 1D sequence data, we set $d = 4$, $p_{\text{max}} = 5$, and $q_{\text{max}} = 4$. Thus we do not explicitly use larger kernels and dilations for tasks with larger input sizes, but determining if this useful to do is an interesting direction. To normalize architecture parameters into a probability distribution, we adopt the soft Gumbel Softmax activation, similar to Xie et al. [2019].

The backbone networks are as follows. For 2D tasks, we use WRN-16-1 as the search backbone to accelerate supernet training and WRN-16-4 for retraining. For 1D tasks, we use 1D WRN [Ismail Fawaz et al., 2020] in the entire pipeline. During search, we subsample the training data at each epoch. Given the loss for the target task, DASH jointly optimizes the model weights and the architecture parameters using direct gradient descent. This single-level optimization is more efficient than two-stage NAS, which finds initial assignments for architecture parameters and trains the candidates from scratch.

After searching for a predefined number of epochs, we discretize the search space and pick $\mathbf{Conv}_{k,d} \in S_{\mathbf{AggConv}_{K,D}}$ with the largest weight for each layer. The final model has a similar overall structure to the backbone, but the intermediate operations are tailored to the target task. To improve training stability, we additionally add a simple hyperparameter tuning stage between search and retraining using grid search (configuration space shown in Appendix 10.C.5). For each setting, we train the discretized model on *a subset* of the training data for *fewer* epochs so the tuning cost is a small fraction of the entire pipeline's cost (Table 10.20), Then, we evaluate the performance on a holdout validation set and select the configuration with the best validation score. As a final step, we retrain the discretized model with the optimal hyperparameters on all training data until convergence. Like other weight-sharing methods with discretization, our final model will be more efficient than the supernet.

10.3.4 Evaluation

We evaluate the performance of DASH on diverse tasks using ten datasets from NAS-Bench-360 [Tu et al., 2022], a benchmark spanning multiple application domains, input dimensions, and learning objectives.[12] These include classical vision tasks such as CIFAR-100 where CNNs do well, scientific computing tasks such as Darcy Flow where standard CNN backbones can perform poorly [Li et al., 2021c], sequence tasks such as DeepSEA where large dilations are preferred [Bai et al., 2018, Zhang et al., 2021], and many others. Thus, our evaluation will not only test whether DASH can find good architectures in the proposed new search space, but also investigate whether multi-scale convolution is a strong competitor for solving different problems. In fact, our results show that DASH is a top choice for many tasks, obtaining in-aggregate the best speed-accuracy tradeoffs among the methods we evaluate (c.f. Figure 10.9).

[12] We give a detailed summary of its tasks in Table 10.17.

Table 10.5: Error rates (lower is better) on NAS-Bench-360 tasks across diverse application domains and problem dimensions (the last three problems are 1D and the rest are 2D). DASH beats *all the other NAS methods* on 7/10 tasks and exceeds hand-designed expert models on 7/10 tasks. Scores of DASH are averaged over three trials. Baseline errors are from Tu et al. [2022]. See Table 10.19 for standard deviations.

	CIFAR-100	Spherical	Darcy Flow	PSICOV	Cosmic	NinaPro	FSD50K	ECG	Satellite	DeepSEA
	0-1 error (%)	0-1 error (%)	relative ℓ_2	MAE_8	1-AUROC	0-1 error (%)	1-mAP	1-F_1	0-1 error (%)	1- AUROC
Expert	**19.39**	67.41	0.008	3.35	**0.13**	8.73	0.62	**0.28**	19.8	0.30
WRN	23.35	85.77	0.073	3.84	0.24	6.78	0.92	0.43	15.49	0.40
TCN	-	-	-	-	-	-	-	0.57	16.21	0.44
WRN-ASHA	23.39	75.46	0.066	3.84	0.25	7.34	0.91	0.43	15.84	0.41
DARTS-GAEA	24.02	**48.23**	0.026	**2.94**	0.22	17.67	0.94	0.34	12.51	0.36
DenseNAS	25.98	72.99	0.10	3.84	0.38	10.17	0.64	0.40	13.81	0.40
AutoDL	-	-	0.049	6.73	0.49	-	-	-	-	-
AMBER	-	-	-	-	-	-	-	0.67	12.97	0.68
Perceiver IO	70.04	82.57	0.24	8.06	0.48	22.22	0.72	0.66	15.93	0.38
BABY DASH	25.56	63.45	0.016	3.94	0.16	8.28	0.62	0.37	13.29	0.37
DASH	24.37	71.28	**0.0079**	3.30	0.19	**6.60**	**0.60**	0.32	**12.28**	**0.28**

Baselines and experimental setup

We compare DASH with the following methods: DenseNAS [Fang et al., 2020] and GAEA PC-DARTS (c.f. Section 9.2.2), which represent general NAS; Auto-DeepLab [Liu et al., 2019a] and AMBER [Zhang et al., 2021], which represent specialist NAS methods for dense prediction and 1D tasks, respectively; 1D temporal convolutional network (TCN) [Bai et al., 2018], regular WRN, and WRN with hyperparameter tuner ASHA [Li et al., 2020b], which represent natural NAS baselines; and Perceiver IO [Jaegle et al., 2022], which represents non-NAS general-purpose models. While these results are available in Tu et al. [2022], we additionally add a BABY DASH baseline: we run DASH in the DARTS convolution space with $K = \{3, 5\}$ and $D = \{1, 2\}$ to study whether large kernel sizes and dilations are necessary to strong performance across-the-board. Finally, we compare our method to expert architectures from NAS-Bench-360. These models are representatives of the best that hand-crafting has to offer.

Each dataset is preprocessed and split using the NAS-Bench-360 script, with the training set being used for search, hyperparameter tuning, and retraining. To construct the multi-scale search space, we set K and D according to the rules in Section 10.3.3. We use the default SGD optimizer for the WRN backbone and fix the learning rate schedule and gradient clipping threshold for every task. The entire DASH pipeline is run on a single NVIDIA V100 GPU, which is what we use to report the runtime cost. More details can be found in Appendix 10.C.5.

We evaluate the performance of all competing methods following the NAS-Bench-360 protocol. We first report results of the target metric for each task by running the model of the *last*

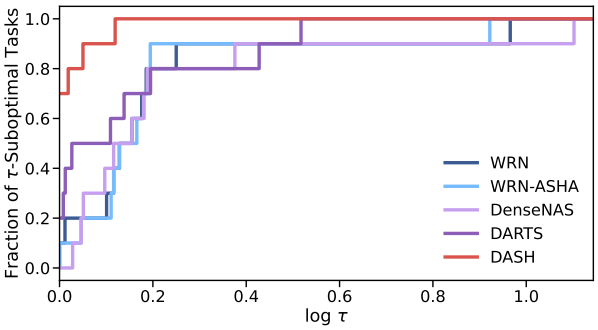

Figure 10.8: Performance profiles of general NAS methods and DASH on NAS-Bench-360. DASH being far in the top left corner indicates it is rarely suboptimal and is often the best.

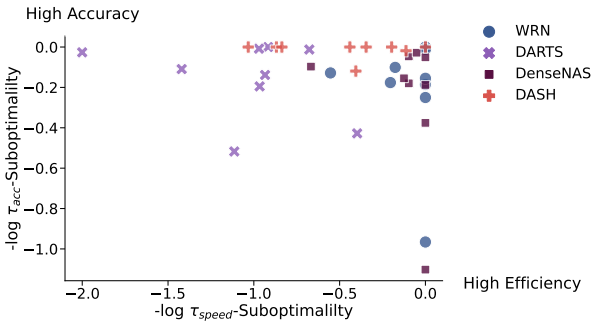

Figure 10.9: Comparing $-\log \tau$-suboptimality of speed vs. accuracy on all tasks. DASH's concentration in the top right corner indicates its strong efficacy-efficiency tradeoffs relative to the other methods.

epoch on the test data. Then, we report aggregate results via *performance profiles* [Dolan and Moré, 2002], a technique that considers both outliers and small performance differences to compare methods across multiple tasks robustly. In such plots, each curve represents one method. The τ on the x-axis denotes the fraction of tasks on which a method is no worse than a τ-factor from the best.

Results and discussion

We present the accuracy results for each task in Table 10.5 and the performance profiles in Figure 10.8. Figure 10.8 clearly demonstrates that DASH is superior to other competing methods in terms of aggregate performance. In particular, it ranks first among all *automated* models for 7/10 tasks, among all *expert* models for 7/10 tasks, and performs favorably when considering both accuracy and efficiency as shown in Figure 10.9. In addition, Table 10.6 shows that DASH outperforms DARTS in speed for all 10 tasks (in several cases by an order of magnitude), and attains comparable efficiency with training vanilla WRNs for 6/10 tasks (full-pipeline time is less than or about twice as long as the WRN training time). In the following, we provide a detailed analysis of the experimental results.

DASH dominates automated methods. Compared to other automated methods, DASH has a clear advantage in accuracy. Even for tasks where it does not beat the expert, e.g. ECG, DASH's performance is significantly better than other AutoML methods. It also outperforms specialist methods AutoDL and AMBER on dense prediction and 1D tasks, respectively. Although DARTS does best on CIFAR-100 (the task for which it was designed), Spherical, and PSICOV, it is the worst on NinaPro and FSD50K. Note that the underperformance of DASH on CIFAR-100 relative to WRN (and on Spherical and Cosmic relative to BABY DASH) suggests suboptimality of the gradient descent optimization procedure but not of the operation space, since WRN and BABY DASH are contained in our search space. This indicates a future direction to improve optimization in the DASH search space.

Table 10.6: Full-pipeline runtime in GPU hours for NAS-Bench-360 (PSICOV results are omitted due to a discrepancy in the implementation of data loading). DASH is consistently faster than DARTS, and it is less than a factor of two slower than simply training a WRN on six of the ten tasks. DenseNAS is fast but its accuracy is far less impressive. XD is too expensive to be applied to tasks other than CIFAR-100.

	CIFAR-100	Spherical	Darcy Flow	Cosmic	NinaPro	FSD50K	ECG	Satellite	DeepSEA
DARTS	9.5	16.5	6.5	21.5	0.5	37	140	28	39.5
DenseNAS	2.5	2.5	0.5	2.5	0.2	4.5	6.5	3	2
WRN	2	2	0.5	4	0.2	4	5	4.5	1.5
DASH	2.5	5	5.3	6.8	0.3	29	1.3	6.5	10

DASH dominates expert architectures. While the degree of sophistication of the expert networks varies task by task, the performance of DASH on tasks such as Darcy Flow suggests that it is capable of competing with highly specialized networks, e.g. Fourier Neural Operator [Li et al., 2021c] for PDE solving. These results imply that DASH, and more generally the strategy of equipping backbone networks with task-specific kernels, is a promising approach for tackling model development in new domains. Meanwhile, DASH consistently outperforms Perceiver IO which represents non-automated general-purpose models. We speculate that the poor performance of Perceiver IO is because it is developed on more sophisticated natural language and multi-modal reasoning tasks and training Transformers is generally difficult.

Large kernels are needed. We also ablate the large-k-large-d design of the search space by comparing DASH with BABY DASH. We hypothesize that for the same task, a small performance gap between the two methods would indicate that small kernels suffice for extracting local features, whereas a major degradation in the quality of the BABY DASH model can imply that the task needs global modeling. Consequently, datasets such as Darcy Flow and ECG provide compelling evidence that kernels with large receptive fields play an important role in solving real-life problems and further back up the design of our multi-scale convolutional search space. An example of the series of WRN kernels found by DASH on Darcy Flow is: $\mathbf{Conv}_{5,3} \rightarrow \mathbf{Conv}_{3,1} \rightarrow \mathbf{Conv}_{3,1} \rightarrow \mathbf{Conv}_{3,15} \rightarrow \mathbf{Conv}_{7,15} \rightarrow \mathbf{Conv}_{9,7} \rightarrow \mathbf{Conv}_{9,7} \rightarrow \mathbf{Conv}_{3,7} \rightarrow \mathbf{Conv}_{5,7} \rightarrow \mathbf{Conv}_{5,15} \rightarrow \mathbf{Conv}_{9,7} \rightarrow \mathbf{Conv}_{3,7} \rightarrow \mathbf{Conv}_{7,7}$. We can see that large kernels are indeed selected during search. More visualizations can be found in Appendix 10.D.

DASH is computationally efficient. In addition to low error, we also care about the efficiency of the selected models. Table 10.6 provides the combined search and retraining time in GPU hours for DARTS, DenseNAS, and DASH, as well as the training time for a vanilla WRN-16-4 without hyperparameter tuning (baseline results are taken from Tu et al. [2022]). We also present the breakdown of DASH's full-pipeline runtime in Table 10.20. A key observation is that the cost of DASH is consistently below DARTS' on all tasks and is similar to training a simple CNN for more than half of them. Although DenseNAS is fast, its prediction performance is poor.

In Figure 10.9, we visualize the tradeoff between efficiency and effectiveness for each method and task combination. Evidently, DASH takes an important step towards bridging the gap between the efficiency of DARTS and the expressivity of XD. The fact that DASH can be trained at a low cost testifies that we need not sacrifice efficiency for adding more operations. In fact, we have actually shown that DASH is *both* faster *and* more effective than DARTS in many tasks.

DASH works with other backbones and is backwards-compatible. Lastly, in addition to the Wide ResNet backbone and NAS-Bench-360 tasks, we have also verified the efficacy of DASH on other backbones including TCN [Bai et al., 2018] and ConvNeXt [Liu et al., 2022], as well as on the larger ImageNet dataset. In particular, DASH is able to achieve a 1.4% increase in top-1 accuracy for ImageNet on top of the ConvNeXt backbone (c.f. Appendix 10.C.6). As the latter was itself developed in part via manual tuning of the kernel size, this means that DASH outperforms human hand-tuning on ImageNet. These results show that DASH is both backbone-agnostic and also works well with computer vision tasks, making it backward-compatible with the original use-case of CNNs.

10.3.5 Limitations and future work

There are several open problems which we leave for future work. First, it is beneficial to study why certain kernel patterns are chosen, as the selected operations can hint us at the intrinsic properties of the datasets. Second, one can improve upon DASH, e.g. by including non-square convolutions for 2D problems, using a better optimization algorithm, or developing techniques that further reduces the memory usage of performing a forward architecture search pass. One could also construct a more comprehensive search space with high-level operators such as self-attention [Vaswani et al., 2017].

Meanwhile, although we focus on NAS, which is an alternative to fine-tuning pretrained models, the aggregated convolution can be a plug-and-play module for algorithms that search for large-scale models. For instance, many Transformer models still depend on convolutions for feature extraction and transformation, and their performance relies on the quality of the embedded features. Since DASH is applicable to any architecture with a convolutional layer, it can be helpful for such models, including Vision Transformer with a convolutional patching layer [Dosovitskiy et al., 2021], Deformable Transformer with a ResNet embedder [Zhu et al., 2021], Swin Transformer with a convolutional decoder [Liu et al., 2021b], and many others.

10.4 Conclusion

In this final chapter of the thesis we argued that a discovering effective architecture for diverse tasks is a crucial but underexplored goal of NAS. To that end, we undertook the study of search spaces likely to contain good operations for processing a wide variety of data modalities, starting with XD-operations and concluding with DASH; we then designed methods for searching for and integrating these operations into existing CNN backbones. The latter approach overcomes the computational limitations of differentiable NAS and obtains high-quality models with accuracy comparable to or better than that of the handcrafted networks on many tasks.

10.A Analyses

10.A.1 XD-operations: Expressivity

Here we collect results on the expressivity of the set of **XD**-operations. For simplicity, our results will be in the following single-dimensional ($N = 1$) setting:

Setting 10.A.1. We consider input spaces of form $\mathcal{X} = \mathbb{R}^{c \times m}$ for input size $m \in \mathcal{N}$ and channel count $c \in \mathcal{N}$ and parameter spaces $\mathcal{W} = \mathbb{R}^{c \times c \times k}$ for filter size $k \in [n]$, where output size $n \geqslant m$ is a power of 2.

It is straightforward to extend the results to multiple dimensions using Kronecker products and to input sizes other than powers of two using padding. Note that all of our results will also assume a circular padded domain.

Convolutions

Definition 10.A.1. A **convolution** in Setting 10.A.1 with filter size k, dilation $d \in [\lfloor \frac{n-1}{k-1} \rfloor]$, stride $s \in [n-1]$, and channel groups described by a matrix $\mathbf{B} \in \{0,1\}^{n \times n}$ s.t. $\mathbf{B}_{[i,j]} = 1$ if channels i and j are in the same group and 0 otherwise is a parameterizable operation that for any weight $\mathbf{w} \in \mathcal{W}$ outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to

$$\frac{1}{n} \begin{pmatrix} \operatorname{diag}(\mathfrak{a}_s(\underline{\mathbf{1}_{\lceil \frac{n}{s} \rceil}})) \sum_{j=1}^{c} \mathbf{B}_{[1,j]} \mathbf{F}_n^{-1} \operatorname{diag}(\mathbf{F}_n \mathfrak{a}_d(\underline{\mathbf{w}_{[1,j]}})) \mathbf{F}_n \mathbf{x}_{[j]} \\ \vdots \\ \operatorname{diag}(\mathfrak{a}_s(\underline{\mathbf{1}_{\lceil \frac{n}{s} \rceil}})) \sum_{j=1}^{c} \mathbf{B}_{[c,j]} \mathbf{F}_n^{-1} \operatorname{diag}(\mathbf{F}_n \mathfrak{a}_d(\underline{\mathbf{w}_{[c,j]}})) \mathbf{F}_n \mathbf{x}_{[j]} \end{pmatrix} \tag{10.10}$$

where $\mathbf{F}_n \in \mathcal{C}^{n \times n}$ is the $n \times n$ DFT and $\mathfrak{a}_d : \mathbb{R}^n \mapsto \mathbb{R}^n$ is an atrous permutation of a vector that is equivalent to multiplication by some permutation matrix $\mathbf{P}_d \in \{0,1\}^{n \times n}$. We will use $\mathbf{Conv}_k$ to denote the case of $d = 1$, $s = 1$, and $\mathbf{B} = \mathbf{1}_{c \times c}$.

Claim 10.A.1. All multi-channel convolutions of the form given in Definition 10.A.1 are contained in the search space of XD-operations of depth $(1, 3, 1)$.

Proof. Setting the architecture parameters to be $\mathbf{K} = \operatorname{diag}(\mathfrak{a}_s(\underline{\mathbf{1}_{\lceil \frac{n}{s} \rceil}}))\mathbf{F}_n^{-1}$, $\mathbf{L} = \mathbf{F}_n \mathbf{P}_d$, $\mathbf{M} = \mathbf{F}_n$, $\mathbf{b} = \mathbf{0}_n$, and $\mathbf{C} = \mathbf{B}$, and noting that (a) the DFT and its inverse are both depth 1 K-matrices, (b) multiplying a K-matrix by a diagonal matrix is another K-matrix of the same depth, and (c) permutation matrices are K-matrices of depth 2 yields the result. These three facts can be found in the original paper [Dao et al., 2020]. □

Remark 10.A.1. Note that for the case of dilation $d = 1$ the result in Claim 10.A.1 holds with depth $\mathbf{1}_3$.

Parameter-free operations

Definition 10.A.2. The **skip-connection** in Setting 10.A.1 is parameterizable operation that outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to itself. The **zero-operation** in Setting 10.A.1 is parameterizable operation that outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to $\mathbf{0}_{c \times n}$.

Claim 10.A.2. The skip-connection and zero-operation are both contained in the search space of XD-operations of depth $\mathbf{1}_3$.

Proof. For both set the architecture parameters to be $\mathbf{K} = \mathbf{F}_n^{-1}$, $\mathbf{L} = \mathbf{0}_{n \times n}$, $\mathbf{M} = \mathbf{F}_n$, and $\mathbf{C} = \mathbf{I}_c$. To obtain the skip-connection set $\mathbf{b} = \mathbf{1}_n$; to obtain the zero-operation set $\mathbf{b} = \mathbf{0}_n$. □

Definition 10.A.3. An **average pooling** operation in Setting 10.A.1 with filter size k, dilation $d \in [\lfloor \frac{n-1}{k-1} \rfloor]$, and stride $s \in [n-1]$ is parameterizable operation outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to the output of a convolution (as in Definition 10.A.1) with the same filter size, dilation, and stride, channel groups described by $\mathbf{B} = \mathbf{I}_c$, and filters $\mathbf{w}_{[j,j]} = \mathbf{1}_k / k \; \forall \; j \in [c]$.

Claim 10.A.3. All average pooling operations are contained in the search space of XD-operations of depth $\mathbf{1}_3$.

Proof. Setting the architecture parameters to be $\mathbf{K} = \mathrm{diag}(\mathfrak{a}_s(\underline{\mathbf{1}_{\lceil \frac{n}{s} \rceil}}))\mathbf{F}_n^{-1}$, $\mathbf{L} = \mathbf{0}_{n \times n}$, $\mathbf{M} = \mathbf{F}_n$, $\mathbf{b} = \mathfrak{a}_d(\underline{\mathbf{1}_k / k})$, and $\mathbf{C} = \mathbf{I}_c$ and noting that (a) the DFT and its inverse are both depth 1 K-matrices and (b) multiplying a K-matrix by a diagonal matrix of the same depth is another K-matrix of the same depth yields the result. □

Compositions with multiplication by fixed K-matrix

Definition 10.A.4. A **fixed linear operation** $\mathbf{Lin}_{\mathbf{A}}$ in Setting 10.A.1 with matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a parameterizable operation that outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to $\mathbf{Lin}_{\mathbf{A}}(\mathbf{w})(\mathbf{x}) = \begin{pmatrix} \mathbf{A}\mathbf{x}_{[1]} & \cdots & \mathbf{A}\mathbf{x}_{[c]} \end{pmatrix}^\top$. For example, $\mathbf{Lin}_{\mathbf{I}_c} = \mathbf{Id}$.

Definition 10.A.5. Let $\mathbf{Op}_1$ and $\mathbf{Op}_2$ be two parameterizable operations in Setting 10.A.1 with $\mathcal{X}$. Then for any weight $\mathbf{w} \in \mathcal{W}$ their **composition** $\mathbf{Op}_1 \circ \mathbf{Op}_2$ outputs the parameterized function $\mathbf{Op}_1(\mathbf{w}) \circ \mathbf{Op}_2(\mathbf{w})$.

Claim 10.A.4. Let $\mathbf{Op}$ be a parameterizable operation in Setting 10.A.1 that is contained in the set of XD-operations of some depth $\mathbf{d} \in \mathcal{N}^3$ and let $\mathbf{A}$ be a K-matrix of depth d'. Then $\mathbf{Op} \circ \mathbf{Lin}_{\mathbf{A}}$ is contained in the set of XD-operations of depth $(\mathbf{d}_{[1]}, \mathbf{d}_{[2]}, \mathbf{d}_{[3]} + d')$ and $\mathbf{Lin}_{\mathbf{A}} \circ \mathbf{Op}$ is contained in the set of XD-operations of depth $(\mathbf{d}_{[1]} + d', \mathbf{d}_{[2]}, \mathbf{d}_{[3]})$.

Proof. Let $\mathbf{K}$ and $\mathbf{M}$ be the first and last K-matrices of the representation of $\mathbf{Op}$ as an XD-operation, which thus have depth at most $\mathbf{d}_{[1]}$ and $\mathbf{d}_{[3]}$, respectively. Then the representation of $\mathbf{Op} \circ \mathbf{Lin}_{\mathbf{A}}$ as an XD-operation is the same except with depth $\mathbf{d}_{[3]} + d'$ K-matrix $\mathbf{MA}$ as the last K-matrix, and similarly the representation of $\mathbf{Lin}_{\mathbf{A}} \circ \mathbf{Op}$ as an XD-operation is the same except with depth $\mathbf{d}_{[1]} + d'$ K-matrix $\mathbf{AK}$ as the first K-matrix. □

Other named operations

Definition 10.A.6. Suppose we have a fixed n-node graph with adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$, and let $\hat{\mathbf{A}}$ and $\hat{\mathbf{D}}$ be the adjacency and degree matrices, respectively, of the same graph but with added self-loops. Then regular **graph convolution** [Kipf and Welling, 2017] in Setting 10.A.1 with $k = 1$ is a parameterizable operation that for any weight $\mathbf{W} \in \mathcal{W}$ outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to $\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{x}^\top\mathbf{w}$ and the **diffusion graph convolution** [Li et al., 2018b] in Setting 10.A.1 with $k = 1$ is a parameterizable operation that for any weight $\mathbf{W} \in \mathcal{W}$ outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to $\mathbf{D}^{-1}\mathbf{A}\mathbf{x}^\top\mathbf{w}$.

Claim 10.A.5. Suppose $\mathbf{A}$ and $\hat{\mathbf{A}}$ can be represented by K-matrices of depth d and $\hat{d}$, respectively. Then the corresponding graph convolution is contained in the search space of XD-operations of depth $(1, 1, \hat{d} + 1)$ and the corresponding diffusion graph convolution in that of depth $(1, 1, d + 1)$.

Proof. For any $\mathbf{G} \in \mathbb{R}^{n\times n}$ we have $\mathbf{G}\mathbf{x}^\top\mathbf{w} = \mathbf{Lin}_{\mathbf{G}}(\mathbf{w})(\mathbf{x})\mathbf{w} = \mathbf{Conv}_1(\mathbf{w})(\mathbf{Lin}_{\mathbf{G}}(\mathbf{w})(\mathbf{x})) = (\mathbf{Conv}_1 \circ \mathbf{Lin}_{\mathbf{G}})(\mathbf{w})(\mathbf{x})$. The result follows by Claims 10.A.1 and 10.A.4, the fact that a K-matrix multiplied by a diagonal matrix is another K-matrix of the same depth, and by substituting $\mathbf{G} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ (for graph convolution) or $\mathbf{G} = \mathbf{D}^{-1}\mathbf{A}$ (for diffusion graph convolution). □

Remark 10.A.2. Note that the above claim is meaningful because adjacency matrices of realistic graphs are usually sparse and sparse matrices can be efficiently represented as K-matrices [Dao et al., 2020].

Definition 10.A.7. A **Fourier neural operator** (FNO) [Li et al., 2021c] in Setting 10.A.1 with even k and thus $k/2$ modes is a parameterizable operation that for any weight $\mathbf{w} \in \mathcal{W}$ outputs a function mapping every $\mathbf{x} \in \mathcal{X}$ to

$$\begin{pmatrix} \text{Real}\left(\sum_{j=1}^{c} \mathbf{F}_n^{-1}\,\text{diag}(\begin{pmatrix}\mathbf{w}_{[1,j,1:k/2]} + i\mathbf{w}_{[1,j,k/2+1:k]} & \mathbf{0}_{n-k/2}\end{pmatrix}^\top)\mathbf{F}_n\mathbf{x}_{[j]}\right) \\ \vdots \\ \text{Real}\left(\sum_{j=1}^{c} \mathbf{F}_n^{-1}\,\text{diag}(\begin{pmatrix}\mathbf{w}_{[c,j,1:k/2]} + i\mathbf{w}_{[c,j,k/2+1:k]} & \mathbf{0}_{n-k/2}\end{pmatrix}^\top)\mathbf{F}_n\mathbf{x}_{[j]}\right) \end{pmatrix} \tag{10.11}$$

Claim 10.A.6. The FNO with $k/2$ modes is contained in the search space of XD-operations of depth $(1, 4, 1)$.

Proof. Setting the architecture parameters to be $\mathbf{K} = \mathbf{F}_n^{-1}$, $\mathbf{L} \in \mathcal{C}^{n\times n}$ the n-sparse matrix mapping $\underline{\mathbf{w}}$ to $\begin{pmatrix}\mathbf{w}_{[1,j,1:k/2]} + i\mathbf{w}_{[1,j,k/2+1:k]} & \mathbf{0}_{n-k/2}\end{pmatrix}^\top$, $\mathbf{M} = \mathbf{F}_n$, $\mathbf{b} = \mathbf{0}_n$, and $\mathbf{C} = \mathbf{1}_{c\times c}$, and noting that an n-sparse matrix is a depth-4 K-matrix [Dao et al., 2020] yields the result. □

Remark 10.A.3. If we allow the parameter space in Setting 10.A.1 to be complex then the FNO with all k modes will be contained in the search space of XD-operations of depth $\mathbf{1}_3$.

Definition 10.A.8. Each channel of **transposed convolution** with stride $d(k-1)+1$, where k is the kernel size and d is the dilation rate, computes a feature map in which each input element is replaced by that element multiplied by the dilated filter of size $d(k-1)+1$. The multi-channel extension of this over parameter space $\mathcal{W} = \mathbb{R}^{c\times c\times k}$ is similar to that for standard convolutions.

Claim 10.A.7. All transposed convolutions with stride equal to the dilated kernel size are contained in the search space of XD-operations of depth $(1, 3, 3)$.

Proof. A transposed convolution is equivalent to a regular convolution with the same filter applied to the input after it has been zero-padded and then permuted to separate all entries by $d(k-1)$ zeros. Since permutations are K-matrices of depth 2 the result follows by Claims 10.A.1 and Claim 10.A.4. □

Definition 10.A.9. A **depthwise-separable convolution** in Setting 10.A.1 with filter size k but with parameter space $\mathcal{W} = \mathbb{R}^{c\times k} \times \mathbb{R}^{c\times c}$ is a parameterizable operation that for any weight $\mathbf{w} \in \mathcal{W}$ outputs $\mathbf{Conv}_1(\mathbf{w}_{[2]}) \circ \mathbf{Conv}_{k,\mathbf{I}_c}(\mathbf{w}_{[1]})$, where $\mathbf{Conv}_{k,\mathbf{I}_c}$ denotes the convolution in Definition 10.A.1 with $\mathbf{B} = \mathbf{I}_c$.

Remark 10.A.4. Since both $\mathbf{Conv}_1$ and $\mathbf{Conv}_{k,\mathbf{I}_c}$ are XD-operations, by definition depthwise-separable convolutions are contained in the search space of composed XD-operations, which by Claim 10.A.2 also contains all of the above operations.

10.A.2 DASH: Asymptotic analysis

In this section we outline the runtime analysis used to populate the asymptotic complexities in Table 10.4. All three methods in the table—*mixed-results*, *mixed-weights*, and DASH—are computing the following weighted sum of convolutions:

$$\mathbf{AggConv}_{K,D}(\mathbf{x}) = \sum_{k\in K}\sum_{d\in D} \alpha_{k,d} \cdot \mathbf{Conv}(\mathbf{w}_{k,d})(\mathbf{x}). \tag{10.12}$$

We consider 1D inputs $\mathbf{x}$ with length n and c_{in} input channels; the convolutions have c_{out} output channels. We view $\mathbf{Conv}(\mathbf{w}_{k,d})(\mathbf{x})$ as having the naive complexity $c_{in}c_{out}kn$ since the deep learning frameworks use the direct (non-Fourier) algorithm. *mixed-results* computes the sum directly, which involves (1) applying one convolution of each size k and dilation to $\mathbf{x}$ at a cost of $c_{in}c_{out}kn$ MULTs and ADDs each for a total cost of $c_{in}c_{out}\bar{K}n$, (2) scalar-multiplying the outputs at a cost of $c_{out}|K||D|n$ MULTs, and (3) summing the results together at a cost of $c_{out}|K||D|n$ ADDs. *mixed-weights* instead (1) multiplies all kernels by their corresponding weight at a cost of $c_{in}c_{out}\bar{K}$ MULTs, (2) zero-pads the results to the largest effective kernel size $\bar{D}$ and adds them together at a cost of $c_{in}c_{out}|K||D|\bar{D}$ ADDs, and (3) applies the resulting $\bar{D}$-size convolution to the input at a cost of $c_{in}c_{out}\bar{D}n$ MULTs and ADDs. Finally, DASH also (1) does the first two steps of *mixed-weights* at a cost of $c_{in}c_{out}\bar{K}$ MULTs and $c_{in}c_{out}|K||D|\bar{D}$ ADDs but then (2) pads the resulting $\bar{D}$-size convolution to size n and applies an FFT at a cost of $\mathcal{O}(c_{in}c_{out}n\log n)$ MULTs and ADDs, (3) applies an FFT to $\mathbf{x}$ at a cost of $\mathcal{O}(c_{in}n\log n)$, (4) element-wise multiplies the transformed filters by the inputs at a cost of $c_{in}c_{out}n$ MULTs, (5) adds up c_{in} results for each of c_{out} output channels at a cost of $c_{in}c_{out}$ MULTs, and (6) applies an iFFT to the result at a cost of $\mathcal{O}(c_{out}n\log n)$.

Table 10.7: Comparison of the computational and memory costs of XD-operations when substituted for convolutions. For simplicity, we consider cases with 2D inputs and where the channel and bias parameters are fixed.

Task (backbone)	input size	kernel size	minutes / epoch		memory (Gb)		param. ($\times 10^6$)	
			Conv	**XD**	**Conv**	**XD**	**Conv**	**XD**
CIFAR-10 (WRN-40-4)	32	3	1.4	4.3	3.73	15.6	8.96	9.08
Darcy Flow (Conv4*)	85	13	0.028	0.14	4.51	5.53	0.701	0.744
PSICOV (ResNet-18)	128	3	5.9	11	1.50	10.7	0.038	0.549

* Four-layer convolutional network with parameterized skip (shortcut) connections derived from the FNO network [Li et al., 2021c] as described in Section 10.2.4.

10.B Computational cost

10.B.1 XD-operations

In this section we report a detailed comparison of computational costs of the XD-operation compared to a convolution; this is presented in Table 10.7. Due to their familiarity, we present results for tasks that have 2D inputs and thus use 2D convolutions in their default backbone. Note that since XD-operations are more general than convolutions, they must by definition be at least as expensive as convolutions in both computation and memory. While our focus is on absolute performance using learning metrics (e.g. test error), we view finding a good tradeoff between the performance of XD-operations on certain tasks and convolutions, for example by restricting the expressivity of XD-operations, as important directions for future work.

10.B.2 DASH

For the speed tests in Figures 10.6 and 10.7 we work with the Sequential MNIST dataset, i.e. the 2D 28×28 images are stretched into 1D with length 784. We zero pad or truncate the input to generate data with different input size n. The backbone is 1D WRN with the same structure as introduced in Section 10.C.5. We run the workflows on a single NVIDIA V100 GPU with a batch size of 128 and report timing results as $\log_{10}$(combined forward and backward pass time for one search epoch).

In Figure 10.6, we study how the size of our multi-scale convolution search space affects the runtimes of *mixed-results*, *mixed-weights*, and DASH for $n = 1000$ (zero-padded MNIST). We define $K = \{3 + 2(p-1) | 1 \leqslant p \leqslant c\}$, $D = \{2^q - 1 | 1 \leqslant q \leqslant c\}$ and vary c from 1 to 7. Consequently, the number of operations included in the search space grows from 1 to 49. In Figure 10.7, we study how the input size affects the runtimes of the three methods. We fix $K = \{3, 5, 7, 9, 11\}$, $D = \{1, 3, 7, 15, 31\}$ and vary n from 2^5 to 2^{12}.

Table 10.8: Architecture optimizer settings on CIFAR-10 tasks. Note that the step-size is updated using the same schedule as the backbone.

search space	backbone	task	optimizer	initial step-size	warmup epochs	perturb
$\tilde{\mathcal{S}}_{\mathbf{discrete}}$	LeNet	CIFAR-10	Adam	1E-1	0	0.1
		Permuted	Adam	1E-1	50	0.875
	ResNet-20	CIFAR-10	Adam	1E-3	0	0.1
		Permuted	Adam	1E-1	0	0.875
$\mathcal{S}_{\mathbf{XD}}$	LeNet	CIFAR-10	Adam	1E-4	0	-
		Permuted	Adam	1E-3	0	-
	ResNet-20	CIFAR-10	Adam	1E-4	50	-
		Permuted	Adam	1E-3	0	-

10.C Evaluations

Code to reproduce our results with XD-operations is available at `https://github.com/nick11roberts/XD`. We have also provided software to easily apply them to new tasks and CNN backbones here: `https://github.com/mkhodak/relax`. Additional code extending the latter to DASH is available at `https://github.com/sjunhongshen/DASH`.

10.C.1 CIFAR-10 and Permuted CIFAR-10

For our experiments with vision backbones we use the standard CIFAR-10 dataset [Krizhevksy, 2009] and a permuted version where all rows and columns are identically permuted. For unpermuted data we use standard data augmentation [He et al., 2016] while for permuted data we do not use any data augmentation. As specified in Section 10.2.3, we keep the training routine of the model weights the same and tune only the architecture optimizer, the settings of which are specified in Table 10.8. Note that for the DARTS operation space we specify a "perturb" parameter that specifies how unbiased the initial architecture parameters are towards the backbone operation; specifically, we initialize architecture parameters so as to assign one minus this quantity as the weight to the backbone operation, so 0.875 means the initialization is uniform (since $|\tilde{\mathcal{S}}_{\mathbf{discrete}}| = 8$) while 0.1 means the backbone operation is assigned 0.9 of the weight.

LeNet

The LeNet backbone we use consists of two $\mathbf{Conv}_{5\times5}$ layers, each followed by $\mathbf{MaxPool}_{2\times2}$, and two fully connected layers. When warm-starting with XD-operations we use $\mathbf{AvgPool}_{2\times2}$ instead of $\mathbf{MaxPool}_{2\times2}$; when warm-starting with the DARTS operations we use $\mathbf{MaxPool}_{3\times3}$. For the baseline training routine we use 200 epochs of Momentum(0.9), with the first 100 at learning rate 0.01, the next 50 at 0.005, and the last 50 at 0.001.

ResNet-20

We use the implementation and training routine provided here: `https://github.com/akamaster/pytorch_resnet_cifar10`. When replacing operations in the backbone we substitute for both the $\mathbf{Conv}_{3\times3}$ operations and the skip-connections $\mathbf{Id}$; some of the latter are downsampled, which XD-operations can handle as strides.

WRN-40-4

We use the same implementation as for ResNet-20 but adapt the original WRN training routine [Zagoruyko and Komodakis, 2016], except with weight decay set to 10^{-4} (as in ResNet-20); on the regular CIFAR-10 tasks this does not seem to affect performance. To conserve computation and memory, we do not tune the architecture optimizer parameters here and simply use the same ones used for ResNet-20; furthermore, we fix the channel and bias parameters of XD-operations and do not allow the kernel size to be larger the 3×3. Because of these modifications, we only use our evaluation here as a sanity check for large-network performance of XD-operations and do not include it in the main results.

DARTS cell search

To search the full DARTS search space, which is a standard NAS benchmark, we use our GAEA PC-DARTS method (c.f. Section 9.2.2). On CIFAR-10 we simply use their best reported cell but evaluate it using the "base" routine [Yang et al., 2020], i.e. without auxiliary losses or additional data augmentation; this is to obtain fair comparison with the other backbone models. Note that the model is still much larger and the training routine much more intensive. On permuted data we follow the standard three-stage pipeline in which we run search four times, train all four found cells and select the best one, and finally train that cell multiple times.

DenseNAS search

We use the DenseNAS search and evaluation code released by the authors here: `https://github.com/JaminFong/DenseNAS`. While the search space is designed for ImageNet [Russakovsky et al., 2015], we adapt it to CIFAR-10 by taking the DenseNAS-R1 setting and downscale the input sizes to match 32x32 images used.

10.C.2 Solving PDEs

For our PDE experiments, we use the FNO code and setup [Li et al., 2021c] provided here: `https://github.com/zongyi-li/fourier_neural_operator`. We use the same training routine and settings as the backbone architecture for each task and only tune the architecture optimizer. We consider the following hyperparameters for the architecture optimizer: Adam vs. SGD (with or without momentum), initial learning rate, and number of warmup epochs. The final hyperparameters for each task can be found in Table 10.10. Our CNN backbone is analogous to the FNO architecture used for each problem. In particular, the CNN backbone architecture used for each task is simply the FNO architecture where FNO layers of dimension N

Table 10.9: Search space comparison on CIFAR-10. Validation accuracies are averages of three trials.

Backbone	Search Space	CIFAR-10	Permuted*	Cost (hours†)
LeNet	backbone	75.5 ± 0.1	43.7 ± 0.5	0.3
	$\tilde{\mathcal{S}}_{\mathbf{discrete}}$	75.6 ± 3.4	47.7 ± 1.0	1.0
	$\mathcal{S}_{\mathbf{XD}}$	77.7 ± 0.7	63.0 ± 1.0	0.9
ResNet-20	backbone	91.7 ± 0.2	58.6 ± 0.7	0.6
	$\tilde{\mathcal{S}}_{\mathbf{discrete}}$	92.7 ± 0.2	58.0 ± 1.0	5.3
	$\mathcal{S}_{\mathbf{XD}}$	92.4 ± 0.2	73.5 ± 1.6	5.6
WRN-40-4	backbone	95.2 ± 0.1	64.7 ± 0.9	4.6
	$\tilde{\mathcal{S}}_{\mathbf{discrete}}$	95.2 ± 0.2	61.3 ± 1.3	19.9
	$\mathcal{S}_{\mathbf{XD}}$	95.0 ± 0.1	72.9 ± 0.8	14.3
ResNet-18 Cell	DenseNAS	94.5 ± 0.3	61.6 ± 3.3	3.6
	DARTS‡	96.0 ± 0.2	66.3 ± 0.5	28.6

* No data augmentation used in the permuted case.
† On a V100 GPU; time for DARTS Cell is training cost only.
‡ Search using GAEA PC-DARTS (c.f. Section 9.2.2); training using "base" routine [Yang et al., 2020].

Table 10.10: Architecture optimizer settings on PDE tasks. Note that the step-size is updated using the same schedule as the backbone.

task	optimizer	initial step-size	warmup epochs
1D Burgers' equation	Adam	1E-3	0
1D Burgers' equation (FNO init)	Momentum(0.5)	1E-4	250
2D Darcy Flow	Momentum(0.5)	1E-1	0
2D Darcy Flow (FNO init)	Momentum(0.5)	1E-1	0
2D Navier Stokes ($\nu = 10^{-4}, T = 30$)	Momentum(0.5)	5E-3	0
2D Navier Stokes ($\nu = 10^{-5}, T = 20$)	Momentum(0.5)	1E-3	0

with m modes are replaced by N-dimensional convolutional layers with filters of size $(m+1)^N$ and circular padding to match the dimensionality of FNO. In Table 10.11 and Table 10.12 we present reported [Li et al., 2021c], reproduced, and our own results on the 1D Burgers' equation and 2D Darcy Flow.

For AutoDL we borrow the code and setup provided here: `https://github.com/NoamRosenberg/autodeeplab`. We only conduct search on the lowest resolution and use the resulting architecture at higher resolutions. Search was conducted for 40 epochs, as in the original paper, and the search learning rate was tuned.

Table 10.11: Test relative errors on the 1D Burgers' equation. We were not able to match the FNO-1D results reported by the authors [Li et al., 2021c] using their published codebase, however, our proposed XD operations outperform our reproduction of their results at every resolution. Furthermore, we outperform their reported test relative errors on every resolution except $s = 4096$, where we roughly match their performance.

Method (source)	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN [Li et al., 2021c]	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN [Li et al., 2021c]	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN [Li et al., 2021c]	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN [Li et al., 2021c]	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO [Li et al., 2021c]	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO [Li et al., 2021c]	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO [Li et al., 2021c]	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO-1d [Li et al., 2021c]	0.0149	0.0158	0.0160	0.0146	**0.0142**	0.0139
CNN (ours)	0.0518	0.1220	0.1830	0.2280	0.2730	0.2970
FNO-1d (reproduced)	0.0181	0.0191	0.0188	0.0184	0.0183	0.0183
CNN XD (ours)	**0.0141**	**0.0079**	**0.0154**	**0.0099**	0.0145	**0.0123**
FNO-1d XD (ours)	0.0153	0.0154	0.0154	0.0167	0.0160	0.0155

Table 10.12: Test relative errors on 2D Darcy Flow. Our reproduction of the FNO-2D results outperform those reported by the authors [Li et al., 2021c]. Nonetheless, our proposed XD operations outperform both our reproduction and the reported results at every resolution.

Method (source)	$s = 85$	$s = 106$	$s = 141$	$s = 211$	$s = 421$
NN [Li et al., 2021c]	0.1716	-	0.1716	0.1716	0.1716
GCN [Li et al., 2021c]	0.0253	-	0.0493	0.0727	0.1097
FCN [Li et al., 2021c]	0.0299	-	0.0298	0.0298	0.0299
PCANN [Li et al., 2021c]	0.0244	-	0.0251	0.0255	0.0259
GNO [Li et al., 2021c]	0.0346	-	0.0332	0.0342	0.0369
LNO [Li et al., 2021c]	0.0520	-	0.0461	0.0445	-
MGNO [Li et al., 2021c]	0.0416	-	0.0428	0.0428	0.0420
FNO-2d [Li et al., 2021c]	0.0108	-	0.0109	0.0109	0.0098
CNN (ours)	0.0404	0.0495	0.0613	0.0813	0.1150
FNO-2d (reproduced)	0.0096	0.0092	0.0091	0.0091	0.0091
CNN XD (ours)	**0.0065**	**0.0065**	**0.0065**	**0.0071**	**0.0066**
FNO-2d XD (ours)	0.0082	0.0079	0.0077	0.0076	0.0074

Figure 10.10: Training curves (dotted) and test curves (solid) on Darcy Flow at resolution 141, showing better generalization of XD-operations.

Table 10.13: Architecture optimizer settings on for our protein folding experiments, across different ResNet depths. Note that the same step-size is used throughout since the backbone has no step-size schedule.

search space	optimizer	step-size	warmup epochs
ResNet-4 XD	Adam	1E-4	2
ResNet-6 XD	Momentum(0.99)	1E-4	2
ResNet-10 XD	Momentum(0.99)	1E-3	2
ResNet-18 XD	Momentum(0.9)	5E-4	2
ResNet-34 XD	Momentum(0.9)	5E-4	2

Table 10.14: Test MAE_8 of the Dilated ResNet of Adhikari [2020], compared to a standard ResNet backbone and XD-operations applied to ResNet. Results are averaged over 3 trials.

Method	depth $= 4$	depth $= 6$	depth $= 10$	depth $= 18$	depth $= 34$
ResNet	5.99 ± 0.43	5.30 ± 0.11	4.91 ± 0.25	4.80 ± 0.07	4.66 ± 0.15
Dilated ResNet	6.04 ± 0.33	5.49 ± 0.02	4.64 ± 0.08	4.59 ± 0.22	4.50 ± 0.13
ResNet XD	$\mathbf{5.59 \pm 0.09}$	$\mathbf{4.59 \pm 0.17}$	$\mathbf{4.25 \pm 0.16}$	$\mathbf{4.22 \pm 0.03}$	$\mathbf{4.00 \pm 0.07}$

10.C.3 Protein folding

For our protein folding experiments, our code is a PyTorch re-implementation of the PDNET code and setup [Adhikari, 2020] provided here: `https://github.com/ba-lab/pdnet`. As before, we use the same training routine and settings as the Dilated ResNet architecture

Table 10.15: Architecture optimizer settings on sequence modeling tasks. Note that the step-size is updated using the same schedule as the backbone.

task	optimizer	initial step-size	warmup epochs
Permuted MNIST	Adam	2E-4	0
JSB Chorales	Adam	2E-4	25
Nottingham	Adam	2E-3	0
Penn Treebank	Adam	2E-6	0

Table 10.16: XD-operations applied to TCNs compared to recent empirical results in sequence modeling. Our results are averages of three trials. Methods achieving within one deviation of the best performance are **bolded**.

Method (source)	Permuted MNIST* (error)	JSB Chorales (loss)	Nottingham (loss)	Penn Treebank (perplexity)
LSTM [Bai et al., 2018]	14.3	8.45	3.29	78.93
GRU [Bai et al., 2018]	12.7	8.43	3.46	92.48
RNN [Bai et al., 2018]	74.7	8.91	4.05	114.50
TCN backbone [Bai et al., 2018]	2.8	8.10	3.07	88.68
TrellisNet [Bai et al., 2019]	1.87	-	-	**54.19**
R-Transformer [Wang et al., 2020c]	-	-	**2.37**	84.38
HiPPO-LegS [Gu et al., 2020]	**1.7**	-	-	-
TCN backbone (reproduced)	2.89 ± 0.04	8.17 ± 0.01	2.97 ± 0.01	88.49 ± 0.31
TCN backbone XD (ours)	$\mathbf{1.75 \pm 0.11}$	$\mathbf{8.07 \pm 0.02}$	2.81 ± 0.05	84.11 ± 0.25
Undilated TCN (ours)	11.3 ± 2.1	8.16 ± 0.04	3.21 ± 0.02	94.30 ± 0.33
Undilated TCN XD (ours)	$\mathbf{1.77 \pm 0.10}$	$\mathbf{8.07 \pm 0.01}$	2.84 ± 0.02	85.04 ± 0.49

* We use depth $\mathbf{d} = (3, 3, 3)$ XD-operations for permuted MNIST experiments; elsewhere we use $(1, 3, 1)$. Results within a standard deviation of the best are **bolded**.

and only tune the architecture optimizer. We consider the following hyperparameters for the architecture optimizer: Adam vs. SGD (with or without momentum), learning rate, and number of warmup epochs. The final hyperparameters for each depth can be found in Table 10.13. Our ResNet backbone differs from Dilated ResNet in that its dilation rate is set to 1 in every convolutional layer. In Table 10.14, we present average MAE_8 on the PSICOV test set for each method at each depth.

10.C.4 Music modeling and sequence modeling

For our sequence modeling experiments we use the TCN code [Bai et al., 2018] provided here: `https://github.com/locuslab/TCN`. As before we use the same settings and training routine as the backbone for all tasks, tuning only the architecture optimizer. The specific settings are provided in Table 10.15. For both the baselines and XD-operations we use the same optimizer settings for both the dilated and undilated TCN backbones. In Table 10.16 we present

Table 10.17: Information about evaluation tasks in NAS-Bench-360 [Tu et al., 2022].

task name	datapoints	data dimension	prediction type	learning objective	expert architecture
CIFAR-100	60K	2D	point	classify natural images into 100 classes	DenseNet-BC [Huang et al., 2017]
Spherical	60K	2D	point	classify spherically projected images into 100 classes	S2CN [Cohen et al., 2018]
NinaPro	3956	2D	point	classify sEMG signals into 18 classes corresponding to hand gestures	Attention Model [Josephs et al., 2020]
FSD50K	51K	2D	point (multi-label)	classify sound events in log-mel spectrograms with 200 labels	VGG [Fonseca et al., 2021]
Darcy Flow	1100	2D	dense	predict the final state of a fluid from its initial conditions	FNO [Li et al., 2021c]
PSICOV	3606	2D	dense	predict pairwise distances between residuals from 2D protein sequence features	DEEPCON [Adhikari, 2019]
Cosmic	5250	2D	dense	predict propablistic maps to identify cosmic rays in telescope images	deepCR-mask [Zhang and Bloom, 2020]
ECG	330K	1D	point	detect atrial cardiac disease from a ECG recording (4 classes)	ResNet-1d [Hong et al., 2020]
Satellite	1M	1D	point	classify satellite image pixels' time series into 24 land cover types	ROCKET [Dempster et al., 2020]
DeepSEA	250K	1D	point (multi-label)	predict chromatin states and binding states of RNA sequences (36 classes)	DeepSEA [Zhou and Troyanskaya, 2015]

results for both music modeling and for two additional benchmarks—permuted MNIST and Penn Treebank—on which we see a similar pattern of XD-operations being able to recover and even beat (dilated) TCN performance starting from an undilated network.

10.C.5 NAS-Bench-360

Term clarification

Since we compare with a variety of methods, here we clarify some of the terms we use.

- Best-Automated (Figure 10.4): WRN, WRN-ASHA, DARTS, DenseNAS, AutoDL, and AMBER
- Hand-Designed (Figure 10.4): the expert architectures in Table 10.17
- AutoML: WRN-ASHA, DARTS, DenseNAS, AutoDL, and AMBER
- NAS: DARTS, DenseNAS, AutoDL, and AMBER
- WRN: WRN without hyperparameter tuning

Backbone network structure

Below we describe the CNN backbones we adopt for all 1D and 2D tasks. In both cases, we modify the original model's activation layer according to the learning objective, e.g. log softmax for classification and sigmoid for dense prediction. Both backbones also incorporate a dropout rate that we set to zero during search and tune as a hyperparameter when retraining.

Table 10.18: Task-specific DASH hyperparameters.

	CIFAR-100	Spherical	Darcy Flow	PSICOV	Cosmic
batch size	64	64	10	8	4
input size	(32, 32)	(60, 60)	(85, 85)	(128, 128)	(128, 128)
kernel sizes (K)	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$
dilations (D)	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$
loss (l)	cross entropy	cross entropy	ℓ_2	MSE	BCE
	NinaPro	FSD50K	Satellite	ECG	DeepSEA
batch size	128	128	256	1024	256
input size	(16, 52)	(96, 101)	46	1000	1000
kernel sizes (K)	$\{3, 5, 7, 9\}$	$\{3, 5, 7, 9\}$	$\{3, 7, 11, 15, 19\}$	$\{3, 7, 11, 15, 19\}$	$\{3, 7, 11, 15, 19\}$
dilations (D)	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$	$\{1, 3, 7, 15\}$
loss (l)	focal	BCE w. logits	cross entropy	cross entropy	BCE w. logits

2D tasks We use WRN-16-4 [Zagoruyko and Komodakis, 2016] as the backbone for 2D tasks, using the code here: `https://github.com/meliketoy/wide-resnet.pytorch`.

1D tasks We use a 1D WRN [Ismail Fawaz et al., 2020] for all 1D tasks, borrowing the implementation here: `https://github.com/okrasolar/pytorch-timeseries`. We modify the number of output channels from the original model's 64 to $\min\{4^{\text{num\_classes}//10+1}, 64\}$ to account for simpler tasks with fewer class labels. Separately, we also evaluate DASH with the Temporal Convolutional Network (TCN) backbone using the implementation of Bai et al. [2018]; we include the results for completeness in the last row of Table 10.23.

DASH pipeline hyperparameters

Search

- epochs: 100
- optimizer: `SGD(momentum=0.9, nesterov=True, weight_decay=5E-4)`
- model weight learning rate: 0.1 for point prediction tasks, 0.01 for dense tasks
- architecture parameter learning rate: 0.05 for point prediction tasks, 0.005 for dense tasks
- learning rate scheduling: decay by 0.2 at epoch 60
- gradient clipping threshold: 1
- softmax temperature: 1
- subsampling ratio: 0.2

Hyperparameter tuning

- epochs: 80
- configuration space: learning rate in $\{1E-1, 1E-2, 1E-3\}$, weight decay in $\{5E-4, 5E-6\}$, momentum in $\{0.9, 0.99\}$, dropout in $\{0, 0.05\}$

Table 10.19: Errors of DASH and the baselines on NAS-Bench-360. Methods are grouped into three classes: non-automated, automated, and DASH. Results of DASH are averaged over three trials using the models obtained after the last retraining epoch.

	CIFAR-100 0-1 error(%)	Spherical 0-1 error(%)	Darcy Flow relative ℓ_2	PSICOV MAE_8	Cosmic 1-AUROC
WRN	23.35±0.05	85.77±0.71	0.073±0.001	3.84±0.053	0.24±0.015
Expert	**19.39±0.20**	67.41±0.76	0.008±0.001	3.35±0.14	**0.13±0.01**
Perceiver IO	70.04±0.44	82.57±0.19	0.24±0.01	8.06±0.06	0.48±0.01
WRN-ASHA	23.39±0.01	75.46±0.40	0.066±0.00	3.84±0.05	0.25±0.021
DARTS-GAEA	24.02±1.92	**48.23±2.87**	0.026±0.001	**2.94±0.13**	0.22±0.035
DenseNAS	25.98±0.38	72.99±0.95	0.10±0.01	3.84±0.15	0.38±0.038
AutoDL	-	-	0.049±0.005	6.73±0.73	0.49±0.004
BABY DASH	25.56±1.37	63.45±0.88	0.016±0.002	3.94±0.54	0.16±0.007
DASH	24.37±0.81	71.28±0.68	**0.0079±0.002**	3.30±0.16	0.19±0.02

	NinaPro 0-1 error (%)	FSD50K 1- mAP	ECG $1\text{-}F_1$	Satellite 0-1 error (%)	DeepSEA 1-AUROC
WRN	6.78±0.26	0.92±0.001	0.43±0.01	15.49±0.03	0.40±0.001
TCN	-	-	0.57±0.005	16.21±0.05	0.44±0.001
Expert	8.73±0.9	0.62±0.004	**0.28±0.00**	19.8±0.00	0.30±0.24
Perceiver IO	22.22±1.80	0.72±0.002	0.66±0.01	15.93±0.08	0.38±0.004
WRN-ASHA	7.34±0.76	0.91±0.03	0.43±0.01	15.84±0.52	0.41±0.002
DARTS-GAEA	17.67±1.39	0.94±0.02	0.34±0.01	12.51±0.24	0.36±0.02
DenseNAS	10.17±1.31	0.64±0.002	0.40±0.01	13.81±0.69	0.40±0.001
AMBER	-	-	0.67±0.015	12.97±0.07	0.68±0.01
BABY DASH	8.28±0.62	0.62±0.01	0.37±0.001	13.29±0.108	0.37±0.017
DASH	**6.60±0.33**	**0.60±0.008**	0.32±0.007	**12.28±0.5**	0.28±0.013
DASH-TCN	-	-	0.29±0.004	12.39±0.043	**0.24±0.012**

Table 10.20: Runtime of DASH on NAS-Bench-360 tasks, in NVIDIA V100 GPU-hours.

	CIFAR-100	Spherical	Darcy Flow	PSICOV	Cosmic	NinaPro	FSD50K	ECG	Satellite	DeepSEA
search	1.6	1.6	0.16	0.88	1.6	0.028	0.88	0.18	1.8	0.36
tune	0.15	0.25	1.6	0.64	0.055	0.16	0.88	0.28	0.4	1.6
retrain	0.77	3.16	3.5	14	5.1	0.11	27	0.83	4.3	8.3
total	2.5	5.0	5.3	15	6.8	0.30	29	1.3	6.5	10

Table 10.21: Time for one search epoch, in seconds.

	WRN	ConvNeXt
# param	3M	28M
DASH	151.3	80.5
mixed-weights	705.4	300.1
mixed-results	330.6	149.6

Table 10.22: Total runtime on ImageNet-1K, in hours.

	WRN	ConvNeXt
DASH search	24	13
DASH retrain	52	48
backbone train	16	41

Table 10.23: Prediction error on ImageNet-1K. Backbone results from Liu et al. [2022].

	WRN	ConvNeXt
vanilla backbone	37.56±0.14	17.9±0.0
DASH model	**34.12±0.21**	**16.42±0.15**

Retraining

- epochs: 200
- learning rate scheduling: for 2D tasks, decay by 0.2 at epoch 60, 120, 160; for 1D tasks, decay by 0.2 at epoch 30, 60, 90, 120, 160

10.C.6 ImageNet

Although vision objectives are not our main motivation, we find that DASH is backward compatible with large-scale image classification by testing it on ImageNet-1K with two backbones of distinct scales. Our results show that DASH generalizes to tasks with large input shape ($3 \times 224 \times 224$), dataset size (1.2M), and number of classes (1000). It improves the accuracy of the original models and searches efficiently regardless of the backbone used.

We evaluated WRN-16-4 from before along with ConvNeXt-T [Liu et al., 2022], a large-scale CNN that has performance on par with state-of-the-art Transformers, and performed experiments on four NVIDIA V100 GPUs. To demonstrate DASH's efficiency in Table 10.21 we present the per-epoch search time (forward and backward time in seconds) for three baselines over the search space $K = \{3, 5, 7, 9, 11\}$, $D = \{1, 3, 7\}$. A subset of 4096 images is used. We can see that DASH's efficiency holds for both backbones; although ConvNeXt has more parameters, it is searched faster than WRN as it has fewer convolution layers and applies downsampling to the input. Then in Table 10.22 we report DASH's runtime vs. the train-time of the vanilla backbone (in hours). We let DASH search for 10 epochs with subsampling ratio 0.2. (Re)training takes 50 and 100 epochs for WRN and ConvNeXt, respectively.

Lastly, in Table 10.23 we report the top-1 accuracy of the searched vs. original models to show DASH generalizes to large vision input. We trained ConvNeXt for 300 epochs. In general, DASH improves backbone performance by adopting task-specific kernels.

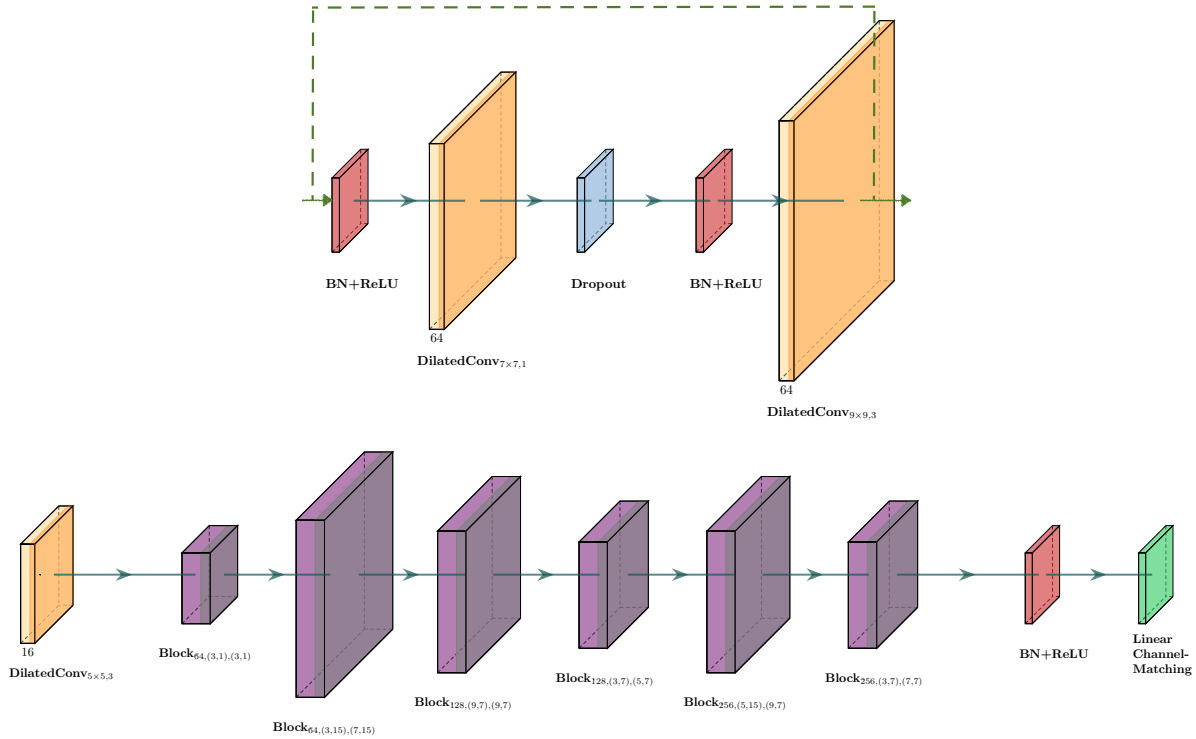

Figure 10.11: Visualization of the architecture DASH discovers on Darcy Flow, for which it generates a WRN-16-4 [Zagoruyko and Komodakis, 2016] for retraining. The network architecture consists of several residual blocks. For instance, the residual block with the structure in the top image can be denoted by $\text{Block}_{64,(7,1),(9,3)}$ (64 is the number of output channels and "BN" denotes the BatchNorm layer). Note that size of a convolutional layer in the figure is proportional to the kernel size but not the number of channels. The bottom image is an example network found by DASH on Darcy Flow; n.b. since Darcy Flow is a dense prediction task, the last layer is a channel-matching (permutation+linear+permutation) layer instead of a pooling+linear layer for classification.

10.D Searched architecture visualization

Lastly, we give two example networks searched by DASH to demonstrate that search spaces containing large kernels matter for diverse tasks. We visualize a 2D architecture discovered for Darcy Flow in Figure 10.11 and a 1D architecture discovered for DeepSEA in Figure 10.12.

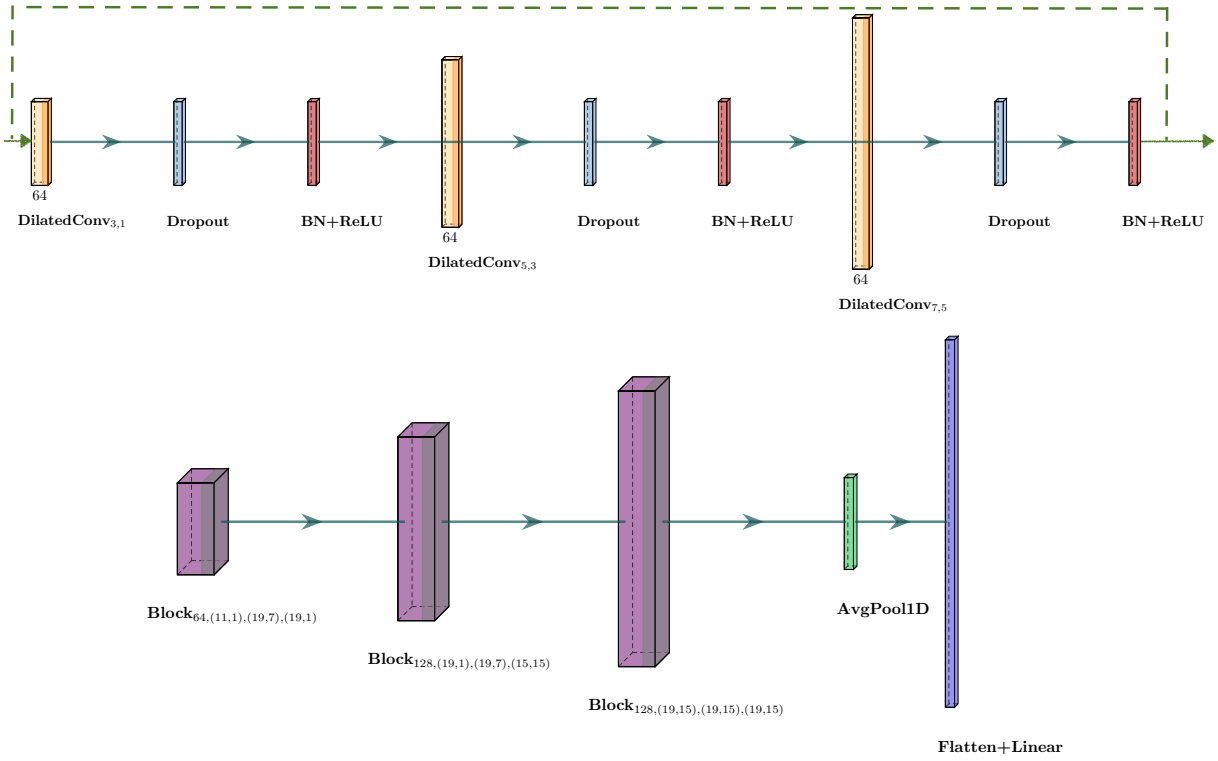

Figure 10.12: Visualization of the architecture DASH discovers on DeepSEA, for which it generates a 1D WRN [Ismail Fawaz et al., 2020] for retraining. The network architecture consists of several residual blocks. For instance, the residual block with the structure in the top image can be denoted by $\text{Block}_{64,(3,1),(5,3),(7,5)}$ (64 is the number of output channels and "BN" denotes the BatchNorm layer). The bottom image is an example network found by DASH on DeepSEA, from which we can see that large kernels are selected for during search.

Appendix A

Notation

Unless otherwise specified, $\tilde{\mathcal{O}}$ will be used to ignore logarithmic factors in standard asymptotic notation. We also use $\mathcal{O}_n(\cdot)$ (and $o_n(\cdot)$) to denote terms with constant (and sub-constant) dependence on n.

A.1 Sets

We use $\mathbb{Z}$ and $\mathbb{R}$ to denote the integers and reals, respectively, $\overline{\mathbb{R}}$ to denote the set of extended real numbers $\mathbb{R} \cup \{\pm\infty\}$, and $\mathbb{Z}_{>0}$, $\mathbb{R}_{>0}$, $\mathbb{Z}_{\geqslant 0}$, and $\mathbb{R}_{\geqslant 0}$ to denote their associated positive and nonnegative subsets. For any $\mathbf{x} \in \mathbb{R}^d$ and $R \geqslant 0$ we use $\mathbb{B}(\mathbf{x}, R)$ to denote the ℓ_2-ball of radius R around $\mathbf{x}$. We use $\mathcal{K}^\circ$ to denote the interior of a closed set $\mathcal{K}$ and ∂K to denote its boundary. Brackets $[n]$ are used to denote the sequence $(1, \ldots, n)$, and we will use standard bracket/parenthesis notations to denote intervals. We use $1_S : S \mapsto \{0, 1\}$ to denote the indicator function on the set S and $|S|$ to denote the number of entries.

A.2 Vectors and matrices

We use bolded lower-case letters to denote vectors, bolded upper-case letters to denote matrices, and brackets in subscripts to index into them (e.g. $\mathbf{x}_{[i]}$ is the ith element of a vector $\mathbf{x}$, $\mathbf{X}_{[i,j]}$ is the jth element in the ith row of a matrix $\mathbf{X}$, and $\mathbf{X}_{[i]}$ is its ith row); whether bracketing is being used to denote a set as in the previous section or to index into a vector will be clear from context. For both vectors and matrices we will use $\|\cdot\|_p$ to denote the entry-wise p-norm, and for the latter we will use $|||\cdot|||_p$ to denote the Schatten p-norm; thus on matrices $\|\cdot\|_2 = \|\cdot\|_F$ is the Frobenius norm, $\|\cdot\|_\infty = \|\cdot\|_{\max}$ is the max-norm, $|||\cdot|||_\infty$ is the spectral norm, and $|||\cdot|||_1 = \|\cdot\|_{\mathrm{Tr}}$ is the trace or nuclear norm. We use $\|\cdot\|_*$ to refer to the dual norm of $\|\cdot\|$. For any two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, $\mathbf{x} \odot \mathbf{y}$ will denote element-wise multiplication, $\frac{\mathbf{x}}{\mathbf{y}}$ will denote element-wise division, $\mathbf{x}^p$ will denote raising each element of $\mathbf{x}$ to the power p (with $\sqrt{\mathbf{x}} = \mathbf{x}^{\frac{1}{2}}$), and $\max\{\mathbf{x}, \mathbf{y}\}$ and $\min\{\mathbf{x}, \mathbf{y}\}$ will denote element-wise maximum and minimum, respectively.

We use $\mathbf{0}_n$, $\mathbf{1}_n$, $\mathbf{I}_n$, $\mathbf{0}_{m\times n}$, and $\mathbf{1}_{m\times n}$ to denote the n-dimensional zero vector, the n-dimensional all-ones vector, the $n \times n$ identity matrix, the $m \times n$ zero matrix, and the $m \times n$ all-ones matrix,

respectively. In cases where the dimension can be inferred from context we will us $\mathbf{e}_i$ to denote one-hot vectors, i.e. $\mathbf{e}_{i[j]} = 1_{i=j}$. In all cases we drop the subscript when the dimension can be inferred from context. Lastly, given an $n \times n$ matrix $\mathbf{A}$, we make use of the following additional notation:

- $\rho(\mathbf{A})$ denotes the spectral radius of $\mathbf{A}$
- $\mathbf{A} \succ \mathbf{0}$ and $\mathbf{A} \succeq \mathbf{0}$ mean that $\mathbf{A}$ is positive definite and positive semi-definite, respectively
- $\kappa(\mathbf{A}) = \|\mathbf{A}\|_\infty \|\mathbf{A}^{-1}\|_\infty$ denotes the condition number of $\mathbf{A} \succ \mathbf{0}$
- $\|\mathbf{x}\|_\mathbf{A} = \|\mathbf{A}^{\frac{1}{2}}\mathbf{x}\|_2$ is the energy norm of a vector $\mathbf{x} \in \mathbb{R}^n$ associated with $\mathbf{A} \succ \mathbf{0}$
- $\|\mathbf{X}\|_\mathbf{A} = \|\mathbf{A}^{\frac{1}{2}}\mathbf{X}\mathbf{A}^{-\frac{1}{2}}\|_\infty$ is the energy norm of a matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$ associated with $\mathbf{A} \succ \mathbf{0}$

A.3 Probability

We will use $\triangle_n = \{\mathbf{x} \in \mathbb{R}^n_{\geqslant 0} : \|\mathbf{x}\|_1 = 1\}$ to denote the n-dimensional probability simplex, $x \sim \mathrm{Unif}(S)$ or $x \sim \mathrm{Unif}\, S$ to denote sampling uniformly from a set S, and $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to denote sampling from a normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma} \succ \mathbf{0}$; the isotropic case where $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$ for some $\sigma^2 \in \mathbb{R}_{>0}$ we be denoted by $\mathcal{N}(\boldsymbol{\mu}, \sigma^2)$. We will also use $\mathrm{Lap}(x)$, $\mathrm{Ber}(p)$, and $\mathrm{Beta}(a, b)$ as shorthands for sampling random variables from a Laplace distribution with scale x, a Bernoulli distribution with probability p, and a Beta distribution with parameters a and b, respectively.

Given a probability measure $\mu : S \mapsto \mathbb{R}_{\geqslant 0}$ on a set S, we use $\mu(S') = \int_{S'} \mu$ to denote the probability it assigns to any subset $S' \subset S$. We will use $\mathbb{E}[x]$ to denote the expectation of a random variable x, $\mathrm{Var}(x)$ to denote its variance, $\Pr(A)$ to denote the probability of an event A, and $D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q})$ to denote the KL-divergence between probability vectors $\mathbf{p}, \mathbf{q} \in \triangle_n$.

Appendix B

Online convex optimization

Throughout this appendix we assume all subsets are convex and in a finite-dimensional real vector space $\mathbb{V}$ with inner product $\langle \cdot, \cdot \rangle$ unless explicitly stated. For sequences of scalars $\sigma_1, \ldots, \sigma_T \in \mathbb{R}$ we will use the notation $\sigma_{1:t}$ to refer to the sum of the first t of them. In the online learning setting, we will use the shorthand ∇_t to denote the subgradient of $\ell_t : \Theta \mapsto \mathbb{R}$ evaluated at action $\boldsymbol{\theta}_t \in \Theta$. We will use $\mathrm{Proj}_S(\cdot)$ to be the projection to any convex subset S.

B.1 Basic function classes

We start with some basic properties of functions.

Definition B.1.1. Consider a closed and convex subset $\mathcal{X} \subset \mathbb{V}$. For any $\alpha > 0$ and norm $\|\cdot\| : \mathcal{X} \mapsto \mathbb{R}_{>0}$ an everywhere-subdifferentiable function $f : \mathcal{X} \mapsto \overline{\mathbb{R}}$ is called α**-strongly-convex** w.r.t. $\|\cdot\|$ if $\forall\, \mathbf{x}, \mathbf{y} \in \mathcal{X}$ we have

$$f(\mathbf{y}) \geqslant f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\alpha}{2} \|\mathbf{y} - \mathbf{x}\|^2 \tag{B.1}$$

Definition B.1.2. Consider a closed and convex subset $\mathcal{X} \subset \mathbb{V}$. For any $\beta > 0$ and norm $\|\cdot\| : \mathcal{X} \mapsto \mathbb{R}_{>0}$ a continuously-differentiable function $f : \mathcal{X} \mapsto \mathbb{R}$ is called β**-strongly-smooth** w.r.t. $\|\cdot\|$ if $\forall\, \mathbf{x}, \mathbf{y} \in \mathcal{X}$ we have

$$f(\mathbf{y}) \leqslant f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\beta}{2} \|\mathbf{y} - \mathbf{x}\|^2 \tag{B.2}$$

Definition B.1.3. An everywhere sub-differentiable function $f : \mathcal{X} \mapsto \mathbb{R}$ is γ**-exp-concave** if $\exp(-\gamma f(\cdot))$ is concave. For $\mathcal{X} \subset \mathbb{R}$ we have that $\frac{\partial_{xx} f(x)}{(\partial_x f(x))^2} \geqslant \gamma \;\forall\, x \in \mathcal{X} \implies f$ is γ-exp-concave.

B.2 The Bregman divergence

We next turn to defining the **Bregman divergence**, a generalized notion of distance used in optimization theory, and stating several of its properties:

Definition B.2.1 (Bregman [1967], Banerjee et al. [2005])**.** Let $\mathcal{X}$ be a closed and convex subset of $\mathbb{V}$. The **Bregman divergence** induced by a strictly convex, continuously-differentiable **distance-generating function (DGF)**[1] $\phi : \mathcal{X} \mapsto \overline{\mathbb{R}}$ is

$$\mathcal{B}_\phi(\mathbf{x}||\mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) - \langle \nabla\phi(\mathbf{y}), \mathbf{x} - \mathbf{y}\rangle \; \forall \, \mathbf{x}, \mathbf{y} \in \mathcal{X} \tag{B.3}$$

By definition, the Bregman divergence satisfies the following properties:

1. $\mathcal{B}_\phi(\mathbf{x}||\mathbf{y}) \geqslant 0 \; \forall \; x, y \in \mathcal{X}$ and $\mathcal{B}_\phi(\mathbf{x}||\mathbf{y}) = 0 \iff x = y$.
2. If ϕ is α-strongly-convex w.r.t. norm $\|\cdot\|$ then so is $\mathcal{B}_\phi(\cdot||\mathbf{y}) \; \forall \; \mathbf{y} \in \mathcal{X}$. Furthermore, $\mathcal{B}_\phi(\mathbf{x}||\mathbf{y}) \geqslant \frac{\alpha}{2}\|\mathbf{x} - \mathbf{y}\|^2 \; \forall \, \mathbf{x}, \mathbf{y} \in \mathcal{X}$.
3. If ϕ is β-strongly-smooth w.r.t. norm $\|\cdot\|$ then so is $\mathcal{B}_\phi(\cdot||\mathbf{y}) \; \forall \; \mathbf{y} \in \mathcal{X}$. Furthermore, $\mathcal{B}_\phi(\mathbf{x}||\mathbf{y}) \leqslant \frac{\beta}{2}\|\mathbf{x} - \mathbf{y}\|^2 \; \forall \, \mathbf{x}, \mathbf{y} \in \mathcal{X}$.

A crucial property of Bregman divergences is that the minimizer of a (weighted) sum of Bregman divergences from a set of points is their (weighted) mean:

Claim B.2.1. Let $\phi : \mathcal{X} \mapsto \mathbb{R}$ be a strictly convex function on $\mathcal{X} \subset \mathbb{V}$, $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ be a sequence satisfying $\alpha_{1:n} > 0$, and $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$. Then

$$\bar{\mathbf{x}} = \frac{1}{\alpha_{1:n}} \sum_{i=1}^{n} \alpha_i \mathbf{x}_i = \arg\min_{\mathbf{y} \in S} \sum_{i=1}^{n} \alpha_i \mathcal{B}_\phi(\mathbf{x}_i||\mathbf{y}) \tag{B.4}$$

Proof. $\forall \; \mathbf{y} \in \mathcal{X}$ we have

$$\begin{aligned}
&\sum_{i=1}^{n} \alpha_i \left(\mathcal{B}_\phi(\mathbf{x}_i||\mathbf{y}) - \mathcal{B}_\phi(\mathbf{x}_i||\bar{\mathbf{x}})\right) \\
&\quad = \sum_{i=1}^{n} \alpha_i \left(\phi(\mathbf{x}_i) - \phi(\mathbf{y}) - \langle \nabla\phi(\mathbf{y}), \mathbf{x}_i - y\rangle - \phi(\mathbf{x}_i) + \phi(\bar{\mathbf{x}}) + \langle \nabla\phi(\bar{\mathbf{x}}), \mathbf{x}_i - \bar{\mathbf{x}}\rangle\right) \\
&\quad = \left(\phi(\bar{\mathbf{x}}) - \phi(\mathbf{y}) + \langle \nabla\phi(\mathbf{y}), \mathbf{y}\rangle\right) \alpha_{1:n} + \sum_{i=1}^{n} \alpha_i \left(-\langle \nabla\phi(\bar{\mathbf{x}}), \bar{\mathbf{x}}\rangle + \langle \nabla\phi(\bar{\mathbf{x}}) - \nabla\phi(\mathbf{y}), \mathbf{x}_i\rangle\right) \\
&\quad = \left(\phi(\bar{\mathbf{x}}) - \phi(\mathbf{y}) - \langle \nabla\phi(\mathbf{y}), \bar{\mathbf{x}} - \mathbf{y}\rangle\right) \alpha_{1:n} \\
&\quad = \alpha_{1:n} \mathcal{B}_\phi(\bar{\mathbf{x}}||\mathbf{y})
\end{aligned} \tag{B.5}$$

By Definition B.2.1 the last expression has a unique minimum at $\mathbf{y} = \bar{\mathbf{x}}$. □

Related to this is the following property relating the average Bregman divergence from a mean vector to evaluations of the generating function:

Claim B.2.2. Let $\psi : \mathcal{K} \mapsto \mathbb{R}$ be a strictly-convex function with Bregman divergence $\mathcal{B}(\cdot||\cdot)$ over a convex set $\mathcal{K} \subset \mathbb{R}^d$ containing points $\mathbf{x}_1, \ldots, \mathbf{x}_T$. Then their mean $\bar{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}_t$ satisfies

$$\sum_{t=1}^{T} \mathcal{B}(\mathbf{x}_t||\bar{\mathbf{x}}) = \sum_{t=1}^{T} \psi(\mathbf{x}_t) - \psi(\bar{\mathbf{x}}) \tag{B.6}$$

[1] Also sometimes called a *regularizer*.

Proof.

$$\begin{aligned}\sum_{t=1}^{T}\mathcal{B}(\mathbf{x}_t||\bar{\mathbf{x}}) &= \sum_{t=1}^{T}\psi(\mathbf{x}_t)-\psi(\bar{\mathbf{x}})-\langle\nabla\psi(\bar{\mathbf{x}}),\mathbf{x}_t-\bar{\mathbf{x}}\rangle \\ &= \sum_{t=1}^{T}\psi(\mathbf{x}_t)-\psi(\bar{\mathbf{x}})-\langle\nabla\psi(\bar{\mathbf{x}}),\sum_{t=1}^{T}\mathbf{x}_t-\bar{\mathbf{x}}\rangle = \sum_{t=1}^{T}\psi(\mathbf{x}_t)-\psi(\bar{\mathbf{x}})\end{aligned} \tag{B.7}$$

□

B.3 Algorithms

Here we provide a review of the online learning algorithms we use. Recall that in this setting our goal is minimizing regret:

Definition B.3.1. The **regret** of an agent playing actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ on a sequence of loss functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\in[T]}$ is

$$\text{Regret} = \sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}_t) - \min_{\boldsymbol{\theta}\in\Theta}\sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}) \tag{B.8}$$

Within-task our focus is on two closely related meta-algorithms, Follow-the-Regularized-Leader (FTRL) and (lazy linearized) Online Mirror Descent (OMD).

Definition B.3.2. Given a strictly convex function $R : \Theta \mapsto \mathbb{R}$, starting point $\phi \in \Theta$, fixed learning rate $\eta > 0$, and a sequence of functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\geqslant 1}$, **Follow-the-Regularized Leader (**$\texttt{FTRL}^{(R)}_{\boldsymbol{\phi},\eta}$**)** plays

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}\in\Theta}\mathcal{B}_R(\boldsymbol{\theta}||\boldsymbol{\phi}) + \eta\sum_{s<t}\ell_s(\boldsymbol{\theta}) \tag{B.9}$$

Definition B.3.3. Given a strictly convex function $R : \Theta \mapsto \mathbb{R}$, starting point $\boldsymbol{\phi} \in \Theta$, fixed learning rate $\eta > 0$, and a sequence of functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\geqslant 1}$, **lazy linearized Online Mirror Descent (**$\texttt{OMD}^{(R)}_{\boldsymbol{\phi},\eta}$**)** plays

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}\in\Theta}\mathcal{B}_R(\boldsymbol{\theta}||\boldsymbol{\phi}) + \eta\sum_{s<t}\langle\nabla_s,\boldsymbol{\theta}\rangle \tag{B.10}$$

These formulations make the connection between the two algorithms—their equivalence in the linear case $\ell_s(\cdot) = \langle\nabla_s,\cdot\rangle$—very explicit. There exists a more standard formulation of OMD that is used to highlight its generalization of OGD—the case of $R(\cdot) = \frac{1}{2}\|\cdot\|_2^2$—and the fact that the update is carried out in the dual space induced by R [Hazan, 2015, Section 5.3].[2] However, we will only need the following regret bound satisfied by both [Shalev-Shwartz, 2011, Theorems 2.11 and 2.15]

[2]Mirror descent can also be formulated in the *offline* setting, c.f. Section 9.2.1.

Theorem B.3.1. Let $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\in[T]}$ be a sequence of convex functions that are G_t-Lipschitz w.r.t. $\|\cdot\|$ and let $R : \Theta \mapsto \mathbb{R}$ be 1-strongly-convex. Then the regret of both $\texttt{FTRL}^{(R)}_{\eta,\boldsymbol{\phi}}$ and $\texttt{OMD}^{(R)}_{\eta,\boldsymbol{\phi}}$ is bounded by

$$\frac{\mathcal{B}_R(\boldsymbol{\theta}^*||\boldsymbol{\phi})}{\eta} + \eta G^2 T \tag{B.11}$$

for all $\boldsymbol{\theta}^* \in \Theta$ and $G^2 \geqslant \frac{1}{T}\sum_{t=1}^{T} G_t^2$.

We next review the online algorithms we use for the meta-update. The main requirement here is logarithmic regret guarantees for the case of strongly convex loss functions, which is satisfied by two well-known algorithms:

Definition B.3.4. Given a sequence of strictly convex functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\geqslant 1}$, **Follow-the-Leader (FTL)** plays arbitrary $\boldsymbol{\theta}_1 \in \Theta$ and for $t > 1$ plays

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{s<t} \ell_s(\boldsymbol{\theta}) \tag{B.12}$$

Definition B.3.5. Given a sequence of functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\geqslant 1}$ that are α_t-strongly-convex w.r.t. $\|\cdot\|_2$, **Adaptive OGD (AOGD)** plays arbitrary $\boldsymbol{\theta}_1 \in \Theta$ and for $t > 1$ plays

$$\boldsymbol{\theta}_{t+1} = \mathrm{Proj}_{\Theta}\left(\boldsymbol{\theta}_t - \frac{1}{\alpha_{1:t}}\nabla f(\boldsymbol{\theta}_t)\right) \tag{B.13}$$

Kakade and Shalev-Shwartz [2008, Theorem 2] and Bartlett et al. [2008, Theorem 2.1] provide for FTL and AOGD, respectively, the following regret bound:

Theorem B.3.2. Let $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\in[T]}$ be a sequence of convex functions that are G_t-Lipschitz and α_t-strongly-convex w.r.t. $\|\cdot\|$. Then the regret of both FTL and AOGD is bounded by

$$\frac{1}{2}\sum_{t=1}^{T} \frac{G_t^2}{\alpha_{1:t}} \tag{B.14}$$

Finally, we state the EWOO algorithm due to Hazan et al. [2007]. While difficult to run in high-dimensions, we will be running this method in single dimensions, when computing it requires only one integral.

Definition B.3.6. Given a sequence of γ-exp-concave functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}$, **Exponentially Weighted Online Optimization (EWOO)** plays

$$\boldsymbol{\theta}_t = \frac{\int_\Theta \boldsymbol{\theta} \exp(-\gamma \sum_{s<t} \ell_s(\boldsymbol{\theta}))d\boldsymbol{\theta}}{\int_\Theta \exp(-\gamma \sum_{s<t} \ell_s(\boldsymbol{\theta}))d\boldsymbol{\theta}} \tag{B.15}$$

Hazan et al. [2007, Theorem 7] provide the following guarantee for EWOO, which is notable for its lack of explicit dependence on the Lipschitz constant.

Theorem B.3.3. Let $\{\ell_t : \Theta \mapsto \mathbb{R}\}$ be a sequence of γ-exp-concave functions. Then the regret of EWOO is bounded by

$$\frac{d}{\gamma}(1 + \log(T+1)) \tag{B.16}$$

B.4 Online-to-batch conversion

As we are also interested in distributional meta-learning, we discuss some techniques for converting regret guarantees into generalization bounds, which are usually named *online-to-batch conversions*. We first state some standard results.

Proposition B.4.1. If a sequence of convex loss functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\in[T]}$ drawn i.i.d. from some distribution $\mathcal{D}$ is given to an online algorithm with regret bound $U(T)$ that generates a sequence of actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ then

$$\mathbb{E}_{\mathcal{D}^T}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*) + \frac{U(T)}{T} \tag{B.17}$$

for $\bar{\boldsymbol{\theta}} = \frac{1}{T}\boldsymbol{\theta}_{1:T}$ and any $\boldsymbol{\theta}^* \in \Theta$.

Proof. Applying Jensen's inequality yields

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}^T}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\bar{\boldsymbol{\theta}}) &\leqslant \frac{1}{T}\mathbb{E}_{\mathcal{D}^T}\sum_{t=1}^{T}\mathbb{E}_{\ell'_t\sim\mathcal{D}}\ell'_t(\boldsymbol{\theta}_t) \\
&= \frac{1}{T}\mathbb{E}_{\{\ell_t\}\sim\mathcal{D}^T}\left(\sum_{t=1}^{T}\mathbb{E}_{\ell'_t\sim\mathcal{D}}\ell'_t(\boldsymbol{\theta}_t) - \ell_t(\boldsymbol{\theta}_t)\right) + \frac{1}{T}\mathbb{E}_{\{\ell_t\}\sim\mathcal{D}^T}\left(\sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}_t)\right) \\
&\leqslant \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\{\ell_s\}_{s<t}\sim\mathcal{D}^{t-1}}\left(\mathbb{E}_{\ell'_t\sim\mathcal{D}}\ell'_t(\boldsymbol{\theta}_t) - \mathbb{E}_{\ell_t\sim\mathcal{D}}\ell_t(\boldsymbol{\theta}_t)\right) + \frac{U(T)}{T} + \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*) \\
&= \frac{U(T)}{T} + \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*)
\end{aligned} \tag{B.18}$$

where we used the fact that $\boldsymbol{\theta}_t$ only depends on $\ell_1, \ldots, \ell_{t-1}$. □

For nonnegative and bounded losses we have the following fact [Cesa-Bianchi et al., 2004, Proposition 1]:

Proposition B.4.2. If a sequence of loss functions $\{\ell_t : \Theta \mapsto [0,1]\}_{t\in[T]}$ drawn i.i.d. from some distribution $\mathcal{D}$ is given to an online algorithm that generates a sequence of actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ then

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}_t) \leqslant \frac{1}{T}\sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}_t) + \sqrt{\frac{2}{T}\log\frac{1}{\delta}} \qquad \text{w.p. } 1-\delta \tag{B.19}$$

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}_t) \geqslant \frac{1}{T}\sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}_t) - \sqrt{\frac{2}{T}\log\frac{1}{\delta}} \qquad \text{w.p. } 1-\delta \tag{B.20}$$

Note that Cesa-Bianchi et al. [2004] only prove the first inequality; the second follows via the same argument but applying the symmetric version of the Azuma-Hoeffding inequality [Azuma, 1967]. The inequalities above can be easily used to derive the following competitive bounds:

Corollary B.4.1. If a sequence of loss functions $\{\ell_t : \Theta \mapsto [0,1]\}_{t\in[T]}$ drawn i.i.d. from some distribution $\mathcal{D}$ is given to an online algorithm with regret bound $U(T)$ that generates a sequence of actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ then

$$\mathbb{E}_{t\sim\mathcal{U}[T]}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}_t) \leqslant \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*) + \frac{U(T)}{T} + \sqrt{\frac{8}{T}\log\frac{1}{\delta}} \qquad \text{w.p. } 1-\delta \tag{B.21}$$

for any $\boldsymbol{\theta}^* \in \Theta$. If the losses are also convex then for $\bar{\boldsymbol{\theta}} = \frac{1}{T}\theta_{1:T}$ we have

$$\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*) + \frac{U(T)}{T} + \sqrt{\frac{8}{T}\log\frac{1}{\delta}} \qquad \text{w.p. } 1-\delta \tag{B.22}$$

Proof. By Proposition B.4.2 we have

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}_t) \leqslant \frac{1}{T}\sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}^*) + \frac{U(T)}{T} + \sqrt{\frac{2}{T}\log\frac{1}{\delta}} \leqslant \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*) + \frac{U(T)}{T} + \sqrt{\frac{8}{T}\log\frac{1}{\delta}} \tag{B.23}$$

Apply linearity of expectations to get the first inequality and Jensen's inequality to get the second. □

The following lemma provides a statement of the conversion in a sample complexity (rather than statistical risk) formulation:

Lemma B.4.1. Suppose an online learner has regret bound $U(T)$ on sequences of convex losses $\ell_{\mathbf{y}_1}, \ldots, \ell_{\mathbf{y}_T} : \mathcal{X} \mapsto [0, B]$ whose data $\mathbf{y}_t$ are drawn i.i.d. from some distribution $\mathcal{D}$. If $\mathbf{x}_1, \ldots, \mathbf{x}_T$ are the actions of the online learner, $\hat{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}_t$ is their average, and $T = \Omega\left(T_\varepsilon + \frac{B^2}{\varepsilon^2}\log\frac{1}{\delta}\right)$ for $T_\varepsilon = \min_{2U(T')\leqslant\varepsilon T'} T'$, then w.p. $\geqslant 1-\delta$ we have $\mathbb{E}_{\mathbf{y}\sim\mathcal{D}}\ell_{\mathbf{y}}(\hat{\mathbf{x}}) \leqslant \min_{\mathbf{x}\in\mathcal{X}}\mathbb{E}_{\mathbf{y}\sim\mathcal{D}}\ell_{\mathbf{y}}(\mathbf{x}) + \varepsilon$.

Proof. Apply Jensen's inequality, Cesa-Bianchi et al. [2004, Proposition 1], the regret bound, and Hoeffding's bound:

$$\begin{aligned}
\mathbb{E}_{\mathbf{y}}\ell_{\mathbf{y}}(\hat{\mathbf{x}}) \leqslant \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\mathbf{y}}\ell_{\mathbf{y}}(\mathbf{x}_t) &\leqslant \frac{1}{T}\sum_{t=1}^{T}\ell_{\mathbf{y}_t}(\mathbf{x}_t) + B\sqrt{\frac{2}{T}\log\frac{2}{\delta}} \\
&\leqslant \min_{\mathbf{x}\in\mathcal{X}}\frac{1}{T}\sum_{t=1}^{T}\ell_{\mathbf{y}_t}(\mathbf{x}) + \frac{U(T)}{T} + B\sqrt{\frac{2}{T}\log\frac{2}{\delta}} \\
&\leqslant \min_{\mathbf{x}\in\mathcal{X}}\mathbb{E}_{\mathbf{y}}\ell_{\mathbf{y}}(\mathbf{x}) + \frac{U(T)}{T} + 2B\sqrt{\frac{2}{T}\log\frac{2}{\delta}}
\end{aligned} \tag{B.24}$$

□

B.4.1 Strongly convex losses

We also discuss some stronger guarantees for certain classes of loss functions. The first, due to Kakade and Tewari [2008, Theorem 2], yields faster rates for strongly convex losses:

Theorem B.4.1. Let $\mathcal{D}$ be some distribution over loss functions $\ell : \Theta \mapsto [0, B]$ for some $B > 0$ that are G-Lipschitz w.r.t. $\|\cdot\|$ for some $G > 0$ and α-strongly-convex w.r.t $\|\cdot\|$ for some $\alpha > 0$. If a sequence of loss functions $\{\ell_t\}_{t\in[T]}$ is drawn i.i.d. from $\mathcal{D}$ and given to an online algorithm with regret bound $U(T)$ that generates a sequence of actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ then w.p. $1 - \delta$ we have for $\bar{\boldsymbol{\theta}} = \frac{1}{T}\boldsymbol{\theta}_{1:T}$ and any $\boldsymbol{\theta}^* \in \Theta$ that

$$\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*) + \frac{U(T)}{T} + \frac{4G}{T}\sqrt{\frac{U(T)}{\alpha}\log\frac{4\log T}{\delta}} + \frac{\max\{16G^2, 6\alpha B\}}{\alpha T}\log\frac{4\log T}{\delta} \tag{B.25}$$

B.4.2 Self-bounding losses

We can also obtain a data-dependent bound using a result of Zhang [2005] under a self-bounding property. Cesa-Bianchi and Gentile [2005, Proposition 2] show a similar but less general result.

Definition B.4.1. A distribution $\mathcal{D}$ over $\ell : \Theta \mapsto \mathbb{R}$ has ρ**-self-bounding** losses if $\forall\ \theta \in \Theta$ we have

$$\rho\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}) \geqslant \mathbb{E}_{\ell\sim\mathcal{D}}(\ell(\boldsymbol{\theta}) - \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\theta))^2 \tag{B.26}$$

Theorem B.4.2. Let $\mathcal{D}$ be some distribution over ρ-self-bounding convex loss functions $\ell : \Theta \mapsto [-1, 1]$ for some $\rho > 0$. If a sequence of loss functions $\{\ell_t\}_{t\in[T]}$ is drawn i.i.d. from $\mathcal{D}$ and given to an online algorithm with bounded regret that generates a sequence of actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ then w.p. $1 - \delta$ we have

$$\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \frac{L(T)}{T} + \sqrt{\frac{2\rho\max\{0, \bar{L}_T\}}{T}\log\frac{1}{\delta}} + \frac{3\rho + 2}{T}\log\frac{1}{\delta} \tag{B.27}$$

where $\bar{\boldsymbol{\theta}} = \frac{1}{T}\boldsymbol{\theta}_{1:T}$ and $L(T) = \sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}_t)$ is the average loss suffered by the agent.

Proof. Apply Jensen's inequality and Zhang [2005, Theorem 4]. □

Note that nonnegative 1-bounded convex losses satisfy the conditions of Theorem B.4.2 with $\rho = 1$. However, we are interested in a different result that can yield a data-dependent competitive bound:

Corollary B.4.2. Let $\mathcal{D}$ be some distribution over convex loss functions $\ell : \Theta \mapsto [0, 1]$ such that the functions $\ell(\boldsymbol{\theta}) - \ell(\theta^*)$ are ρ-self-bounded for some $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}\in\Theta}\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta})$. If a sequence of loss functions $\{\ell_t\}_{t\in[T]}$ is drawn i.i.d. from $\mathcal{D}$ and given to an online algorithm with regret bound $U(T)$ that generates a sequence of actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ then w.p. $1 - \delta$ we have

$$\mathbb{E}_{\ell\sim\mathcal{D}}\ell(\bar{\boldsymbol{\theta}}) \leqslant \mathbb{E}_{\ell\sim\mathcal{D}}\ell(\boldsymbol{\theta}^*) + \frac{U(T)}{T} + \frac{1}{T}\sqrt{2\rho U(T)\log\frac{1}{\delta}} + \frac{3\rho + 2}{T}\log\frac{1}{\delta} \tag{B.28}$$

where $\bar{\boldsymbol{\theta}} = \frac{1}{T}\boldsymbol{\theta}_{1:T}$ and $\mathcal{E}^* = \arg\min_{\boldsymbol{\theta}\in\Theta}\mathbb{E}\ell(\boldsymbol{\theta})$.

Proof. Apply Theorem B.4.2 over the sequence of functions $\{\ell_t(\boldsymbol{\theta}) - \ell_t(\boldsymbol{\theta}^*)\}_{t\in[T]}$ and by definition of regret substitute $\bar{L}_T = \frac{1}{T}\sum_{t=1}^{T}\ell_t(\boldsymbol{\theta}) - \ell_t(\boldsymbol{\theta}^*) \leqslant \frac{U(T)}{T}$. □

Zhang [2005, Lemma 7] shows that the conditions in Corollary B.4.2 are satisfied for $\rho = 4$ by least-squares regression.

B.5 Dynamic regret

Here we review several results for optimizing dynamic regret. We first define this quantity:

Definition B.5.1. The **dynamic regret** of an agent playing actions $\{\boldsymbol{\theta}_t \in \Theta\}_{t\in[T]}$ on a sequence of loss functions $\{\ell_t : \Theta \mapsto \mathbb{R}\}$ w.r.t. a sequence of reference parameters $\Psi = \{\boldsymbol{\psi}_t\}_{t\in[T]}$ is

$$\mathrm{Regret}_{\Psi} = \sum_{t=1}^{T} \ell_t(\boldsymbol{\theta}_t) - \sum_{t=1}^{T} \ell_t(\boldsymbol{\psi}_t) \tag{B.29}$$

Mokhtari et al. [2016, Corollary 1] show the following guarantee for OGD over strongly convex functions:

Theorem B.5.1. Let $\{\ell_t : \Theta \mapsto \mathbb{R}\}_{t\in[T]}$ be a sequence of α-strongly-convex, β-strongly-smooth, and G-Lipschitz functions w.r.t. $\|\cdot\|_2$. Then OGD with step-size $\eta \leqslant \frac{1}{\beta}$ achieves dynamic regret

$$\mathrm{Regret}_{\Psi} \leqslant \frac{GD}{1-\rho}\left(1 + \sum_{t=2}^{T} \|\boldsymbol{\psi}_t - \boldsymbol{\psi}_{t-1}\|_2\right) \tag{B.30}$$

w.r.t. reference sequence $\Psi = \{\boldsymbol{\psi}_t\}_{t\in[T]}$ for $\rho = \sqrt{1 - \frac{h\alpha}{\eta}}$ for any $h \in (0, 1]$ and D the ℓ_2-diameter of Θ.

Bibliography

Yasin Abbasi-Yadkori, Peter Bartlett, Victor Gabillon, Alan Malek, and Michal Valko. Best of both worlds: Stochastic & adversarial best-arm identification. In *Proceedings of the 31st Conference On Learning Theory*, 2018. 2.4, 2.4.2, 2.A.8

Jacob Abernethy, Peter L. Bartlett, Alexander Rakhlin, and Ambuj Tewari. Optimal strategies and minimax lower bounds for online convex games. Technical report, EECS Department, University of California, Berkeley, 2008a. 2.2

Jacob Abernethy, Elad Hazan, and Alexander Rakhlin. Competing in the dark: An efficient algorithm for bandit linear optimization. In *Proceedings of the International Conference on Computational Learning Theory*, 2008b. 2.4, 2.4.1, 2.4.3, 2.4.3, 2.4.3, 2.A.11, 2.A.10, 2.A.16, 2.A.10, 2.A.10

Jacob Abernethy, Chansoo Lee, and Ambuj Tewari. Fighting bandits with a new kind of smoothness. In *Advances in Neural Information Processing Systems*, 2015. 2.4, 2.4, 2.4.1, 2.4.2, 2.4.2, 7.3.3, 7.A, 7.A.1

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina annd Paul N. Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *Proceedings of the 9th International Conference on Learning Representations*, 2021. 3.3.1, 3.B

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Stefano Soatto, and Pietro Perona. Task2Vec: Task embedding for meta-learning. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 1.3.2, 8.3

Badri Adhikari. DEEPCON: Protein contact prediction using dilated convolutional neural networks with dropout. *Bioinformatics*, 36(2):470–477, 2019. 8.2.2, 10.3, 10.17

Badri Adhikari. A fully open-source framework for deep learning protein real-valued distances. *Scientific Reports*, 10, 2020. 10.2.4, 10.2.4, 10.14, 10.C.3

Naman Agarwal and Karan Singh. The price of differential privacy for online learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. 6.5.1, 6.5.2

Nir Ailon, Omer Leibovich, and Vineet Nair. Sparse linear networks with a fixed butterfly structure: Theory and practice. arXiv, 2020. 10.1

Youhei Akimoto, Shinichi Shirakawa, Nozomu Noshinari, Kento Uchida, Shota Saito, and Kouhei Nishida. Adaptive stochastic natural gradient method for one-shot neural architec-

ture search. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 8.A.3, 9.1, 9.4

Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *Proceedings of the 6th International Conference on Learning Representations*, 2018. 1

Maruan Al-Shedivat, Jennifer Gillenwater, Eric Xing, and Afshin Rostamizadeh. Federated learning via posterior averaging: A new perspective and practical algorithms. In *Proceedings of the 9th International Conference on Learning Representations*, 2021. 3.3.1, 3.B

Keivan Alizadeh vahid, Anish Prabhu, Ali Farhadi, and Mohammad Rastegari. Butterfly transform: An efficient FFT based neural architecture design. In *Proceedings of the IEEE Conference on Conference on Computer Vision and Pattern Recognition*, 2020. 10.1

Moray Allan and Christopher Williams. Harmonising chorales by probabilistic inference. In *Advances in Neural Information Processing Systems*, 2005. 10.2.4

Pierre Alquier, The Tien Mai, and Massimiliano Pontil. Regret bounds for lifelong learning. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017. 1.1, 1.A.2, 2.2.3

Ehsan Amid, Arun Ganesh, Rajiv Mathews, Swaroop Ramaswamy, Shuang Song, Thomas Steinke, Vinith M. Suriyakumar, Om Thakkar, and Abhradeep Thakurta. Public data-assisted mirror descent for private model training. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. 6.2.1

Kareem Amin, Travis Dick, Alex Kulesza, Andrés Muñoz Medina, and Sergei Vassilvitskii. Differentially private covariance estimation. In *Advances in Neural Information Processing Systems*, 2019. 6, 6, 6.2.3, 6.3.2, 6.3.2, 6.3.2, 6.B.3

Kareem Amin, Travis Dick, Mikhail Khodak, and Sergei Vassilvitskii. Private algorithms with private predictions. Theory and Practice of Differential Privacy Workshop, 2023. 0.3, 0

Ron Amit and Ron Meir. Meta-learning by adjusting priors based on extended PAC-Bayes theory. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. 1.1

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 2023. 4.3, 7.2

Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ML predictions for online algorithms. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 4, 4.1, 5, 5.1

Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. A regression approach to learning-augmented online algorithms. In *Advances in Neural Information Processing Systems*, 2021. 5.1

Oren Anava and Zohar S. Karnin. Multi-armed bandits: Competing with optimal sequences. In *Advances in Neural Information Processing Systems*, 2016. 2.4

Galen Andrew, Om Thakkar, H. Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. In *Advances in Neural Information Processing Sys-*

tems, 2021. 6.2.1

Hartwig Anzt, Edmond Chow, Jens Saak, and Jack Dongarra. Updating incomplete factorization preconditioners for model order reduction. *Numerical Algorithms*, 73:611–630, 2016. 7.3.1

Sohei Arisaka and Qianxiao Li. Principled acceleration of iterative numerical methods using machine learning. arXiv, 2023. 7.2

Sanjeev Arora and Anirudh Goyal. A theory for emergence of complex skills in language models. arXiv, 2023. 1.3.2

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of the 5th International Conference on Learning Representations*, 2017. 9.B.2

Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi. Minimax policies for combinatorial prediction games. In *Proceedings of the International Conference on Computational Learning Theory*, 2011. 2.4

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal of Computing*, 32:48–77, 2002. 2.4, 2.4, 2.4.1, 2.4.2, 7.3.3, 7.A

Peter Auer, Pratik Gajane, and Ronald Ortner. Adaptively tracking the best bandit arm with an unknown number of distribution changes. In *Proceedings of the 32nd Annual Conference on Learning Theory*, 2019. 2.4

Baruch Awerbuch and Robert D. Kleinberg. Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, 2004. 2.4.3

Owe Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1994. 7.3.5, 7.D.4

Mohammad Gheshlaghi Azar, Alessandro Lazaric, Emma Brunskill, et al. Sequential transfer in multi-armed bandit with finite set of models. In *Advances in Neural Information Processing Systems*, 2013. 2.4

MohammadJavad Azizi, Thang Duong, Yasin Abbasi-Yadkori, András György, Claire Vernade, and Mohammad Ghavamzadeh. Non-stationary bandits and meta-learning with a small set of optimal arms. arXiv, 2022. 2.4, 2.4.2, 2.4.2, 2.4.2

Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tôhoku Mathematical Journal*, 19:357–367, 1967. B.4

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv, 2018. 8.2.2, 8.2.2, 10.2.4, 10.3, 10.3.4, 10.3.4, 10.3.4, 10.16, 10.C.4, 10.C.5

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *Proceedings of the 7th International Conference on Learning Representations*, 2019. 8.2.2, 10.2.4, 10.16

Maria-Florina Balcan. Data-driven algorithm design. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, Cambridge, UK, 2021. 2.3,

2.A.5, 2.A.8, 4, 4.1, 1

Maria-Florina Balcan and Avrim Blum. Approximation algorithms and online mechanisms for item pricing. *Theory of Computing*, 3:179–195, 2007. 5.1

Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. Efficient representations for lifelong learning and autoencoding. In *Proceedings of the 28th Annual Conference on Learning Theory*, 2015. 1.1, 2.3.1

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *Proceedings of the 35th International Conference on Machine Learning*, 2018a. 4.1

Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science*, pages 603–614, 2018b. 2.3, 2.3.2, 2.3.1, 2.3.4, 2.3.4, 2.A.5, 2.A.5, 2.A.5, 2.A.5, 4, 4.1, 4, 5.5.2, 3, 7.2, 7.3.2, 7.4.1

Maria-Florina Balcan, Travis Dick, and Colin White. Data-driven clustering via parameterized Lloyd's families. In *Advances in Neural Information Processing Systems*, 2018c. 2.3, 2.3.4, 2.3.4, 2.3.8, 2.3.4, 2.A.5, 2.A.10, 2.A.5

Maria-Florina Balcan, Travis Dick, and Manuel Lang. Learning to link. In *Proceedings of the 9th International Conference on Learning Representations*, 2019. 2.3

Maria-Florina Balcan, Travis Dick, and Wesley Pegden. Semi-bandit optimization in the dispersed setting. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2020a. 2.3.4, 2.3.4, 2.A.5, 2.A.5, 2.A.5, 2.A.5, 2.A.5, 4.1, 7.2, 7.3.2

Maria-Florina Balcan, Travis Dick, and Dravyansh Sharma. Learning Piecewise-Lipschitz Functions in Changing Environments. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, pages 3567–3577, 2020b. 2.3.2, 2.3.4, 2.A.5

Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021a. 7.4.3

Maria-Florina Balcan, Mikhail Khodak, Dravyansh Sharma, and Ameet Talwalkar. Learning-to-learn non-convex piecewise-Lipschitz functions. In *Advances in Neural Information Processing Systems*, 2021b. 0.3, 0

Maria-Florina Balcan, Mikhail Khodak, Dravyansh Sharma, and Ameet Talwalkar. Provably tuning the ElasticNet across instances. In *Advances in Neural Information Processing Systems*, 2022. 7, 7.1, 7.2, 7.4.3

Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Advances in Neural Information Processing Systems*, 2020. 4, 5.3.2, 5.5.2, 6.2.2

Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005. B.2.1

Peter Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, 2017. 9.3.3

Peter Bartlett, Piotr Indyk, and Tal Wagner. Generalization bounds for data-driven numerical linear algebra. In *Proceedings of the 35th Annual Conference on Learning Theory*, 2022. 4.3, 7, 3, 7.2, 7.4.3, 7.E.4

Peter L. Bartlett, Elad Hazan, and Alexander Rakhlin. Adaptive online gradient descent. In *Advances in Neural Information Processing Systems*, 2008. 3.A, B.3

Raef Bassily, Mehryar Mohri, and Ananda Theertha Suresh. Private domain adaptation from a public source. arXiv, 2022. 6.2.1, 6.5

Soumya Basu, Branislav Kveton, Manzil Zaheer, and Csaba Szepesvári. No regrets for learning the prior in bandits. In *Advances in Neural Information Processing Systems*, 2021. 2.4

Manuel Baumann and Martin B. van Gijzen. Nested Krylov methods for shifted linear systems. *SIAM Journal on Scientific Computing*, 37:S90–S112, 2015. 7.3.1

Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000. 1.1, 2.1, 2.2.3

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31:167–175, 2003. 8.2.1, 9.2.1, 9.2.1

Stefania Bellavia, Valentina De Simone, Daniela di Serafina, and Benedetta Morini. Efficient preconditioner updates for shifted linear systems. *SIAM Journal on Scientific Computing*, 33: 1785–1809, 2011. 7.3.1

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. 9.3.1

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. 3.1, 3.3.2, 8.1, 9.3.1, 10.1

Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert E. Schapire. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011. 7.3.4

Alex Bie, Gautam Kamath, and Vikrant Singhal. Private estimation with public data. In *Advances in Neural Information Processing Systems*, 2022. 6, 6.2.1, 6.2.2, 6.2.2, 6.5

Sourav Biswas, Yihe Dong, Gautam Kamath, and Jonathan Ullman. CoinPress: Practical private mean and covariance estimation. In *Advances in Neural Information Processing Systems*, 2020. 6, 6.3.2, 6.7, 6.B.1

Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Caledon Press, 2012. 6.5.1

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 2.A.5

David M. Bradley and J. Andrew Bagnell. Differentiable sparse coding. In *Advances in Neural Information Processing Systems*, 2008. 9.2.1

Lev M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Math-*

ematics and Mathematical Physics, 7:200–217, 1967. 2.2.1, 9.2.1, B.2.1

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020. 1.3.2

Sébastien Bubeck. Five miracles of mirror descent, 2019. Lectures on Some Geometric Aspects of Randomized Online Decision Making. 9.2.1

Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012. 7.3.2

Brian Bullins, Elad Hazan, Adam Kalai, and Roi Livni. Generalize across tasks: Efficient algorithms for linear representation learning. In *Proceedings of the 30th International Conference on Algorithmic Learning Theory*, 2019. 1.1

Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Proceedings, Part I, of the 14th International Conference on Theory of Cryptography*, 2016. 6.B.1, 6.B.1

J. Burridge. A note on maximum likelihood estimation for regression models using grouped data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 43(1):41–45, 1981. 6.2.2, 6.6.1, 6.E.1

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proceedings of the 7th International Conference on Learning Representations*, 2019. 3.1, 8.A.3, 9, 9.1, 9.4, 9.5

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *Proceedings of the 8th International Conference on Learning Representations*, 2020. 10

Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. arXiv, 2018. 2.3, 2.2.4, 2.B.1, 3.1, 3.6

Rich Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997. 1.A.2

Rich Caruana, Shumeet Baluja, and Tom Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In *Advances in Neural Information Processing Systems*, 1995. 1.A.2

Giovanni Cavallanti, Nicolò Cesa-Bianchi, and Claudio Gentile. Linear algorithms for online multitask classification. *Journal of Machine Learning Research*, 11:2901–2934, 2010. 1.A.2

Leonardo Cella, Alessandro Lazaric, and Massimiliano Pontil. Meta-learning with stochastic linear bandits. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 2.4

Nicolò Cesa-Bianchi and Claudio Gentile. Improved risk tail bounds for on-line algorithms. In *Advances in Neural Information Processing Systems*, 2005. B.4.2

Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006. 2.A.6, 6.D.3, 7.3.2

Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of online learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004. 1.A.1, 5.2.2, 6.1, B.4, B.4, B.4

Nicolò Cesa-Bianchi, Pierre Gaillard, Gàbor Lugosi, and Gilles Stoltz. A new look at shifting regret. HAL, 2012. 2.2.1

Kamalika Chaudhuri and Staal A. Vinterbo. A stability-based validation procedure for differentially private machine learning. In *Advances in Neural Information Processing Systems*, 2013. 6.2.1

Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. arXiv, 2018a. 1, 3.2

Haokun Chen, Denis Krompass, Jindong Gu, and Volker Tresp. FedPop: Federated population-based hyperparameter tuning. arXiv, 2024. 1.3.1

Justin Y. Chen, Sandeep Silwal, Ali Vakilian, and Fred Zhang. Faster fundamental graph algorithms via learned predictions. In *Proceedings of the 40th International Conference on Machine Learning*, 2022. 4.1, 5, 5.2.2, 5.D, 5.D, 7.2

Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, 2018b. 10.1

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin P. Murphy, and Alan Loddon Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:834–848, 2018c. 8.2.2

Mayee F. Chen, Nicholas Roberts, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, and Christopher Ré. Skill-it! a data-driven skills framework for understanding and training language models. In *Advances in Neural Information Processing Systems*, 2023. 1.3.2

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *Proceedings of the 4th International Conference on Learning Representations*, 2016. 10.1, 10.3.1

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 8.A.3, 9.1, 9.B.1, 9.4, 9.5

Elliott Ward Cheney. *Introduction to Approximation Theory*. Chelsea Publishing Company, 1982. 7.B.1

Anda Cheng, Zhen Wang, Yaliang Li, and Jian Cheng. HPN: Personalized federated hyperparameter optimization. arXiv, 2023. 1.3.1

Anna Choromanska, Mikael Henaff, Michael Mathieu Gérard Ben Arous, and Yann LeCun. The loss surface of multilayer networks. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, 2015. 9.3.3

Nicolas Christianson, Bo Sun, Steven Low, and Adam Wierman. Risk-sensitive online algorithms. In *Proceedings of the 37th Annual Conference on Learning Theory*, 2024. 4.3

Jeremy Cohen, Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *Proceedings of the 9th International Conference on Learning Representations*, 2021. 8.3

Taco S. Cohen, Mario Geiger, Jonas Koehler, and Max Welling. Spherical CNNs. In *Proceedings of the 6th International Conference on Learning Representations*, 2018. 10.17

Vincent Cohen-Addad and Varun Kanade. Online optimization of smoothed piecewise constant functions. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017. 2.3.2

Vincent Cohen-Addad, Tommaso d?Orsi, Anupam Gupta, and Euiwoong Lee. Max-Cut with ε-accurate predictions. arXiv, 2024. 4.3

Thomas M. Cover. Universal portfolios. *Mathematical Finance*, 1:1–29, 1991. 6.D.1

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. AutoAugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Conference on Computer Vision and Pattern Recognition*, 2019. 9.1, 9.5

Madeleine Cule and Richard Samworth. Theoretical properties of the log-concave maximum likelihood estimator of a multidimensional density. *Electronic Journal of Statistics*, 4:254–270, 2010. 6.E.1

Zhongxiang Dai, Kian Hsiang Low, and Patrick Jaillet. Federated Bayesian optimization via Thompson sampling. In *Advances in Neural Information Processing Systems*, 2020. 3.2

Cong D. Dang and Guanghui Lan. Stochastic block mirror descent methods for nonsmooth and stochastic optimization. *SIAM Journal on Optimization*, 25:856–881, 2015. 9.2.1, 9.2.1

Varsha Dani, Thomas Hayes, and Sham Kakade. The price of bandit information for online optimization. In *Advances in Neural Information Processing Systems*, 2008. 2.4.3

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 10.1

Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *Proceedings of the 8th International Conference on Learning Representations*, 2020. 8.2.2, 10.1, 10.2.1, 10.2.2, 10.2.2, 8, 10.A.1, 10.A.2, 10.A.1

Herbert Aron David and Haikady Navada Nagaraja. *Order Statistics*. John Wiley & Sons, Inc., 2003. 6.4.1

Chandler Davis. Notions generalizing convexity for functions defined on spaces of matrices. In *Proceedings of Symposia in Pure Mathematics*, 1963. 2.A.3

Ofer Dekel, Arthur Flajolet, Nika Haghtalab, and Patrick Jaillet. Online learning with a hint. In *Advances in Neural Information Processing Systems*, 2017. 5.1

Angus Dempster, François Petitjean, and Geoffrey I. Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34:1454–1495, 2020. 10.17

Giulia Denevi, Carlo Ciliberto, Dimitris Stamos, and Massimiliano Pontil. Incremental learning-to-learn with statistical guarantees. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2018. 1.1

Giulia Denevi, Carlo Ciliberto, Riccardo Grazzi, and Massimiliano Pontil. Learning-to-learn stochastic gradient descent with biased regularization. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 1.1, 2.2.3, 2.2.3

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019. 1.3.2, 8.3

Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. arXiv, 2017. 9.4

Ilias Diakonikolas, Vasilis Kontonis, Christos Tzamos, Ali Vakilian, and Nikos Zarifis. Learning online algorithms with distributional advice. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. 4, 4.1, 5, 4, 5.1, 5.5.2, 5.5.2

Christos Dimitrakakis, Blaine Nelson, Zuhe Zhang, Aikaterini Mitrokotsa, and Benjamin I. P. Rubinstein. Differential privacy for bayesian inference through posterior sampling. *Journal of Machine Learning Research*, 18:1–39, 2017. 6, 6.2.1, 6.A.1

Xiaohan Ding, X. Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. RepVGG: Making VGG-style ConvNets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. 2

Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In *Advances in Neural Information Processing Systems*, 2021. 4, 4.1, 5, 5, 1, 5.1, 5.2, 5.2.1, 5.2.1, 5.2.2, 5.2.2, 5.3.1, 5.B, 5.C, 5.D, 7.2

Elizabeth D. Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002. 10.3.4

Wei Dong, Yuting Liang, and Ke Yi. Differentially private covariance revisited. In *Advances in Neural Information Processing Systems*, 2022. 4.2.3, 6, 6, 2, 6.2.3, 6.2.3, 6.3, 6.3.2, 6.3.2, 6.3.2, 1, 6.3.2, 6.B.1, 6.B.2

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 3.2, 3.4.1, 8.A.3, 9.1, 9.2.1, 9.2.2

Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020. 8.1, 8.2.1, 9.1, 3, 9.2.2, 9.2.2, 9.2, 9.B.1, 10.1, 10.2.2

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the 9th International Conference on Learning Representations*, 2021. 8.3, 10.3.5

Elbert Du, Franklyn Wang, and Michael Mitzenmacher. Putting the "learning" into learning-augmented algorithms for frequency estimation. In *Proceedings of the 38th International Conference on Machine Learning*, 2021a. 4

Simon S. Du, Wei Hu, Sham M. Kakade, Jason D. Lee, and Qi Lei. Few-shot learning via learning the representation, provably. In *Proceedings of the 9th International Conference on Learning Representations*, 2021b. 1.3.1, 2.3.1

Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in Neural Information Processing Systems*, 2017. 1

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari. Composite objective mirror descent. In *Proceedings of the 23rd Annual Conference on Learning Theory*, 2010. 6.2.3, 6.5.3, 6.5.3

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. 1.1, 2.1.1, 2.2.2, 2.2.4

Paul Dütting, Guru Guruganesh, Jon Schneider, and Joshua R. Wang. Optimal no-regret learning for one-sided lipschitz functions. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. 7.A

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014. 6, 6.1.1

Louis W. Ehrlich. An ad hoc SOR method. *Journal of Computational Physics*, 44:31–45, 1981. 7

Lakhdar Elbouyahyaoui, Mohammed Heyouni, Azita Tajaddini, and Farid Saberi-Movahed. On restarted and deflated block FOM and GMRES methods for sequences of shifted linear systems. *Numerical Algorithms*, 87:1257–1299, 2021. 7.2

Marek Eliáš, Haim Kaplan, Yishay Mansour, and Shay Moran. Learning-augmented algorithms with explicit predictors. arXiv, 2024. 4.3

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019a. 10.1

Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. arXiv, 2019b. 3.2

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi-task learning. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004. 1.A.2

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. On the convergence theory of gradient-

based model-agnostic meta-learning algorithms. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, 2020. 1.3.1, 2.3.1, 3.2

Xiequan Fan, Ion Grama, and Quansheng Liu. Hoeffding's inequality for supermartingales. *Stochastic Processes and their Applications*, 122(10):3545–3559, 2012. 2.A.8

Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 10, 10.2.3, 10.3.4

Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *Proceedings of the 6th International Conference on Learning Representations*, 2018. 1.1

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. 1, 1.3.2, 1.A.2, 2.1, 2.2.4, 3.B

Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergei Levine. Online meta-learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 1.1

Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, and Xavier Serra. FSD50K: An open dataset of human-labeled sound events. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:829–852, 2021. 10.17

Dylan J. Foster and Alexander Rakhlin. Beyond UCB: Optimal and efficient contextual bandits with regression oracles. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 7.1, 6, 7.4.2, 7.B.1, 7.B.2

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. 2

Isaac Fried and Jim Metzler. SOR vs. conjugate gradients in a finite element discretization. *International Journal for Numerical Methods in Engineering*, 12:1329–1332, 1978. 7.1.1

Andreas Frommer and Uew Glässner. Restarted GMRES for shifted linear systems. *SIAM Journal on Scientific Computing*, 19:15–26, 1998. 7.3.1

Anubhav Garg, Amit Kumar Saha, and Debo Dutta. Direct federated neural architecture search. arXiv, 2020. 3.2

Joseph Geumlek, Shuang Song, and Kamalika Chaudhuri. Rényi differential privacy mechanisms for posterior sampling. In *Advances in Neural Information Processing Systems*, 2017. 6, 6.2.1, 6.A.1

Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. arXiv, 2020. 3.2

Jennifer Gillenwater, Matthew Joseph, and Alex Kulesza. Differentially private quantiles. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. 6, 6.6.2

Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):

1115–1145, 1995. 2.A.5

Gene H. Golub and Qiang Ye. Inexact preconditioned conjugate gradient method with inner-outer iteration. *SIAM Journal on Scientific Computing*, 21:1305–1320, 1999. 7

Anne Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, 1997. 7

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent memory with optimal polynomial projections. In *Advances in Neural Information Processing Systems*, 2020. 10.16

Anupam Gupta, Aaron Roth, and John Ullman. Iterative constructions and private data release. In *Theory of Cryptography Conference*, 2012. 6.7

Anupam Gupta, Debmalya Panigrahi, Bernardo Subercaseaux, and Kevin Sun. Augmenting online algorithms with ε-accurate predictions. In *Advances in Neural Information Processing Systems*, 2022. 4.3

Rishi Gupta and Timothy Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017. 2.3, 2.3.4, 2.3.4, 2.A.5, 4, 4.1, 7, 7.2

Wolfgang Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer International Publishing, 2016. 7, 7.3.1, 7.3.2, 7.4, 7.C, 7.C.1, 7.C, 7.C, 7.D.1, 7.D.4, 7.E.3

Eric C. Hall and Rebecca M. Willet. Online optimization in dynamic environments. arXiv, 2016. 2.2.1

Moritz Hardt and Guy Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *51st Annual IEEE Symposium on Foundations of Computer Science*, 2010. 6.7

Mortiz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems*, 2012. 6, 6, 6.2.4, 6.3.3, 6.C

Keegan Harris, Ioannis Anagnostides, Gabriele Farina, Mikhail Khodak, Tuomas Sandholm, and Zhiwei Steven Wu. Meta-learning in games. In *Proceedings of the 11th International Conference on Learning Representations*, 2023. 1.3.1

Elad Hazan. Introduction to online convex optimization. In *Foundations and Trends in Optimization*, volume 2, pages 157–325. Now Publishers Inc., 2015. 1.A.1, 2.2.1, 2.4.1, 2.A.7, B.3

Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: Optimal algorithms for stochastic strongly-convex optimization. *Journal of Machine Learning Research*, 15:2489–2512, 2014. 1.A.1, 5.2.2

Elad Hazan and C. Seshadri. Efficient learning algorithms for changing environments. In *Proceedings of the 26th International Conference on Machine Learning*, 2009. 2.2.1

Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69:169–192, 2007. 2.2.1, 2.3.3, 2.3.3, 2.3.3, 2.4.1, 2.A.3,

2.A.4, 6.5.2, 6.D.1, 7.4.2, 7.B.3, B.3, B.3

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Towards non-i.i.d. and invisible data with FedNAS: Federated deep learning via neural architecture search. arXiv, 2020. 3.2

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 8.A.1, 8.A.2, 10.2.1, 10.2.3, 10.C.1

David P. Helmbold and Manfred K. Warmuth. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10:1705–1736, 2009. 5.E

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997. 2.2.4, 10.2.4

Shenda Hong, Yanbo Xu, Alind Khare, Satria Priambada, Kevin O. Maher, Alaa Aljiffry, Jimeng Sun, and Alexey Tumanov. HOLMES: Health onLine model ensemble serving for deep learning models in intensive care units. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. 10.17

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv, 2017. 9.5

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 9.1, 9.5

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 10.17

Andrew Hundt, Varun Jain, and Gregory D. Hager. sharpDARTS: Faster and more accurate differentiable architecture search. arXiv, 2019. 8.A.3

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*, 2011. 3.1, 9.3.1

Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrović, and Ronitt Rubinfeld. Online page migration with ML advice. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, 2022. 4, 4.1, 5, 2, 5.3, 5.3.1, 5.3.2

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015. 10.2.5

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, 34:1936–1962, 2020. 10.3.1, 10.3.3, 10.C.5, 10.12

Ali Jadbabaie, Alexander Rakhlin, and Shahin Shahrampour. Online optimization: Competing with dynamic comparators. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, 2015. 2.2.1, 2.4, 5.1, 5.4

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and Jo ao Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *Proceedings of the 10th International Conference on Learning Representations*, 2022. 10.1, 10.3.4

Prateek Jain, Pravesh Kothari, and Abhradeep Thakurta. Differentially private online learning. In *Proceedings of the 25th Annual Conference on Learning Theory*, 2012. 6.6.3

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, 2015. 2.4.2

Ghassen Jerfel, Erin Grant, Thomas L. Griffiths, and Katherine Heller. Reconciling meta-learning and continual learning with online mixtures of tasks. In *Advances in Neural Information Processing Systems*, 2019. 1.A.2

Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. arXiv, 2019. 3.2, 3.6

Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming*, 2020. 4

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-Keras: Efficient neural architecture search with network morphism. arXiv, 2018. 8.A

David Josephs, Carson Drake, Andrew M. Heroy, and John Santerre. sEMG gesture recognition with a simple model of attention. In *Machine Learning for Health*, 2020. 10.17

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. 10.2.4

Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. Practical and private (deep) learning without sampling or shuffling. In *Proceedings of the 38th International Conference on Machine Learning*, 2021a. 4.2.3, 3, 12, 6.5.1, 6.5.1, 4, 6.5.2, 6.6.3

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer

Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14:1–210, 2021b. 3.1

Sham Kakade and Shai Shalev-Shwartz. Mind the duality gap: Logarithmic regret algorithms for online optimization. In *Advances in Neural Information Processing Systems*, 2008. 1.2.3, B.3

Sham Kakade and Ambuj Tewari. On the generalization ability of online strongly convex programming algorithms. In *Advances in Neural Information Processing Systems*, 2008. 2.2.3, B.4.1

Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71:291–307, 2005. 2.4, 2.4.3

Haim Kaplan, Shachar Schnapp, and Uri Stemmer. Differentially private approximate quantiles. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. 6, 6, 6.2.2, 6.3.1, 6.3.1, 6.3.1, 6.3.1, 6.4.1, 6.6.2, 6.A.1, 6.A.1, 6.A.1, 6.A.1, 6.A.1, 6.A.1

Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-Lojasiewicz condition. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2016. 2.1.1

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 3.1, 3.3.1, 3.B

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3:422–440, 2021. 7.2

Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. Can mean-curvature flow be modified to be non-singular? In *Eurographics Symposium on Geometry Processing*, 2012. 7

Michael Kearns, Yishay Mansour, Andrew Y. Ng, and Dana Ron. An experimental and theoretical comparison of model selection methods. *Machine Learning*, 27:7–50, 1997. 9.3.3, 9.A.2

Vanshaj Khattar, Yuhao Ding, Bilgehan Sel, Javad Lavaei, and Ming Jin. A CMDP-within-online framework for meta-safe reinforcement learning. In *Proceedings of the 11th International Conference on Learning Representations*, 2023. 1.3.1

Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Adaptive gradient-based meta-learning methods. In *Advances in Neural Information Processing Systems*, 2019a. 0.3, 0

Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Provable guarantees for gradient-based meta-learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019b. 0.3, 0

Mikhail Khodak, Liam Li, Nicholas Roberts, Maria-Florina Balcan, and Ameet Talwalkar. A

simple setting for understanding neural architecture search with weight-sharing. AutoML Workshop, 2020. 0.3, 0

Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina Balcan, Virginia Smith, and Ameet Talwalkar. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. In *Advances in Neural Information Processing Systems*, 2021. 0.3, 0

Mikhail Khodak, Maria-Florina Balcan, Ameet Talwalkar, and Sergei Vassilvitskii. Learning predictions for algorithms with predictions. In *Advances in Neural Information Processing Systems*, 2022. 0.3, 0

Mikhail Khodak, Kareem Amin, Travis Dick, and Sergei Vassilvitskii. Learning-augmented private algorithms for multiple quantile release. In *Proceedings of the 40th International Conference on Machine Learning*, 2023a. 0.3, 0

Mikhail Khodak, Ilya Osadchiy, Keegan Harris, Maria-Florina Balcan, Kfir Y. Levy, Ron Meir, and Zhiwei Steven Wu. Meta-learning adversarial bandit algorithms. In *Advances in Neural Information Processing Systems*, 2023b. 0.3, 0

Mikhail Khodak, Edmond Chow, Maria-Florina Balcan, and Ameet Talwalkar. Learning to relax: Setting solver parameters across a sequence of linear system instances. In *Proceedings of the 12th International Conference on Learning Representations*, 2024. 0.3, 0

Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Choi, and Jiwon Kim. Auto-Meta: Automated gradient based meta learner search. arXiv, 2018. 1

John B. King, Samim Anghaie, and Henry M. Domanus. Comparative performance of the conjugate gradient and SOR methods for computational thermal hydraulics. In *Proceedings of the Joint Meeting of the American Nuclear Society and the Atomic Industrial Forum*, 1987. 7.1.1

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015. 2.1.1, 2.2.4, 2.2.4, 3.B, 3.B, 9.2.1, 9.A.1, 10.2.3

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017. 10.2.2, 10.A.6

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132:1–63, 1997. 3.4.1, 6.5.2

Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? In *Proceedings of the 35th International Conference on Machine Learning*, 2018. 9.A.1

Robert Kleinberg. Nearly tight bounds for the continuum-armed bandit problem. In *Advances in Neural Information Processing Systems*, 2004. 7.1, 7.3.3, 7.A

Ron Kohavi. Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996. 6.6.2, 6.A.1

Antti Koskela and Antti Honkela. Learning rate adaptation for federated and differentially private

learning. arXiv, 2018. 3.2

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, 2018. 4.1, 5.4, 5.C

Walid Krichene, Maximilian Balandat, Claire Tomlin, and Alexandre Bayen. The hedge algorithm on a continuum. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015. 5.5.1, 5.A.2

Akshay Krishnamurthy, John Langord, Alexandrs Slivkins, and Chicheng Zhang. Contextual bandits with continuous actions: Smoothing, zooming, and adapting. In *Proceedings of the 32nd Conference on Learning Theory*, 2019. 7.4.2

Alex Krizhevksy. Learning multiple layers of features from tiny images. Technical report, 2009. 8.2.1, 9.2.2, 10.C.1

Ravi Kumar, Manish Purohit, and Zoya Svitkina. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems*, 2018. 4, 5.5.1, 5.5.2

Kevin Kuo, Pratiksha Thaker, Mikhail Khodak, John Nguyen, Daniel Jiang, Ameet Talwalkar, and Virginia Smith. On noisy evaluation in federated hyperparameter tuning. In *Proceedings of Machine Learning and Systems*, 2023. 1.3.1

Branislav Kveton, Martin Mladenov, Chih-Wei Hsu, Manzil Zaheer, Csaba Szepesvári, and Craig Boutilier. Meta-learning bandit policies by gradient ascent. arXiv, 2020. 2.4

Branislav Kveton, Mikhail Konobeev, Manzil Zaheer, Chih-Wei Hsu, Martin Mladenov, Craig Boutilier, and Csaba Szepesvári. Meta-Thompson sampling. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. 2.4

John Lafferty, Han Liu, and Larry Wasserman. Statistical machine learning. 2010. 7.E.4, 9.A.2

Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Conference of the Cognitive Science Society*, 2017. 2.2.4, 2.3, 2.3.4

Rafał Latała, Ramon van Handel, and Pierre Youssef. The dimension-free structure of nonhomogeneous random matrices. *Inventiones Mathematicae*, 214:1031–1080, 2018. 6.5.3

Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, 2020. 4, 3, 5.4, 5.4.1, 1, 5.4, 5.C

Kevin Alexander Laube and Andreas Zell. Prune and replace NAS. In *Proceedings of the IEEE International Conference on Machine Learning and Applications*, 2019. 8.A.3

Sören Laue, Matthias Mitterreiter, and Joachim Giesen. Computing higher order derivatives of matrix and tensor expressions. In *Advances in Neural Information Processing Systems*, 2018. 7.E.1, 7.E.1

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*. 1999. 10.2.3

Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*.

SIAM, 2007. 7.5

Guilin Li, Xing Zhang, Zitong Wang, Zhenguo Li, and Tong Zhang. StacNAS: Towards stable and consistent differentiable neural architecture search. arXiv, 2019. 3.2, 3.4.1

Jeffrey Li, Mikhail Khodak, Sebastian Caldas, and Ameet Talwalkar. Differentially private meta-learning. In *Proceedings of the 8th International Conference on Learning Representations*, 2020a. 1.3.1, 3.2

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2019. 9, 9.1, 9.2, 9.2.2, 9.1, 9.3.1, 9.3.1, 9.4, 10

Liam Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018a. 3.1, 3.3.2, 3.3.2, 3.6, 8.1, 9.3.1, 9.3.2, 9.3.2, 9.B.2, 9.B.2, 10.1

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of the Conference on Machine Learning and Systems*, 2020b. 10.3.4

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the 9th International Conference on Learning Representations*, 2021a. 0.3, 0

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37, 2020c. 3.1

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proceedings of the Conference on Machine Learning and Systems*, 2020d. 3.1, 3.3.1, 3.B

Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *Proceedings of the 38th International Conference on Machine Learning*, 2021b. 3.2, 3.3.1, 3.B

Tian Li, Manzil Zaheer, Sashank Reddi, and Virginia Smith. Private adaptive optimization with side information. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. 6.2.1

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *Proceedings of the 6th International Conference on Learning Representations*, 2018b. 10.2.2, 10.A.6

Yichen Li, Peter Yichen Chen, Tao Du, and Wojciech Matusik. Learning preconditioners for conjugate gradient PDE solvers. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. 7.2, 7.5

Yingzhou Li, Haizhao Yang, Eileen R. Martin, Kenneth L. Ho, and Lexing Ying. Butterfly factorization. *Multiscale Modeling & Simulation*, 13(2):714–732, 2015. 10.2.4

Yingzhou Li, Haizhao Yang, and Lexing Ying. Multidimensional butterfly factorization. *Applied and Computational Harmonic Analysis*, 44(3):737–758, 2018c. 10.2.4

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-SGD: Learning to learning quickly for few-shot learning. arXiv, 2017. 1, 2.1, 2.2.4

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *Proceedings of the 9th International Conference on Learning Representations*, 2021c. 7.2, 8.2.2, 10.2.2, 10.2.4, 10.3.4, 10.3.4, 10.A.7, 10.7, 10.C.2, 10.11, 10.12, 10.17

Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta-learning. In *Proceedings of the 8th International Conference on Learning Representations*, 2020. 3.2

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. DARTS+: Improved differentiable architecture search with early stopping. arXiv, 2019. 8.A.3, 9.B.1

Elliott H. Lieb. Convex trace functions and the Wigner-Yanase-Dyson conjecture. *Advances in Mathematics*, 11:267–288, 1973. 2.A.7

Sen Lin, Mehmet Dedeoglu, and Junshan Zhang. Accelerating distributed online meta-learning via multi-agent collaboration under limited communication. In *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2021. 1.3.1

Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, 2022. 5.E, 5.E, 5.E.1, 5.E

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019a. 8.2.2, 10.2.4, 10.3.4

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, 2019b. 3.1, 3.2, 8.1, 8.2.1, 8.2.2, 8.2.2, 8.2.2, 8.A.3, 9, 9.1, 9.1, 9.2.1, 9.2.2, 9.2.2, 9.1, 9.3.3, 9.A.1, 9.B.1, 2, 9.4, 9.5, 10, 10.1, 1, 10.2.1, 2

Terrance Liu, Giuseppe Vietri, Thomas Steinke, Jonathan Ullman, and Zhiwei Steven Wu. Leveraging public data for practical private query release. In *Proceedings of the 38th International Conference on Machine Learning*, 2021a. 6, 6, 6.2.1, 6.2.2, 6.2.4, 6.3.3, 6.5

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical vision Transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021b. 8.3, 10.3.5

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022. 8.2.2, 10.3.4, 10.23, 10.C.6

Edward Loper and Steven Bird. NLTK: The natural language toolkit. arXiv, 2002. 6.E.3, 9.B.2

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, 2022. 8.3, 10.1

Tyler Lu, Dávid Pál, and Martin Pál. Contextual multi-armed bandits. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010. 7.A

Haipeng Luo. CSCI 699 Lecture 13. 2017. URL `https://haipeng-luo.net/courses/CSCI699/lecture13.pdf`. 2.A.7, 2.A.7, 2.A.8

Haipeng Luo, Chen-Yu Wei, Alekh Agarwal, and John Langford. Efficient contextual bandits in non-stationary worlds. In *Proceedings of the 31st Annual Conference on Learning Theory*, 2018. 2.4

Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 7.2

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4), 2021. 4, 4.A, 5.5, 2, 6.2.2

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *Proceedings of the European Conference on Computer Vision*, 2018. 9.5

Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Online optimization with uncertain information. *ACM Transactions on Algorithms*, 8:1–29, 2012. 5.5.2

Maryam Majzoubi, Chicheng Zhang, Rajan Chari, Akshay Krishnamurthy, John Langford, and Alexandrs Slivkins. Efficient contextual bandits with continuous actions. In *Advances in Neural Information Processing Systems*, 2020. 7.4.2

Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. arXiv, 2020. 3.2

Teodor Marinov and Julian Zimmert. The Pareto frontier of model selection for general contextual bandits. In *Advances in Neural Information Processing Systems*, 2021. 2.4

Donald Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963. 7

Tanya Marwah, Zachary C. Lipton, and Andrej Risteski. Parametric complexity bounds for approximating PDEs with neural networks. In *Advances in Neural Information Processing Systems*, 2021. 7.2

Andreas Maurer. Algorithmic stability and meta-learning. *Journal of Machine Learning Research*, 6:967–994, 2005. 2.1

Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. The benefit of multitask representation learning. *Journal of Machine Learning Research*, 17(1):2853–2884, 2016. 2.3.1

H. Brendan McMahan. A survey of algorithms and analysis for adaptive online learning. *Journal of Machine Learning Research*, 18, 2017. 6.5.1

H. Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex

optimization. In *Proceedings of the 23rd Annual Conference on Learning Theory*, 2010. 1.1

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artifical Intelligence and Statistics*, 2017. 1.A.2, 2.2.1, 2.3, 2.2.4, 2.B.1, 3.1, 3.1, 3.3, 3.3.1, 3.6, 3.B

Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, 2007. 6.2.2, 6.3.1, 6.A.1

Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. AtomNAS: Fine-grained end-to-end neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020. 8.1, 10.1

Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, Cambridge, UK, 2021. 2.1.1, 4, 4.1, 4.A, 5.1, 6, 7.1

Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. ACDC: A structured efficient linear layer. arXiv, 2015. 10.2.2

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012. 9.3.3

Aryan Mokhtari, Shahin Shahrampour, Ali Jadbabaie, and Alejandro Ribeiro. Online optimization in dynamic environments: Improved regret rates for strongly convex problems. In *Proceedings of the 55th IEEE Conference on Decision and Control*, 2016. 2.2.1, 5.4, B.5

Ahmadreza Moradipari, Mohammad Ghavamzadeh, Taha Rajabzadeh, Christos Thrampoulidis, and Mahnoosh Alizadeh. Multi-environment meta-learning in stochastic linear bandits. In *Proceedings of the 2022 IEEE International Symposium on Information Theory*, 2022. 2.4

Ken-ichiro Moridomi, Kohei Hatano, and Eiji Takimoto. Online linear optimization with the log-determinant regularizer. *IEICE Transactions on Information and Systems*, E101-D(6): 1511–1520, 2018. 2.A.6

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 3.2, 3.C.1

Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? In *Advances in Neural Information Processing Systems*, 2019. 9.1, 9.5

Cameron Musco and Christopher Musco. Randomized block Krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems*, 2015. 7.4.3

Vaishnavh Nagarajan and J. Zico Kolter. Generalization in deep networks: The role of distance from initialization. arXiv, 2017. 9.3.3

Vaishnavh Nagarajan and J. Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. In *Advances in Neural Information Processing Systems*, 2019. 9.3.3

Krishna Kanth Nakka, Ahmed Frikha, Ricardo Mendis, Xue Jiang, and Xuebing Zhou. Federated hyperparameter optimization through reward-based strategies: Challenges and insights. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2024. 1.3.1

Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 8.1, 10.1

Arkadi Nemirovski and David Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, 1983. 8.2.1, 9.2.1

Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM Studies in Applied and Numerical Mathematics, 1994. 2.4.3, 2.A.10, 2.A.10, 2.A.10

Gergely Neu. Explore no more: Improved high-probability regret bounds for non-stochastic bandits. In *Advances in Neural Information Processing Systems*, 2015. 2.4.2, 2.4.2, 2.A.7

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. arXiv, 2018. 1, 1.A.2, 2.1, 2.2, 2.2.1, 2.1, 2.2.4, 2.B.1, 3.3.1, 3.4.2, 3.B

Asaf Noy, Niv Nayman, Tal Ridnik, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik-Manor. ASAP: Architecture search, anneal and prune. arXiv, 2019. 8.A.3, 9.B.1

Taihei Oki and Shinsaku Sakaue. Faster discrete convex function minimization with predictions: The M-convex case. In *Advances in Neural Information Processing Systems*, 2023. 4.3

Francesco Orabona. A modern introduction to online learning. arXiv, 2022. 1.A.1

Francesco Orabona and David Pal. Coin betting and parameter-free online learning. In *Advances in Neural Information Processing Systems*, 2016. 2.3.3, 5.4, 5.4, 5.C.1

Francesco Orabona and Tatiana Tomassi. Training deep networks without learning rates through coin betting. In *Advances in Neural Information Processing Systems*, 2017. 6.6.2

Francesco Orabona, Nicolò Cesa-Bianchi, and Claudio Gentile. Beyond logarithmic bounds in online learning. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, 2012. 2.3.3

Samet Oymak, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural architecture search with train-validation split. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. 8.3

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013. 9.2.1

Michael L. Parks, Eric de Sturler, Greg Mackey, Duane D. Johnson, and Spandan Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5):1651–1674, 2006. 7.2

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and

Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 9.B.2

Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 8.2.2

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014. 6.E.3

Anastasia Pentina and Christoph H. Lampert. A PAC-Bayesian bound for lifelong learning. In *Proceedings of the 31st International Conference on Machine Learning*, 2014. 1.1

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. 3.1, 3.4.1, 8.1, 8.A, 8.A.3, 9, 9.1, 9.1, 9.2.1, 9.1, 9.3.1, 9.A.1, 9.4, 10, 1, 10.2.1, 10.2.3

Alo is Pourchot, Alexis Ducarouge, and Olivier Sigaud. To share or not to share: A comprehensive appraisal of weight-sharing. arXiv, 2020. 9.1

John W. Pratt. Concavity of the log likelihood. *Journal of the American Statistical Association*, 76(373):103–106, 1981. 6.2.2, 6.6.1, 6.E.1

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2008. 9.3.2

Alexander Rakhlin and Karthik Sridharan. Online learning with predictable sequences. In *Proceedings of the 26th Annual Conference on Learning Theory*, 2013. 5.1

Alexander Rakhlin and Karthik Sridharan. Efficient online multiclass prediction on graphs via surrogate losses. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017. 5.1

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. arXiv, 2017. 9.1, 9.5

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *Proceedings of the 5th International Conference on Learning Representations*, 2017. 1, 2.2.4

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2019. 9, 9.1, 9.4, 9.5

Esteban Real, Chen Liang, David R. So, and Quoc V. Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 8.1, 10.1

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. In *Proceedings of the 9th International Conference on Learning Representations*, 2021. 3.2

Nicholas Roberts, Mikhail Khodak, Tri Dao, Liam Li, Chris Ré, and Ameet Talwalkar. Re-

thinking neural operations for diverse tasks. In *Advances in Neural Information Processing Systems*, 2021. 0.3, 0

Nicholas Roberts, Samuel Guo, Cong Xu, Ameet Talwalkar, David Lander, Lvfang Tao, Linhang Cai, Shuaicheng Niu, Jianyu Heng, Hongyang Qin, Minwen Deng, Johannes Hog, Alexander Pfefferle, Sushil Ammanaghatta Shivakumar, Arjun Krishnakumar, Yubo Wang, Rhea Sukthanker, Frank Hutter, Euxhen Hasanaj, Tien-Dung Le, Mikhail Khodak, Yuriy Nevmyvaka, Kashif Rasul, Frederic Sala, Anderson Schneider, Junhong Shen, and Evan Sparks. AutoML Decathlon: Diverse tasks, modern methods, and efficiency at scale. In *Advances in Neural Information Processing Systems: Competition Track*, 2022. 8.3

Nicholas Roberts, Yingyu Liang, and Fred Sala. Understanding neural architecture search by its architecture parameters. 2023. 8.3

Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, 2020. 4

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, 2015. 6

Timothy Roughgarden. *Beyond Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020. 4.1

Reuven Y. Rubinstein and Alexander Shapiro. *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*. John Wiley & Sons, Inc., 1993. 3.4.1

Alessandro Rudi and Lorenzo Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems*, 2017. 9.A.2

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 8.2.1, 9.2.2, 10.C.1

Ankan Saha, Prateek Jain, and Ambuj Tewari. The interplay between stability and regret in online learning. arXiv, 2012. 2.A.1, 2.A.1

Shinsaku Sakaue and Taihei Oki. Discrete-convex-analysis-based framework for warm-starting algorithms with predictions. In *Advances in Neural Information Processing Systems*, 2022. 4.3, 7.2

Shinsaku Sakaue and Taihei Oki. Rethinking warm-starts with predictions: Learning predictions close to sets of optimal solutions for faster L-/$L^\natural$-convex function minimization. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. 4.3

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In *Proceedings of the 5th Annual Conference on Learning for Dynamics and Control*, 2023. 7.2

Nikunj Saunshi, Yi Zhang, Mikhail Khodak, and Sanjeev Arora. A sample complexity separation between non-convex and convex meta-learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 1.3.1, 2.1.2, 2.3.1

Nikunj Saunshi, Arushi Gupta, and Wei Hu. A representation learning perspective on the importance of train-validation splitting in meta-learning. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. 1.3.1

Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. Uniform bounds for scheduling with job size estimates. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference*, 2022. 4

Jeremy Seeman, Aleksandra Slavkovic, and Matthew Reimherr. Private posterior inference consistent with public information: A case study in small area estimation from synthetic census data. In *Proceedings of the International Conference on Privacy in Statistical Databases*, 2020. 6, 6.2.1, 6.A.1

Jonas Seng, Pooja Prasad, Martin Mundt, Devendra Singh Dhami, and Kristian Kersting. FEATHERS: Federated architecture and hyperparameter search. arXiv, 2023. 1.3.1

Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020. 10.2.4

Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011. 1.A.1, 1.A.1, 2.1.1, 2.2.1, 2.2.1, 2.2.2, 2.3.3, 2.3.3, 2.3.3, 2.4.1, 2.A.1, 2.A.5, 2.A.6, 3.A, 3.E, 5.2.2, 5.3.2, 5.3.2, 5.4, 5.A.3, 5.B, 5.C.1, 5.C.2, 5.C.2, 5.E, 6.5.1, 6.E.2, 9.2.1, B.3

Amr Sharaf and Hal Daumé III. Meta-learning effective exploration strategies for contextual bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. 2.4

Junhong Shen, Mikhail Khodak, and Ameet Talwalkar. Efficient architecture search for diverse tasks. In *Advances in Neural Information Processing Systems*, 2022. 0.3, 0

Junhong Shen, Liam Li, Lucio M. Dery, Corey Staten, Mikhail Khodak, Graham Neubig, and Ameet Talwalkar. Cross-modal fine-tuning: Align then refine. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. 8.3

Junhong Shen, Neil Tenenholtz, James Brian Hall, David Alvarez-Melis, and Nicolò Fusi. Tag-LLM: Repurposing general-purpose LLMs for specialized domains. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. 8.3

David Simchi-Levi and Yunzong Xu. Bypassing the monster: A faster and simpler optimal algorithm for contextual bandits under realizability. *Mathematics of Operations Research*, 47, 2021. 7.1, 7.F.1

Max Simchowitz, Christopher Tosh, Akshay Krishnamurthy, Daniel J. Hsu, Thodoris Lykouris, Miro Dudik, and Robert E. Schapire. Bayesian decision-making under misspecified priors with applications to meta-learning. In *Advances in Neural Information Processing Systems*, 2021. 2.4

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representa-*

tions, 2015. 8.A.2

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018. 10.2.4

Adam Smith and Abhradeep Thakurta. (Nearly) optimal algorithms for private online learning in full-information and bandit settings. In *Advances in Neural Information Processing Systems*, 2013. 6.5.1, 6.5.2, 6.6.3, 6.D.1

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, 2017. 2.2.4, 3.2

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, 2017. 1, 1.1

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012. 3.3.2

Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004. 2.3, 4.1

Nathan Srebro, Karthik Sridharan, and Ambuj Tewari. On the universality of online mirror descent. In *Advances in Neural Information Processing Systems*, 2011. 1.A.1

Karthik Sridharan, Nathan Srebro, and Shai Shalev-Schwartz. Fast rates for regularized objectives. In *Advances in Neural Information Processing Systems*, 2008. 9.A.2

Vaidehi Srinivas and Avrim Blum. Competitive strategies to use "warm start" algorithms with predictions. arXiv, 2024. 4.3

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, J. Liu, and Diana Marculescu. Single-path NAS: Designing hardware-efficient ConvNets in less than 4 hours. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2020. 3

Bo Sun, Jerry Huang, Nicolas Christianson, Mohammad Hajiesmaili, Adam Wierman, and Raouf Boutaba. Online algorithms with uncertainty-quantified predictions. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. 4.3

Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021. 7.2

Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. *Journal of Machine Learning Research*, 4:773–818, 2003. 2.4, 2.4.3

Jurjen D. Tebbens and Miroslav Tůma. Efficient preconditioning of sequences of nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 29:1918–1941, 2007. 7.2

James William Thomas. *Numerical Partial Differential Equations*. Springer Science+Business Media, 1999. 7

Sebastian Thrun and Lorien Pratt. *Learning to Learn*. Springer Science & Business Media, 1998. 1, 2.1

Lloyd N. Trefethen. Is Gauss quadrature better than Clenshaw-Curtis? *SIAM Review*, 50(1):

67–87, 2008. 7.B.1, 7.B.1

Lloyd N. Trefethen and Mark Embree. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*. Princeton University Press, 2005. 7.2, 7.3.2

Nilesh Tripuraneni, Chi Jin, and Michael I. Jordan. Provable meta-learning of linear representations. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. 1.3.1, 2.3.1

Constantino Tsallis. Possible generalization of Boltzmann-Gibbs statistics. *Journal of Statistical Physics*, 52:479–487, 1988. 2.4

Renbo Tu, Mikhail Khodak, Nicholas Roberts, and Ameet Talwalkar. NAS-Bench-360: Benchmarking diverse tasks for neural architecture search. In *Advances in Neural Information Processing Systems: Track on Datasets and Benchmarks*, 2022. 8.2.2, 8.3, 10.1, 1, 10.2.4, 10.5, 10.3.4, 10.5, 10.3.4, 10.3.4, 10.17

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. In *Proceedings of the 9th ISCA Workshop on Speech Synthesis Workshop*, 2016. 8.2.2

L. Dale Van Vleck and D. J. Dwyer. Successive overrelaxation, block iteration, and method of conjugate gradients for solving equations for multiple trait evaluation of sires. *Jorunal of Dairy Science*, 68:760–767, 1985. 7.1.1

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. 8.A.2, 10.2.4, 10.2.5, 10.3.5

Mengting Wan and Julian J. McAuley. Item recommendation on monotonic behavior chains. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018. 6.6.2

Rui-Rui Wang, Qiang Niu, Xiao-Bin Tang, and Xiang Wang. Solving shifted linear systems with restarted GMRES augmented with error approximations. *Computers & Mathematics with Applications*, 78:1910–1918, 2019. 7.3.1

Sida Wang and Christopher D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 2012. 9.3.2, 9.B.2

Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. MergeNAS: Merge operations into one for differentiable architecture search. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, 2020a. 2, 10.3.2

Yujing Wang, Yaming Yang, Yiren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Yunhai Tong, Mao Yang, and Lidong Zhou. Textnas: A neural architecture search space tailored for text representation. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2020b. 8.1, 10.1

Zhen Wang, Weirui Kuang, Ce Zhang, Bolin Ding, and Yaliang Li. FedHPO-Bench: A benchmark suite for federated hyperparameter optimization. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. 1.3.1

Zhiwei Wang, Yao Ma, Zitao Liu, and Jiliang Tang. R-Transformer: Recurrent neural network enhanced transformer. In *Proceedings of the 8th International Conference on Learning Representations*, 2020c. 10.3, 10.2.4, 10.16

Chen-Yu Wei and Haipeng Luo. Non-stationary reinforcement learning without prior knowledge: An optimal black-box approach. In *Proceedings of the 34th Annual Conference on Learning Theory*, 2021. 2.4

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *Proceedings of the 33th International Conference on Machine Learning*, 2016. 10.1, 10.3.1

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th International Conference on Machine Learning*, 2009. 1.A.2

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. In *Advances in Neural Information Processing Systems*, 2020. 8.1

Colin White, Sam Nolen, and Yash Savani. Local search is state of the art for NAS benchmarks. In *Proceedings of the 7th ICML Workshop on Automated Machine Learning*, 2021. 8.1

Colin White, Mikhail Khodak, Renbo Tu, Shital Shah, and Sébastien Bubeck. A deeper look at zero-cost proxies for lightweight NAS. In *Proceedings of the 10th International Conference on Learning Representations: Block Track*, 2022. 8.3

Zbigniew I. Woźnicki. On numerical analysis of conjugate gradient method. *Japan Journal of Industrial and Applied Mathematics*, 10:487–519, 1993. 7.1.1

Zbigniew I. Woźnicki. On performance of SOR method for solving nonsymmetric linear systems. *Journal of Computational and Applied Mathematics*, 137:145–176, 2001. 7.1.1

Tianyi Wu, Sheng Tang, Rui Zhang, Juan Cao, and Jintao Li. Tree-structured Kronecker convolutional network for semantic segmentation. *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 940–945, 2019. 10.3.2

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, 2019. 8.A.3, 9.2, 9.1, 9.B.1, 9.4, 9.5, 10.3.3

Mengwei Xu, Yuxin Zhao, Kaigui Bian, Gang Huang, Qiaozhu Mei, and Xuanzhe Liu. Federated neural architecture search. arXiv, 2020a. 3.2

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020b. 9.2, 9.2.1, 9.2.2, 9.2.2, 9.1, 9.B.1, 9.B.1, 9.4, 9.5, 10

Yoshihiro Yamada, Masakazu Iawmura, and Koichi Kise. ShakeDrop regularization. arXiv, 2018. 9.4

Takuya Yamano. Some properties of q-logarithm and q-exponential functions in Tsallis statistics. *Physica A: Statistical Mechanics and its Applications*, 305:486–496, 2002. 2.A.7

Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly

hard. In *Proceedings of the 8th International Conference on Learning Representations*, 2020. 10.2.1, 10.C.1, 10.9

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. 8.2.2

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. 9.2.2, 10, 10.2.2, 10.2.3

David M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, 1971. 7.1, 7.3.1, 7.3.2, 7.3.2, 7.3.4

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *Proceedings of the 4th International Conference on Learning Representations*, 2016. 8.2.2

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020a. 1.3.1, 9.1

Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation. arXiv, 2020b. 3.2

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016. 10.3.1, 10.C.1, 10.C.5, 10.11

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020a. 9.1, 9.2

Arber Zela, Julien Siems, and Frank Hutter. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020b. 8.1, 9.2.2, 9.2.2, 9.2, 9.B.1, 10, 10.1

Keming Zhang and Joshua S. Bloom. deepCR: Cosmic ray rejection with deep learning. *The Astrophysical Journal*, 889(1), 2020. 10.17

Lijun Zhang, Tianbao Yang, Jinfeng Yi, and Rong Jin Zhi-Hua Zhou. Improved dynamic regret for non-degenerate functions. In *Advances in Neural Information Processing Systems*, 2017. 2.2.1

Siqi Zhang and Niao He. On the convergence rate of stochastic mirror descent for nonsmooth nonconvex optimization. arXiv, 2018. 9.2.1, 9.2.1, 9.2.1, 9.A.1, 9.A.2, 9.A.3, 9.A.1

Tong Zhang. Data dependent concentration bounds for sequential prediction algorithms. In *Proceedings of the International Conference on Learning Theory*, 2005. 2.2.3, B.4.2, B.4.2, B.4.2

Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. arXiv, 2020. 9.1

Zijun Zhang, Christopher Y. Park, Chandra L. Theesfeld, and Olga G. Troyanskaya. An auto-

mated framework for efficiently designing deep convolutional neural networks in genomics. *Nature Machine Intelligence*, 3(5):392–400, 2021. 1, 10.3.4, 10.3.4

Peng Zhao, Guanghui Wang, Lijun Zhang, and Zhi-Hua Zhou. Bandit convex optimization in non-stationary environments. *Journal of Machine Learning Research*, 22(125):1–45, 2021. 2.4

Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature Methods*, 12:931–934, 2015. 10.17

Yi Zhou, Parikshit Ram, Theodoros Salonidis, Nathalie Baracaldo, Horst Samulowitz, and Heiko Ludwig. Single-shot general hyper-parameter optimization for federated learning. In *Proceedings of the 11th International Conference on Learning Representations*, 2023. 1.3.1

Yufan Zhou, Zhenyi Wang, Jiayi Xian, Changyou Chen, and Jinhui Xu. Meta-learning with neural tangent kernels. In *Proceedings of the 9th International Conference on Learning Representations*, 2021. 2.3.1

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable Transformers for end-to-end object detection. In *Proceedings of the 9th International Conference on Learning Representations*, 2021. 10.3.5

Yinglun Zhu and Paul Mineiro. Contextual bandits with smooth regret: Efficient learning in continuous action spaces. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. 7.4.2

Julian Zimmert and Yevgeny Seldin. Tsallis-INF: An optimal algorithm for stochastic and adversarial bandits. *Journal of Machine Learning Research*, 22:1–49, 2021. 17, 7.3.3, 19, 7.A.2, 7.F.1

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, 2003. 1.A.1, 2.2.1, 5.2.2, 5.4, 5.B, 6.5.1

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 8.1, 9.1, 9.4, 9.5, 1