Optimal and Adaptive Online Learning

Haipeng Luo

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Robert E. Schapire

May 2016

Abstract

Online learning is one of the most important and well-established machine learning models. Generally speaking, the goal of online learning is to make a sequence of accurate predictions "on the fly," given some information of the correct answers to previous prediction tasks. Online learning has been extensively studied in recent years, and has also become of great interest to practitioners due to its effectiveness in dealing with non-stationary data as well as its applicability to large-scale applications.

While many useful ideas are well-established in the offline learning setting where a set of data is available beforehand, their counterparts in the online setting are not always straightforward and require more understanding. Moreover, existing online learning algorithms are not always directly applicable in practice. One important reason is that they usually rely on sophisticated tuning of parameters, a delicate approach that can yield sound theoretical guarantees, but that does not work well in practice. Another reason is that existing algorithms are usually guaranteed to work well in one particular situation or another, but not all. A single algorithm that can ensure worst-case robustness while still enjoying the ability to exploit easier data at the same time is relatively rare and certainly desirable in practice.

Motivated by all the above issues, this thesis focuses on designing more practical, adaptive and ready-to-use online learning algorithms, including

- novel online algorithms which combine expert advice in an optimal and parameter-free way and work simultaneously under different patterns of data as well as different evaluation criteria;

- a novel and rigorous theory of online boosting which studies improving the accuracy of any existing online learning algorithm by training and combining several copies of it in a carefully designed manner;

- a family of highly efficient online learning algorithms which make use of second order information of the data and enjoy good performance even when dealing with ill-conditioned data.

In summary, this thesis develops and analyzes several novel, optimal and adaptive online learning algorithms which greatly improve upon previous work and have great practical potential.

Acknowledgements

As I always tell myself, the most fortunate thing for me during this PhD journey is undoubtedly to have Rob Schapire as my advisor. I can not express in words how thankful I am to Rob. He took me as his student even if I did not have much background in machine learning. During these five years, he has always been patient, helpful and supportive to me, and step by step guided me to become an independent researcher. His clarity in communication, rigorousness in conducing research and wisdom in solving hard problems have been incredibly influential to me. But his impact on me is far beyond the academic area. I learned so much from his kindness to others, his good humor, his positive attitude to life, his "family first" motto and so many other virtues. Rob has always been and will always remain a role model whom I look up to.

I would also like to express my sincere gratitude to Elad Hazan, who generously supported me after Rob left Princeton. I am fortunate enough to also have a chance to work with and learn from him, and my only regret is that I would not have much time to stay at Princeton working with him.

I also have the great fortunate to collaborate with many other outstanding scholars, including Alekh Agarwal, Alina Beygelzimer, Nicolò Cesa-Bianchi, Yoav Freund, Peter Grünwald, Patrick Haffner, Satyen Kale, Wouter Koolen, John Langford, Gergely Neu, Jean-François Paiement, Vasilis Syrgkanis and Tim van Erven. In particular, I would like to thank Alina and Satyen for being my great internship mentors at Yahoo Labs, and Alekh and John for also being my great mentors at Microsoft Research. A significant part of this thesis was conducted during these two great internships. Also a special thanks to Peter Grünwald, who generously invited me to CWI for a one-month productive visit.

I am also grateful to Sanjeev Arora, Barbara Engelhardt, Elad and Satyen for being my committee members and providing helpful suggestions for this thesis.

My research would not have been possible without the support of NSF Grant #1016029.

I would also like to thank Ellen Anagbo, Susan Greene, Barbara Jensen, Melissa Lawson, Paul Nutkowitz and Tom Taylor for helping me improve my English during my first year at Princeton. Their help has been extremely important for both my career and daily life as a foreigner in the US.

Finally, I am very grateful for having the most supportive parents who know little about what I am doing but always let me freely pursue my dreams even though that means I need to be far away from them. Last but certainly not least, I need to thank my beloved wife Xiaohui for joining me in this challenging yet wonderful foreign journey. I owe her so many weekends and I hope I could make up for her in the near future.

To my parents and my beloved wife Xiaohui.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Online learning studies how to make a sequence of accurate predictions "on the fly" when interacting with the environment. Many real-life applications can be modeled as an online learning problem, such as email spam detection, internet routing, recommendation systems, to name a few. The study of online learning dates back to the 1950s. A general and modern framework initiated by Zinkevich [112] treats online learning as a game iteratively played by a *learner* and an *adversary*. On each round $t = 1, \ldots, T$ of the game:

1) the learner makes a decision by picking a point $\boldsymbol{w}_t$ from a decision space $\mathcal{K}$;

2) the adversary, knowing the learner's decision $\boldsymbol{w}_t$, decides a loss function f_t defined on $\mathcal{K}$.

3) the learner suffers loss $f_t(\boldsymbol{w}_t)$ for this round, and receives feedback about f_t.

For example, in the application of web advertisement placing, we can treat each visit of the webpage as a single round of the online learning game. For each of them, the learner's job is to pick one of the N ads to display, with the goal of maximizing clickthrough rate. Here, the decision space $\mathcal{K}$ can be defined as the set of all distributions over these N ads, and thus for each visitor t, the learner needs to choose a distribution $\boldsymbol{w}_t \in \mathcal{K}$ from which an ad is drawn to display. For the loss

function, suppose displaying ad $i \in \{1, \ldots, N\}$ for visitor t will incur loss $\ell_{t,i}$ (e.g. $\ell_{t,i} = 1$ if the visitor would not click ad i and $\ell_{t,i} = 0$ otherwise). Then we can define $f_t(\boldsymbol{w})$ as $\boldsymbol{w}^\top \boldsymbol{\ell}_t$ where $\boldsymbol{\ell}_t$ is an N dimensional vector with the i-th coordinate being $\ell_{t,i}$. In other words, the loss of the learner $f_t(\boldsymbol{w}_t)$ would be the expected loss incurred by displaying ads randomly according to $\boldsymbol{w}_t$.

The typical goal of online learning is to minimize *regret* which, roughly speaking, measures how much less loss the learner would have suffered in retrospect if another strategy had been played throughout. For example, in the above formal setting, the regret to a fixed decision $\boldsymbol{w} \in \mathcal{K}$ is defined as

$$R_T(\boldsymbol{w}) \stackrel{\text{def}}{=} \sum_{t=1}^{T} f_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} f_t(\boldsymbol{w}),$$

which is simply the difference between the learner's total loss and the loss of playing a fixed point $\boldsymbol{w}$ all the time (assuming the adversary would not change the outcomes). Usually, we are interested in minimizing regret to the best fixed point in hindsight, that is, a point $\boldsymbol{w}$ that minimizes $\sum_{t=1}^{T} f_t(\boldsymbol{w})$. In the advertisement placing example above, this tells us how much less loss the learner would have suffered if he always display an ad that leads to the most clicks in retrospect.

There are several possible forms of feedback that the learner could receive on each round. For example, in the simplest case, the learner observes the entire loss function f_t chosen by the adversary. A more difficult situation is to only observe some partial information of f_t, such as its gradient or even just its value at the choice of the learner (which is the case for the advertisement application). All these feedback models have been considered in the literature and they capture different kinds of applications. In this thesis, we will mostly focus on the gradient feedback model.

Using these feedback, the learner applies some learning algorithm to discover and exploit the pattern of the data in order to make smart choices on each round and

obtain good overall prediction performance. Designing these algorithms and analyzing their theoretical guarantees are the main research topic of online learning. We refer the reader to [22, 101, 53] for excellent surveys of online learning.

1.1 Motivation

Although online learning has been an active research area in recent years, many challenges still remain. For example, existing online learning algorithms are not always directly applicable in practice. One important reason is that they usually rely on sophisticated tuning of parameters, a delicate approach that can yield sound theoretical guarantees, but that does not work well in practice. People usually end up spending a significant amount of time and work on tuning these parameters for one specific task, and then do it again when facing a different task or even the same task but with new data. Another reason is that existing algorithms are usually guaranteed to work well in one particular situation or another, but not all. A single algorithm that can ensure worst-case robustness while still enjoying the ability to exploit easier data at the same time is relatively rare and is certainly of great value in practice.

Moreover, while many useful ideas are well-established in the offline learning setting where a set of data is available beforehand, their counterparts in the online setting are not always straightforward and require more understanding. For example, boosting [96, 37], the idea of combining weak "rules of thumb" into a strong predictor, has been thoroughly studied in the offline setting and has proven extremely useful in practice since it was proposed more than two decades ago. However, it is not clear what boosting actually means in the online setting and whether it can also be as useful in terms of improving accuracy of existing online learning algorithms.

Another example is data preprocessing which aims to provide better represented data as the input of learning algorithms in order to achieve better performance. While

this is a simple and low-cost approach in the offline setting, it becomes much less clear what the right counterpart is and how to do it efficiently in the online setting where we do not have all the data in advance.

1.2 Contributions

Motivated by all the above issues, this thesis focuses on designing more practical, adaptive and ready-to-use online learning algorithms. Our main contributions are summarized below.

Combining Expert Advice More Effectively. In Chapter 2, we study how to more effectively combine expert advice when making online decisions. This is a classic online learning problem with numerous real-life applications. For example, which stock should we invest in given the advice of several different consultants? Or what should we predict for tomorrow's temperature given the predictions of several different forecast models? Moreover, this problem is also the foundation and meta-algorithm of many other learning algorithms (such as boosting, online principal component analysis (PCA), matrix completion, etc).

Although the problem is well studied, there are still many challenging and unachieved goals. For example, can we learn effectively when there are a huge or even infinite number of experts? Can we achieve much better performance when dealing with easy problems, while still ensuring worst-case robustness in difficult situations? These objectives were only separately achieved before by different algorithms, which means in practice we still need to try out these different algorithms and see which one works better for different problems.

To solve these problems, Chapter 2 first discusses a general framework to design algorithms for this problem through a game-theoretic analysis. Using this framework we then develop one single parameter-free algorithm which can achieve all the above-

mentioned goals simultaneously so that one can safely use our algorithm in different applications without tuning any parameters. This is the first such algorithm and it has many other implicit applications. For example, suppose we need to predict what the subject is in an image given an overly complex decision tree, and our goal is to predict almost as well as the best pruning tree within this complex tree. We show that using our algorithm leads to exponential improvement compared to previous work.

Online boosting. In Chapter 3, we study online boosting, a theory of boosting the accuracy of any existing online learning algorithm. As mentioned above, traditional batch boosting is a central methodology in machine learning which studies the task of combining weak rules of thumb into a strong learning algorithm. In practice, one can pick any off-the-shelf algorithm, such as decision trees or neural networks, and apply boosting to get potentially improved accuracy. While boosting is well-studied in the offline setting, there is very little work on the theory of online boosting. The main question is whether we can improve the accuracy of existing online prediction algorithms in a similar way as we do in the offline case. For example, given an online weak spam filter which on average only has about 51% accuracy, can we somehow build a very accurate spam filter (say, 99% accuracy) by combining several copies of these weak spam filters in an online fashion?

Starting with this question, we propose a novel online boosting framework with a mild and natural assumption that is a direct analogue of the one in the offline case. Roughly speaking, the assumption requires that the base learning algorithm can predict slightly better than random guessing in the long run. Based on this assumption, we develop an online boosting algorithm which can combine the predictions of several of these base algorithms to achieve arbitrarily small accuracy in the long run. Our algorithm is optimal in the sense that no other algorithms can achieve the same

result with less data or less copies of the base algorithms. This answers the question of whether online boosting is possible in the affirmative.

However, this optimal algorithm again relies on a parameter that is often unknown in practice. To address this problem, we borrow the idea of the well-known AdaBoost algorithm [38] for the offline setting, and develop its online variant which is parameter-free and able to adapt how well each base learning algorithm is doing.

Furthermore we extend the theory and algorithms to more general regression problems so that it is more applicable in practice. In this setup we even relax the weak learning assumption, only assuming that we have access to an oracle which can find the best among a base hypothesis class in an online fashion.

Our algorithms are implemented in the latest version of Vowpal Wabbit (VW), an open source machine learning system [105], and are available to the public. Therefore, whenever a new algorithm is implemented in VW, one can easily try running our boosting algorithms on top of it (with just an extra command line option) to see whether the accuracy can be improved, which we believe is very beneficial in practice. We have also conducted initial experiments on benchmark data and observed that our algorithms indeed improve over previous work and significantly improve the accuracy of the existing base learning algorithms implemented in VW.

Efficient Second Order Online Learning. In Chapter 4 we study efficient invariant online learning. The performance of classic learning algorithms (such as gradient descent) usually depends on how the data is represented, for instance, whether length is measured in meters or centimeters. In the offline setting where all the data is available in advance, one can simply perform preprocessing (such as whitening) on the data before feeding it to the algorithm, so that the way the data is represented does not affect the algorithm anymore. Since this is only done once before training happens, the overhead is negligible. In the online setting, however, examples arrive one

by one on the fly and data preprocessing is no longer an option. The main question is thus whether in this case we can still have an efficient algorithm whose performance is independent of how the data is represented.

We propose a variant of online gradient descent that makes use of second order information of the data and is invariant to linear transformation, meaning that had the data been represented in a different way (by applying a linear transformation), the algorithm would still make the exact same predictions. As a result, the algorithm enjoys a convergence guarantee that is independent of how the data is represented. In other words, with our algorithm, in some sense one does not need to be concerned about how to represent the data anymore.

Moreover, to overcome the quadratic running time of the algorithm, we combine different "sketching" techniques with our algorithm. The idea is to sketch the data by keeping a very small amount of key information and then use this information to guide the direction of gradient descent. The running time of our algorithm can then be reduced to be linear in the number of dimensions or even the sparsity of the data. We rigorously show that the performance of our algorithm is only slightly affected as long as the size of the sketch is larger than the number of significant directions of the data.

Our algorithm is again implemented in VW and publicly available. Empirical study shows that our algorithm indeed outperforms previous work on a large sets of benchmark data.

1.3 Notation

We introduce some common notation which is used throughout this thesis. Vectors are represented by bold letters (e.g., $\boldsymbol{x}$, $\boldsymbol{w}$, ...) and matrices by capital letters (e.g.,

M, A, ...). $M_{i,j}$ denotes the (i,j) entry of matrix M. $\boldsymbol{I}_d$ represents the $d \times d$ identity matrix, $\mathbf{0}_{m \times d}$ represents the $m \times d$ matrix of zeroes.

We use $[N]$ to denote the set $\{1, \ldots, N\}$, Δ_N to denote the simplex of all distributions over $[N]$, and $\mathrm{RE}(\boldsymbol{p} \mid\mid \boldsymbol{q}) \stackrel{\text{def}}{=} \sum_{i=1}^{N} p_i \ln\left(\frac{p_i}{q_i}\right)$ to denote the *relative entropy* (also known as the Kullback-Leibler divergence) between two distributions $\boldsymbol{p} \in \Delta_N$ and $\boldsymbol{q} \in \Delta_N$. The sign function $\mathrm{sign}(a)$ is 1 if $a \geq 0$ and -1 otherwise.

We use the $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ notation to suppress dependence on polylogarithmic factors in the natural parameters. Also we consider "log log" terms to be nearly constant, and use $\hat{O}()$ notation to hide these terms. (Indeed, $\ln \ln x$ is smaller than 4 even when x is as large as the age of the universe expressed in microseconds.)

Chapter 2

Combining Expert Advice Effectively

In this chapter we study the classic online learning problem of predicting with expert advice, where the goal is to combine the expert advice in an efficient and effective way to make online decisions. For example, which stock should we invest in given the advice of several different consultants? Or what should we predict for tomorrow's temperature given the predictions of several different forecast models?

Although the problem is well studied, there are still many challenging and unachieved goals. To solve these problems, we first provide a general mechanism to design expert algorithms based on a minimax analysis within a game-theoretic framework called *drifting games*. The original minimax analysis for drifting games is generalized by applying a series of relaxations, starting from choosing a convex surrogate of the 0-1 loss function. With different choices of surrogates, we not only recover existing algorithms, but also propose a new algorithm, NormalHedge.DT, that is totally parameter-free and enjoys some useful properties. Our drifting-games framework naturally allows us to study a generalized notion of regret that measures how good the algorithm is compared to all but the top small fraction of candidates.

As an application, we translate NormalHedge.DT into a new adaptive boosting algorithm that is computationally faster than previous work as shown in experiments, since it ignores a large number of examples on each round.

Next we propose a further improved variant of NormalHedge.DT, called AdaNormalHedge, which is a highly adaptive algorithm that achieves several objectives simultaneously without using any prior information. On one hand, AdaNormalHedge ensures small regret when the competitor has small loss and almost constant regret when the losses are stochastic. On the other hand, AdaNormalHedge is able to compete with any convex combination of the experts simultaneously, with a regret in terms of the relative entropy of the prior and the competitor. This resolves an open problem proposed by Chaudhuri et al. [26] and Chernov and Vovk [29].

Moreover, we extend the results to the sleeping expert setting and provide two applications to illustrate the power of AdaNormalHedge: 1) competing with time-varying unknown competitors and 2) predicting almost as well as the best pruning tree. Our results on these applications significantly improve previous work from different aspects, and a special case of the first application resolves another open problem proposed by Warmuth and Koolen [106] on whether one can simultaneously achieve optimal shifting regret for both adversarial and stochastic losses.

This chapter includes joint work with Robert Schapire and has appeared in [73, 74].

2.1 A General Framework to Design Algorithms

To solve an online learning problem, it is natural to consider game-theoretically optimal solutions which find the best solution in the worst-case scenario. This is possible for some special cases ([23, 2, 1, 72]) but difficult in general. On the other hand, many other efficient algorithms with optimal regret rate (but not exactly minimax optimal) have been proposed for different learning settings (such as the exponential

weights algorithm [38, 39], and follow the perturbed leader [63]). However, it is not always clear how to come up with these algorithms. Recent work by Rakhlin et al. [91] built a bridge between these two classes of methods by showing that many existing algorithms can indeed be derived from a minimax analysis followed by a series of relaxations.

We provide a parallel way to design expert algorithms by first converting the problem into a *drifting game*, and then applying a minimax analysis and relaxations. Drifting games [97] generalize Freund's "majority-vote game" [37] and subsume some well-studied boosting and online learning settings. It turns out the connections between drifting games and online learning go far beyond what has been discussed previously. In this section we start from connecting drifting games and the classic expert problem, and then discuss more generalization in Section 2.1.6.

2.1.1 Reviewing Drifting Games

We consider a simplified version of drifting games similar to the one described by Schapire and Freund [98, chap. 13] (also called chip games). This game proceeds through T rounds, and is played between a player and an adversary who controls N chips on the real line. The positions of these chips at the end of round t are denoted by $\boldsymbol{s}_t \in \mathbb{R}^N$, with each coordinate $s_{t,i}$ corresponding to the position of chip i. Initially, all chips are at position 0 so that $\boldsymbol{s}_0 = \mathbf{0}$. On every round $t = 1, \ldots, T$: the player first chooses a distribution $\boldsymbol{p}_t$ over the chips, then the adversary decides the movements of the chips $\boldsymbol{z}_t$ so that the new positions are updated as $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + \boldsymbol{z}_t$. Here, each $z_{t,i}$ has to be picked from a prespecified set $B \subset \mathbb{R}$, and more importantly, satisfy the constraint $\boldsymbol{p}_t \cdot \boldsymbol{z}_t \geq \beta \geq 0$ for some fixed constant β.

At the end of the game, each chip is associated with a nonnegative loss defined by $L(s_{T,i})$ for some nonincreasing function L mapping from the final position of the chip to $\mathbb{R}_+$. The goal of the player is to minimize the chips' average loss $\frac{1}{N}\sum_{i=1}^{N} L(s_{T,i})$

after T rounds. So intuitively, the player aims to "push" the chips to the right by assigning appropriate weights on them so that the adversary has to move them to the right by β in a weighted average sense on each round. This game captures many learning problems. For instance, binary classification via boosting can be translated into a drifting game by treating each training example as a chip (see [97] for details).

We regard a player's strategy $\mathcal{D}$ as a function mapping from the history of the adversary's decisions to a distribution that the player is going to play with, that is, $\boldsymbol{p}_t = \mathcal{D}(\boldsymbol{z}_{1:t-1})$ where $\boldsymbol{z}_{1:t-1}$ stands for $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{t-1}$. The player's worst case loss using this algorithm is then denoted by $L_T(\mathcal{D})$. The minimax optimal loss of the game is computed by the following expression:

$$\min_{\mathcal{D}} L_T(\mathcal{D}) = \min_{\boldsymbol{p}_1 \in \Delta_N} \max_{\boldsymbol{z}_1 \in \mathcal{Z}_{\boldsymbol{p}_1}} \cdots \min_{\boldsymbol{p}_T \in \Delta_N} \max_{\boldsymbol{z}_T \in \mathcal{Z}_{\boldsymbol{p}_T}} \frac{1}{N} \sum_{i=1}^{N} L(\sum_{t=1}^{T} z_{t,i}),$$

where Δ_N is the N dimensional simplex and $\mathcal{Z}_{\boldsymbol{p}} = B^N \cap \{\boldsymbol{z} : \boldsymbol{p} \cdot \boldsymbol{z} \geq \beta\}$ is assumed to be compact. A strategy $\mathcal{D}^*$ that realizes the minimum in $\min_{\mathcal{D}} L_T(\mathcal{D})$ is called a minimax optimal strategy.

A nearly optimal strategy and its analysis is originally given in [97], and a derivation by directly tackling the above minimax expression can be found in [98, chap. 13]. Specifically, a sequence of potential functions of a chip's position is defined recursively as follows:

$$\Phi_T(s) = L(s), \quad \Phi_{t-1}(s) = \min_{w \in \mathbb{R}_+} \max_{z \in B} (\Phi_t(s+z) + w(z-\beta)). \tag{2.1}$$

Intuitively, the potential function measures the contribution to the final loss of a chip at a specific position, assuming that both the player and the adversary will play optimally afterwards. Let $w_{t,i}$ be the weight that realizes the minimum in the

definition of $\Phi_{t-1}(s_{t-1,i})$, that is,

$$w_{t,i} \in \arg\min_{w} \max_{z} (\Phi_t(s_{t-1,i} + z) + w(z - \beta)).$$

Then the player's strategy is to set $p_{t,i} \propto w_{t,i}$. The key property of this strategy is that it assures that the sum of the potentials over all the chips never increases, connecting the player's final loss with the potential at time 0 as follows:

$$\frac{1}{N}\sum_{i=1}^{N} L(s_{T,i}) \leq \frac{1}{N}\sum_{i=1}^{N} \Phi_T(s_{T,i}) \leq \frac{1}{N}\sum_{i=1}^{N} \Phi_{T-1}(s_{T-1,i}) \leq \cdots \leq \frac{1}{N}\sum_{i=1}^{N} \Phi_0(s_{0,i}) = \Phi_0(0). \tag{2.2}$$

It has been shown in [97] that this upper bound on the loss is optimal in a very strong sense.

Moreover, in some cases the potential functions have nice closed forms and thus the algorithm can be efficiently implemented. For example, in the boosting setting, B is simply $\{-1, +1\}$, and one can verify $\Phi_t(s) = \frac{1+\beta}{2}\Phi_{t+1}(s+1) + \frac{1-\beta}{2}\Phi_{t+1}(s-1)$ and $w_{t,i} = \frac{1}{2}\left(\Phi_t(s_{t-1,i} - 1) - \Phi_t(s_{t-1,i} + 1)\right)$. With the loss function $L(s)$ being $\mathbf{1}\{s \leq 0\}$, these can be further simplified and eventually give exactly the boost-by-majority algorithm [37].

2.1.2 Algorithmic Equivalence

We now show strong connections between drifting games and the expert problem [38]. In the expert problem, a player tries to earn as much as possible (or lose as little as possible) by cleverly spreading a fixed amount of money to bet on a set of actions on each day. Formally, the game proceeds for T rounds, and on each round $t = 1, \ldots, T$: the player chooses a distribution $\boldsymbol{p}_t$ over N actions, then the adversary decides the actions' losses $\boldsymbol{\ell}_t$ (i.e. action i incurs loss $\ell_{t,i} \in [0, 1]$) which are revealed to the player. The player suffers a weighted average loss $\boldsymbol{p}_t \cdot \boldsymbol{\ell}_t$ at the end of this round. The goal of

the player is to minimize his regret, which is usually defined as the difference between his total loss and the loss of the best action.

Here, we consider an even more general notion of regret studied in [66, 65, 26, 29], which we call ϵ-*regret*. Suppose the actions are ordered according to their total losses after T rounds (i.e. $\sum_{t=1}^{T} \ell_{t,i}$) from smallest to largest, and let i_ϵ be the index of the action that is the $\lceil N\epsilon \rceil$-th element in the sorted list ($0 < \epsilon \leq 1$). Now, ϵ-regret is defined as $R_T^\epsilon(\boldsymbol{p}_{1:T}, \boldsymbol{\ell}_{1:T}) = \sum_{t=1}^{T} \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t - \sum_{t=1}^{T} \ell_{t,i_\epsilon}$. In other words, ϵ-regret measures the difference between the player's loss and the loss of the $\lceil N\epsilon \rceil$-th best action (recovering the usual regret with $\epsilon \leq 1/N$), and sublinear ϵ-regret implies that the player's loss is almost as good as all but the top ϵ fraction of actions. Similarly, $R_T^\epsilon(\mathcal{H})$ denotes the worst case ϵ-regret for a specific algorithm $\mathcal{H}$. For convenience, when $\epsilon \leq 0$ or $\epsilon > 1$, we define ϵ-regret to be ∞ or $-\infty$ respectively.

Now we show how this expert problem is highly related to drifting games. Consider a variant of drifting games where $B = [-1, 1], \beta = 0$ and $L(s) = \mathbf{1}\{s \leq -R\}$ for some constant R. Additionally, we impose an extra restriction on the adversary: $|z_{t,i} - z_{t,j}| \leq 1$ for all i and j. In other words, the difference between any two chips' movements is at most 1. We denote this specific variant of drifting games by DGv1 (summarized in Figure 2.1) and a corresponding algorithm by $\mathcal{D}_R$ to emphasize the dependence on R. The reductions in Algorithms 1 and 2 and Theorem 1 show that DGv1 and the expert problem are algorithmically equivalent (note that both conversions are valid). By Theorem 1, it is clear that the minimax optimal algorithm for one setting is also minimax optimal for the other under these conversions.

Theorem 1. *DGv1 and the expert problem are algorithmically equivalent in the following sense:*

(1) Algorithm 1 produces a DGv1 algorithm $\mathcal{D}_R$ satisfying $L_T(\mathcal{D}_R) \leq i/N$ where $i \in \{0, \ldots, N\}$ is such that $R_T^{(i+1)/N}(\mathcal{H}) < R \leq R_T^{i/N}(\mathcal{H})$.

DGv1

Given: a loss function $L(s) = \mathbf{1}\{s \leq -R\}$.
For $t = 1, \ldots, T$:

1. The player chooses a distribution $\boldsymbol{p}_t$ over N chips.
2. The adversary decides the movement of each chip $z_{t,i} \in [-1, 1]$ subject to $\boldsymbol{p}_t \cdot \boldsymbol{z}_t \geq 0$ and $|z_{t,i} - z_{t,j}| \leq 1$ for all i and j.

The player suffers loss $\sum_{i=1}^{N} L(\sum_{t=1}^{T} z_{t,i})$.

Figure 2.1: Protocol for a drifting game variant DGv1

Algorithm 1 Conversion of an Expert Algorithm $\mathcal{H}$ to a DGv1 Algorithm $\mathcal{D}_R$

Input: An Expert Algorithm $\mathcal{H}$
for $t = 1$ **to** T **do**
  Query $\mathcal{H}$: $\boldsymbol{p}_t = \mathcal{H}(\boldsymbol{\ell}_{1:t-1})$.
  Set: $\mathcal{D}_R(\boldsymbol{z}_{1:t-1}) = \boldsymbol{p}_t$.
  Receive movements $\boldsymbol{z}_t$ from the adversary.
  Set: $\ell_{t,i} = z_{t,i} - \min_j z_{t,j}$, $\forall i$.
end for

(2) Algorithm 2 produces an expert algorithm $\mathcal{H}$ with $R_T^{\epsilon}(\mathcal{H}) < R$ for any R such that $L_T(\mathcal{D}_R) < \epsilon$.

Proof. We first show that both conversions are valid. In Algorithm 1, it is clear that $\ell_{t,i} \geq 0$. Also, $\ell_{t,i} \leq 1$ is guaranteed due to the extra restriction of DGv1. For Algorithm 2, $z_{t,i}$ lies in $B = [-1, 1]$ since $\ell_{t,i} \in [0, 1]$, and direct computation shows $\boldsymbol{p}_t \cdot \boldsymbol{z}_t = 0 \geq \beta (= 0)$ and $|z_{t,i} - z_{t,j}| = |\ell_{t,i} - \ell_{t,j}| \leq 1$ for all i and j.

(1) For any choices of $\boldsymbol{z}_t$, we have

$$\sum_{i=1}^{N} L(s_{T,i}) = \sum_{i=1}^{N} L\left(\sum_{t=1}^{N} z_{t,i}\right) \leq \sum_{i=1}^{N} L\left(\sum_{t=1}^{N} (z_{t,i} - \boldsymbol{p}_t \cdot \boldsymbol{z}_t)\right),$$

where the inequality holds since $\boldsymbol{p}_t \cdot \boldsymbol{z}_t$ is required to be nonnegative and L is a nonincreasing function. By Algorithm 1, $z_{t,i} - \boldsymbol{p}_t \cdot \boldsymbol{z}_t$ is equal to $\ell_{t,i} - \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t$, leading

Algorithm 2 Conversion of a DGv1 Algorithm $\mathcal{D}_R$ to an Expert Algorithm $\mathcal{H}$

Input: A DGv1 Algorithm $\mathcal{D}_R$

1: **for** $t = 1$ **to** T **do**
2: Query $\mathcal{D}_R$: $\boldsymbol{p}_t = \mathcal{D}_R(\boldsymbol{z}_{1:t-1})$.
3: Set: $\mathcal{H}(\boldsymbol{\ell}_{1:t-1}) = \boldsymbol{p}_t$.
4: Receive losses $\boldsymbol{\ell}_t$ from the adversary.
5: Set: $z_{t,i} = \ell_{t,i} - \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t$, $\forall i$.
6: **end for**

to

$$\sum_{i=1}^{N} L(s_{T,i}) \leq \sum_{i=1}^{N} L\left(\sum_{t=1}^{N} (\ell_{t,i} - \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t)\right) = \sum_{i=1}^{N} \mathbf{1}\left\{R \leq \sum_{t=1}^{N} (\boldsymbol{p}_t \cdot \boldsymbol{\ell}_t - \ell_{t,i})\right\}.$$

Since $R_T^{(i+1)/N}(\mathcal{H}) < R \leq R_T^{i/N}(\mathcal{H})$, we must have $\sum_{t=1}^{N} (\boldsymbol{p}_t \cdot \boldsymbol{\ell}_t - \ell_{t,j}) < R$ except for the best i actions, which means $\sum_{i=1}^{N} L(s_{T,i}) \leq i$. This holds for any choices of $\boldsymbol{z}_t$, so $L_T(\mathcal{D}_R) \leq i/N$.

(2) By Algorithm 2 and the condition $L_T(D_R) < \epsilon$, we have

$$\frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\left\{R \leq \sum_{t=1}^{N} (\boldsymbol{p}_t \cdot \boldsymbol{\ell}_t - \ell_{t,i})\right\} = \frac{1}{N}\sum_{i=1}^{N} L(s_{T,i}) \leq L_T(D_R) < \epsilon,$$

which means there are at most $\lceil N\epsilon \rceil - 1$ actions satisfying $R \leq \sum_{t=1}^{N} (\boldsymbol{p}_t \cdot \boldsymbol{\ell}_t - \ell_{t,i})$, and thus $\sum_{t=1}^{N} (\boldsymbol{p}_t \cdot \boldsymbol{\ell}_t - \ell_{t,i_\epsilon}) < R$. Since this holds for any choices of $\boldsymbol{\ell}_t$, we have $R_T^{\epsilon}(\mathcal{H}) < R$. □

2.1.3 Relaxations

From now on we only focus on the direction of converting a drifting game algorithm into an expert algorithm. In order to derive a minimax expert algorithm, Theorem 1 tells us it suffices to derive minimax DGv1 algorithms. Exact minimax analysis is usually difficult, and appropriate relaxations seem to be necessary. To make use of the existing analysis for standard drifting games, the first obvious relaxation is to drop the

additional restriction in DGv1, that is, $|z_{t,i} - z_{t,j}| \leq 1$ for all i and j. Doing this will lead to the exact setting discussed in [83] where a near optimal strategy is proposed using the recipe in Eq. (2.1). It turns out that this relaxation is reasonable and does not give too much more power to the adversary. To see this, first recall that results from [83], written in our notation, state that $\min_{\mathcal{D}_R} L_T(\mathcal{D}_R) \leq \frac{1}{2^T} \sum_{j=0}^{\frac{T-R}{2}} \binom{T+1}{j}$, which, by Hoeffding's inequality, is upper bounded by $2\exp\left(-\frac{(R+1)^2}{2(T+1)}\right)$. Second, statement (2) in Theorem 1 clearly remains valid if the input of Algorithm 2 is a drifting game algorithm for this relaxed version of DGv1. Therefore, by setting $\epsilon > 2\exp\left(-\frac{(R+1)^2}{2(T+1)}\right)$ and solving for R, we have $R_T^{\epsilon}(\mathcal{H}) \leq O\left(\sqrt{T\ln(\frac{1}{\epsilon})}\right)$, which is the known optimal regret rate for the expert problem, showing that we lose little due to this relaxation.

However, the algorithm proposed in [83] is not computationally efficient since the potential functions $\Phi_t(s)$ do not have closed forms. To get around this, we would want the minimax expression in Eq. (2.1) to be easily solved, just like the case when $B = \{-1, 1\}$. It turns out that convexity would allow us to treat $B = [-1, 1]$ almost as $B = \{-1, 1\}$. Specifically, if each $\Phi_t(s)$ is a convex function of s, then due to the fact that the maximum of a convex function is always realized at the boundary of a compact region, we have

$$\min_{w\in\mathbb{R}_+} \max_{z\in[-1,1]} (\Phi_t(s+z) + wz) = \min_{w\in\mathbb{R}_+} \max_{z\in\{-1,1\}} (\Phi_t(s+z) + wz) = \frac{\Phi_t(s-1) + \Phi_t(s+1)}{2}, \tag{2.3}$$

with $w = (\Phi_t(s-1) - \Phi_t(s+1))/2$ realizing the minimum. Since the 0-1 loss function $L(s)$ is not convex, this motivates us to find a convex surrogate of $L(s)$. Fortunately, relaxing the equality constraints in Eq. (2.1) does not affect the key property of Eq. (2.2) as we will show in the proof of Theorem 2. "Compiling out" the input of Algorithm 2, we thus have our general recipe (Algorithm 3) for designing expert algorithms with the following regret guarantee.

Algorithm 3 A General Expert Algorithm $\mathcal{H}$

Input: A convex, nonincreasing, nonnegative function $\Phi_T(s)$.

for $t = T$ **to** 1 **do**
    Find a convex function $\Phi_{t-1}(s)$ s.t. $\forall s$,
    $\Phi_t(s-1) + \Phi_t(s+1) \leq 2\Phi_{t-1}(s)$.
end for
Set: $\boldsymbol{s}_0 = \mathbf{0}$.
for $t = 1$ **to** T **do**
    Set: $\mathcal{H}(\boldsymbol{\ell}_{1:t-1}) = \boldsymbol{p}_t$ s.t. $p_{t,i} \propto \Phi_t(s_{t-1,i} - 1) - \Phi_t(s_{t-1,i} + 1)$.
    Receive losses $\boldsymbol{\ell}_t$ and set $s_{t,i} = s_{t-1,i} + \ell_{t,i} - \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t,\ \forall i$.
end for

Theorem 2. *For Algorithm 3, if R and ϵ are such that $\Phi_0(0) < \epsilon$ and $\Phi_T(s) \geq \mathbf{1}\{s \leq -R\}$ for all $s \in \mathbb{R}$, then $R_T^{\epsilon}(\mathcal{H}) < R$.*

Proof. It suffices to show that Eq. (2.2) holds so that the theorem follows by a direct application of statement (2) of Theorem 1. Let

$$w_{t,i} = (\Phi_t(s_{t-1,i} - 1) - \Phi_t(s_{t-1,i} + 1))/2.$$

Then we have

$$\sum_i \Phi_t(s_{t,i}) \leq \sum_i \left(\Phi_t(s_{t-1,i} + z_{t,i}) + w_{t,i} z_{t,i}\right)$$

since $p_{t,i} \propto w_{t,i}$ and $\boldsymbol{p}_t \cdot \boldsymbol{z}_t \geq 0$. On the other hand, by Eq. (2.3), we have

$$\begin{aligned}\Phi_t(s_{t-1,i} + z_{t,i}) + w_{t,i} z_{t,i} &\leq \min_{w \in \mathbb{R}_+} \max_{z \in [-1,1]} \left(\Phi_t(s_{t-1,i} + z) + wz\right) \\ &= \frac{1}{2}\left(\Phi_t(s_{t-1,i} - 1) + \Phi_t(s_{t-1,i} + 1)\right),\end{aligned}$$

which is at most $\Phi_{t-1}(s_{t-1,i})$ by Algorithm 3. This shows $\sum_i \Phi_t(s_{t,i}) \leq \sum_i \Phi_{t-1}(s_{t-1,i})$ and Eq. (2.2) follows. □

Theorem 2 tells us that if solving $\Phi_0(0) < \epsilon$ for R gives $R > \underline{R}$ for some value $\underline{R}$, then the regret of Algorithm 3 is less than any value that is greater than $\underline{R}$, meaning the regret is at most $\underline{R}$.

Table 2.1: Different potential based algorithms and comparisons with NormalHedge

	EXP	2-norm	NormalHedge.DT	NormalHedge
$\Phi_T(s)$	$e^{-\eta(s+R)}$	$a[s]_-^2$	$a\left(e^{[s]_-^2/3T}-1\right)$	N/A
$p_{t,i} \propto$	$e^{-\eta s_{t-1,i}}$	$[s_{t-1,i}-1]_-^2$ $-[s_{t-1,i}+1]_-^2$	$e^{[s_{t-1,i}-1]_-^2/3t}$ $-e^{[s_{t-1,i}+1]_-^2/3t}$	$-[s_{t-1,i}]_- e^{[s_{t-1,i}]_-^2/c}$ (c is s.t. $\sum_i e^{[s_{t-1,i}]_-^2/c} = Ne$)
$R_T^\epsilon(\mathcal{H})$	$O\left(\sqrt{T\ln\frac{1}{\epsilon}}\right)$	$O\left(\sqrt{T/\epsilon}\right)$	$O\left(\sqrt{T\ln\frac{\ln T}{\epsilon}}\right)$	$O\left(\sqrt{T\ln\frac{1}{\epsilon}+\ln^2 N}\right)$
Adaptive?	No	Yes	Yes	Yes

2.1.4 Designing Potentials and Algorithms

Now we are ready to recover existing algorithms and develop new ones by choosing an appropriate potential $\Phi_T(s)$ as Algorithm 3 suggests. We will discuss three different algorithms below, and summarize these examples in Table 2.1.

Exponential Weights (EXP) Algorithm. Exponential loss is an obvious choice for $\Phi_T(s)$ as it has been widely used as the convex surrogate of the 0-1 loss function in the literature. It turns out that this will lead to the well-known exponential weights algorithm [38, 39]. Specifically, we pick $\Phi_T(s)$ to be $\exp\left(-\eta(s+R)\right)$ which exactly upper bounds $\mathbf{1}\{s \le -R\}$. To compute $\Phi_t(s)$ for $t \le T$, we simply let $\Phi_t(s-1)+\Phi_t(s+1) \le 2\Phi_{t-1}(s)$ hold with equality. Indeed, direct computations show that all $\Phi_t(s)$ share a similar form: $\Phi_t(s) = \left(\frac{e^\eta+e^{-\eta}}{2}\right)^{T-t} \cdot \exp\left(-\eta(s+R)\right)$. Therefore, according to Algorithm 3, the player's strategy is to set

$$p_{t,i} \propto \Phi_t(s_{t-1,i}-1) - \Phi_t(s_{t-1,i}+1) \propto \exp\left(-\eta s_{t-1,i}\right),$$

which is exactly the same as EXP (note that R becomes irrelevant after normalization). To derive regret bounds, it suffices to require $\Phi_0(0) < \epsilon$, which is equivalent to $R > \frac{1}{\eta}\left(\ln(\frac{1}{\epsilon}) + T\ln\frac{e^\eta+e^{-\eta}}{2}\right)$. By Theorem 2 and Hoeffding's lemma (see [22, Lemma A.1]), we thus know $R_T^\epsilon(\mathcal{H}) \le \frac{1}{\eta}\ln\left(\frac{1}{\epsilon}\right) + \frac{T\eta}{2} = \sqrt{2T\ln\left(\frac{1}{\epsilon}\right)}$ where the last step is by

optimally tuning η to be $\sqrt{2(\ln \frac{1}{\epsilon})/T}$. Note that this algorithm is *not adaptive* in the sense that it requires knowledge of T and ϵ to set the parameter η.

We have thus recovered the well-known exponential weights algorithm and given a new analysis using the drifting-games framework. More importantly, as in [91], this derivation may shed light on why this algorithm works and where it comes from, namely, a minimax analysis followed by a series of relaxations, starting from a reasonable surrogate of the 0-1 loss function.

2-norm Algorithm. We next move on to another simple convex surrogate: $\Phi_T(s) = a[s]_-^2 \geq \mathbf{1}\{s \leq -1/\sqrt{a}\}$, where a is some positive constant and $[s]_- = \min\{0, s\}$ represents a truncating operation. The following lemma shows that $\Phi_t(s)$ can also be simply described.

Lemma 1. *If $a > 0$, then $\Phi_t(s) = a\left([s]_-^2 + T - t\right)$ satisfies $\Phi_t(s-1) + \Phi_t(s+1) \leq 2\Phi_{t-1}(s)$.*

Proof. It suffices to show $[s-1]_-^2 + [s+1]_-^2 \leq 2[s]_-^2 + 2$. When $s \geq 0$, we have

$$[s-1]_-^2 + [s+1]_-^2 = [s-1]_-^2 \leq 1 < 2 = 2[s]_-^2 + 2.$$

When $s < 0$, we also have

$$[s-1]_-^2 + [s+1]_-^2 \leq (s-1)^2 + (s+1)^2 = 2s^2 + 2 = 2[s]_-^2 + 2,$$

completing the proof. □

Thus, Algorithm 3 can again be applied. The resulting algorithm is extremely concise:

$$p_{t,i} \propto \Phi_t(s_{t-1,i} - 1) - \Phi_t(s_{t-1,i} + 1) \propto [s_{t-1,i} - 1]_-^2 - [s_{t-1,i} + 1]_-^2.$$

We call this the "2-norm" algorithm since it resembles the p-norm algorithm [22] when $p = 2$. The difference is that the p-norm algorithm sets the weights proportional to the derivative of potentials, instead of the difference of them as we are doing here. A somewhat surprising property of this algorithm is that it is totally adaptive and parameter-free (since a disappears under normalization), a property that we usually do not expect to obtain from a minimax analysis. Direct application of Theorem 2 ($\Phi_0(0) = aT < \epsilon \Leftrightarrow 1/\sqrt{a} > \sqrt{T/\epsilon}$) shows that its regret achieves the optimal dependence on the horizon T.

Corollary 1. *Algorithm 3 with potential $\Phi_t(s)$ defined in Lemma 1 produces an expert algorithm $\mathcal{H}$ such that $R_T^\epsilon(\mathcal{H}) \leq \sqrt{T/\epsilon}$ simultaneously for all T and ϵ.*

NormalHedge.DT. The regret for the 2-norm algorithm does not have the optimal dependence on ϵ. An obvious follow-up question would be whether it is possible to derive an adaptive algorithm that achieves the optimal rate $O(\sqrt{T \ln(1/\epsilon)})$ simultaneously for all T and ϵ using our framework. An even deeper question is: instead of choosing convex surrogates in a seemingly arbitrary way, is there a more natural way to find the *right* choice of $\Phi_T(s)$?

To answer these questions, we recall that the reason why the 2-norm algorithm can get rid of the dependence on ϵ is that ϵ appears merely in the multiplicative constant a that does not play a role after normalization. This motivates us to let $\Phi_T(s)$ in the form of $\epsilon F(s)$ for some $F(s)$. On the other hand, from Theorem 2, we also want $\epsilon F(s)$ to upper bound the 0-1 loss function $\mathbf{1}\{s \leq -\sqrt{dT \ln(1/\epsilon)}\}$ for some constant d. Taken together, this is telling us that the right choice of $F(s)$ should be of the form $\Theta\left(\exp(s^2/T)\right)$ (similar potential was also proposed in recent work [79, 87] for a different setting). Of course we still need to refine it to satisfy the monotonicity

and other properties. We define $\Phi_T(s)$ formally and more generally as:

$$\Phi_T(s) = a\left(\exp\left(\frac{[s]_-^2}{dT}\right) - 1\right) \geq \mathbf{1}\left\{s \leq -\sqrt{dT\ln\left(\frac{1}{a}+1\right)}\right\},$$

where a and d are some positive constants. This time it is more involved to figure out what other $\Phi_t(s)$ should be. The following lemma addresses this issue (proof deferred to Appendix A.1).

Lemma 2. *If* $b_t = 1 - \frac{1}{2}\sum_{\tau=t+1}^{T}\left(\exp\left(\frac{4}{d\tau}\right) - 1\right), a > 0, d \geq 3$ *and* $\Phi_t(s) = a\left(\exp\left(\frac{[s]_-^2}{dt}\right) - b_t\right)$ *(define* $\Phi_0(s) \equiv a(1-b_0)$*), then we have* $\Phi_t(s-1) + \Phi_t(s+1) \leq 2\Phi_{t-1}(s)$ *for all* $s \in \mathbb{R}$ *and* $t = 2, \ldots, T$*. Moreover, Eq.* (2.2) *still holds.*

Note that even if $\Phi_1(s-1) + \Phi_1(s+1) \leq 2\Phi_0(s)$ is not valid in general, Lemma 2 states that Eq. (2.2) still holds. Thus Algorithm 3 can indeed still be applied, leading to our new algorithm:

$$p_{t,i} \propto \Phi_t(s_{t-1,i} - 1) - \Phi_t(s_{t-1,i} + 1) \propto \exp\left(\frac{[s_{t-1,i}-1]_-^2}{dt}\right) - \exp\left(\frac{[s_{t-1,i}+1]_-^2}{dt}\right).$$

Here, d seems to be an extra parameter, but in fact, simply setting $d = 3$ is good enough:

Corollary 2. *Algorithm 3 with potential* $\Phi_t(s)$ *defined in Lemma 2 and* $d = 3$ *produces an expert algorithm* $\mathcal{H}$ *such that the following holds simultaneously for all* T *and* ϵ*:*

$$R_T^\epsilon(\mathcal{H}) \leq \sqrt{3T\ln\left(\frac{1}{2\epsilon}\left(e^{4/3}-1\right)(\ln T + 1) + 1\right)} = O\left(\sqrt{T\ln(1/\epsilon) + T\ln\ln T}\right).$$

Proof. Recall that $\Phi_T(s) \geq \mathbf{1}\left\{s \leq -\sqrt{dT\ln\left(\frac{1}{a}+1\right)}\right\}$. So by setting $\Phi_0(0) = a(1-b_0) < \epsilon$ and applying Theorem 2, we arrive at

$$R_T^\epsilon(\mathcal{H}) \leq \sqrt{dT\ln\left(\frac{1-b_0}{\epsilon}+1\right)}.$$

It suffices to upper bound $1 - b_0$, which, by definition, is $\frac{1}{2}\sum_{t=1}^{T}\left(\exp\left(\frac{4}{dt}\right) - 1\right)$. Since $e^x - 1 \le \frac{e^c-1}{c}x$ for any $x \in [0, c]$, we have

$$\sum_{t=1}^{T}\left(\exp\left(\frac{4}{dt}\right) - 1\right) \le (e^{4/d} - 1)\sum_{t=1}^{T}\frac{1}{t} \le (e^{4/d} - 1)(\ln T + 1).$$

Plugging $d = 3$ gives the corollary. □

We have thus proposed a parameter-free adaptive algorithm with optimal regret rate (ignoring the $\ln\ln T$ term) using our drifting-games framework. In fact, our algorithm bears a striking similarity to NormalHedge [26], the first algorithm that has this kind of adaptivity. We thus name our algorithm NormalHedge.DT[1]. We include NormalHedge in Table 2.1 for comparison. One can see that the main differences are: 1) On each round NormalHedge performs a numerical search to find out the right parameter used in the exponents; 2) NormalHedge uses the derivative of potentials as weights.

Compared to NormalHedge, the regret bound for NormalHedge.DT has no explicit dependence on N, but has a slightly worse dependence on T (indeed $\ln\ln T$ is almost negligible). We emphasize other advantages of our algorithm over NormalHedge: 1) NormalHedge.DT is more computationally efficient especially when N is very large, since it does not need a numerical search for each round; 2) our analysis is arguably simpler and more intuitive than the one in [26]; 3) as we will discuss in Section 2.1.6, NormalHedge.DT can be easily extended to deal with the more general online convex optimization problem where the number of actions is infinitely large, while it is not clear how to do that for NormalHedge by generalizing the analysis in [26]. Indeed, the extra dependence on the number of actions N for the regret of NormalHedge makes this generalization even seem impossible. Finally, we will later see that NormalHedge.DT outperforms NormalHedge in experiments. Despite the differences, it is

[1] "DT" stands for discrete time.

worth noting that both algorithms assign zero weight to some actions on each round, an appealing property when N is huge.

2.1.5 High Probability Bounds

We briefly remark how our drifting-games framework also allows us to study high probability bounds without resorting to any concentration results. We consider a common variant of the expert problem: on each round, instead of choosing a distribution $\boldsymbol{p}_t$, the player has to randomly pick a single action i_t, while the adversary decides the losses $\boldsymbol{\ell}_t$ at the same time (without seeing i_t). For now we only focus on the player's regret to the best action: $R_T(i_{1:T}, \boldsymbol{\ell}_{1:T}) = \sum_{t=1}^T \ell_{t,i_t} - \min_i \sum_{t=1}^T \ell_{t,i}$. Notice that the regret is now a random variable, and we are interested in a bound that holds with high probability. Using Azuma's inequality, standard analysis (see for instance [22, Lemma 4.1]) shows that the player can simply draw i_t according to $\boldsymbol{p}_t = \mathcal{H}(\boldsymbol{\ell}_{1:t-1})$, the output of a standard expert algorithm, and suffers regret at most $R_T(\mathcal{H}) + \sqrt{T \ln(1/\delta)}$ with probability $1 - \delta$. Below we recover similar results as a simple side product of our drifting-games analysis *without* resorting to concentration results, such as Azuma's inequality.

For this, we only need to modify Algorithm 3 by setting $z_{t,i} = \ell_{t,i} - \ell_{t,i_t}$. The restriction $\boldsymbol{p}_t \cdot \boldsymbol{z}_t \geq 0$ is then relaxed to hold in expectation. Moreover, it is clear that Eq. (2.2) also still holds in expectation. On the other hand, by definition and the union bound, one can show that $\sum_i \mathbb{E}[L(s_{T,i})] = \sum_i \Pr[s_{T,i} \leq -R] \geq \Pr[R_T(i_{1:T}, \boldsymbol{\ell}_{1:T}) \geq R]$. So setting $\Phi_0(0) = \delta$ shows that the regret is smaller than R with probability $1 - \delta$. Therefore, for example, if EXP is used, then the regret would be at most $\sqrt{2T \ln(N/\delta)}$ with probability $1 - \delta$, giving basically the same bound as the standard analysis. One draw back is that EXP would need δ as a parameter. However, this can again be addressed by NormalHedge.DT for the exact same reason

that NormalHedge.DT is independent of ϵ. We have thus derived high probability bounds without using any concentration inequalities.

2.1.6 Generalizations

We have so far shown strong relations between drifting games and a specific online learning problem. However, the connections go far beyond that as we briefly discuss here.

Multi-armed Bandit (MAB) Problem. We first consider the multi-armed bandit problem [93]. The only difference between the expert problem (randomized version) and the non-stochastic MAB problem [7] is that on each round, after picking i_t, the player only sees the loss for this single action ℓ_{t,i_t} instead of the whole vector $\boldsymbol{\ell}_t$. The goal is still to compete with the best action. A common technique used in the bandit setting is to build an unbiased estimator $\hat{\boldsymbol{\ell}}_t$ for the losses, which in this case could be $\hat{\ell}_{t,i} = \mathbf{1}\{i = i_t\} \cdot \ell_{t,i_t}/p_{t,i_t}$. Then algorithms such as EXP can be used by replacing $\boldsymbol{\ell}_t$ with $\hat{\boldsymbol{\ell}}_t$, leading to the EXP3 algorithm [7] with regret $O(\sqrt{TN \ln N})$.

One might expect that Algorithm 3 would also work well by replacing $\boldsymbol{\ell}_t$ with $\hat{\boldsymbol{\ell}}_t$. However, doing so breaks an important property of the movements $z_{t,i}$: boundedness. Indeed, Eq. (2.3) no longer makes sense if z could be infinitely large, even if in expectation it is still in $[-1, 1]$ (note that $z_{t,i}$ is now a random variable). It turns out that we can address this issue by imposing a variance constraint on $z_{t,i}$. Formally, we consider a variant of drifting games where on each round, the adversary picks a random movement $z_{t,i}$ for each chip such that: $z_{t,i} \geq -1, \mathbb{E}_t[z_{t,i}] \leq 1, \mathbb{E}_t[z_{t,i}^2] \leq 1/p_{t,i}$ and $\mathbb{E}_t[\boldsymbol{p}_t \cdot \boldsymbol{z}_t] \geq 0$. We call this variant DGv2 and summarize it in Appendix A.2. The standard minimax analysis and the derivation of potential functions need to be modified in a certain way for DGv2, as stated in Theorem 20 (Appendix A.2). Using the analysis for DGv2, we propose a general recipe for designing MAB algorithms

in a similar way as for the expert problem and also recover EXP3 (see Algorithm 24 and Theorem 21 in Appendix A.2). Unfortunately so far we do not know other appropriate potentials due to some technical difficulties. We conjecture, however, that there is a potential function that could recover the poly-INF algorithm [5, 6] or give its variants that achieve the optimal regret $O(\sqrt{TN})$.

Online Convex Optimization. We next consider the most general online convex optimization (OCO) setting [112] that is mentioned in Chapter 1. Let $\mathcal{K} \subset \mathbb{R}^d$ be a compact convex set, and $\mathcal{F}$ be a set of convex functions with range $[0, 1]$ on $\mathcal{K}$. Recall that in this setting, On each round t, the learner chooses a point $\boldsymbol{w}_t \in S$, and the adversary chooses a loss function $f_t \in \mathcal{F}$ (knowing $\boldsymbol{w}_t$). The learner then suffers loss $f_t(\boldsymbol{w}_t)$. The regret after T rounds is $R_T = \sum_{t=1}^{T} f_t(\boldsymbol{w}_t) - \min_{\boldsymbol{w}\in\mathcal{K}} \sum_{t=1}^{T} f_t(\boldsymbol{w})$. There are two general approaches to OCO: one builds on convex optimization theory [101], and the other generalizes the exponential weights algorithm to a continuous space [31, 84]. We will see how the drifting-games framework can recover the latter method and also leads to new ones.

To do so, we introduce a continuous variant of drifting games (DGv3, see Appendix A.3). There are now infinitely many chips, one for each point in $\mathcal{K}$. On round t, the player needs to choose a distribution over the chips, that is, a probability density function $p_t(\boldsymbol{w})$ on S. Then the adversary decides the movements for each chip, that is, a function $z_t(\boldsymbol{w})$ with range $[-1, 1]$ on S (not necessarily convex or continuous), subject to a constraint $\mathbb{E}_{\boldsymbol{w}\sim p_t}[z_t(\boldsymbol{w})] \geq 0$. At the end, each point $\boldsymbol{w}$ is associated with a loss $L(\boldsymbol{w}) = \mathbf{1}\{\sum_t z_t(\boldsymbol{w}) \leq -R\}$, and the player aims to minimize the total loss $\int_{\boldsymbol{w}\in\mathcal{K}} L(\boldsymbol{w}) d\boldsymbol{w}$.

OCO can be converted into DGv3 by setting $z_t(\boldsymbol{w}) = f_t(\boldsymbol{w}) - f_t(\boldsymbol{w}_t)$ and predicting $\boldsymbol{w}_t = \mathbb{E}_{\boldsymbol{w}\sim p_t}[\boldsymbol{w}] \in \mathcal{K}$. The constraint $\mathbb{E}_{\boldsymbol{w}\sim p_t}[z_t(\boldsymbol{w})] \geq 0$ holds by the convexity of f_t. Moreover, it turns out that the minimax analysis and potentials for DGv1 can readily

Algorithm 4 NH-Boost.DT

Input: Training examples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}, i = 1, \ldots, N$.
Input: A weak learning algorithm.
Input: Number of rounds T.
Output: A Hypothesis $H(\boldsymbol{x}) : \mathbb{R}^d \to \{-1, +1\}$.

Set: $\boldsymbol{s}_0 = \boldsymbol{0}$.
for $t = 1$ **to** T **do**
    Set: $p_{t,i} \propto \exp\left([s_{t-1,i} - 1]_-^2/3t\right) - \exp\left([s_{t-1,i} + 1]_-^2/3t\right), \ \forall i$.
    Invoke the weak learning algorithm to get h_t with edge $\gamma_t = \frac{1}{2}\sum_i p_{t,i} y_i h_t(\boldsymbol{x}_i)$.
    Set: $s_{t,i} = s_{t-1,i} + \frac{1}{2} y_i h_t(\boldsymbol{x}_i) - \gamma_t, \ \forall i$.
end for
Set: $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} h_t(\boldsymbol{x}))$.

be used here, and the notion of ϵ-regret, now generalized to the OCO setting, measures the difference of the player's loss and the loss of a best fixed point in a subset of S that excludes the top ϵ fraction of points. With different potentials, we obtain versions of each of the three algorithms of Section 2.1.4 generalized to this setting, with the same ϵ-regret bounds as before. Again, two of these methods are adaptive and parameter-free. To derive bounds for the usual regret, at first glance it seems that we have to set ϵ to be close to zero, leading to a meaningless bound. Nevertheless, this is addressed by Theorem 22 using similar techniques in [57], giving the usual $O(\sqrt{dT \ln T})$ regret bound. All details can be found in Appendix A.3.

2.2 Applications to Boosting

There is a deep and well-known connection between the expert problem and boosting [38, 98]. In principle, every expert algorithm with sublinear regret can be converted into a boosting algorithm; for instance, this is how AdaBoost was derived from the exponential weights algorithm [38]. In the same way, NormalHedge.DT can be converted into a new boosting algorithm with some nice properties, which we discuss in details below.

Algorithm 5 NH-Boost

Input: Training examples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{-1, +1\}, i = 1, \ldots, N$.
Input: A weak learning algorithm.
Input: Number of rounds T.
Output: A Hypothesis $H(\boldsymbol{x}) : \mathbb{R}^d \to \{-1, +1\}$.

Set: $\boldsymbol{s}_0 = \boldsymbol{0}$.
for $t = 1$ **to** T **do**
  if $t = 1$ **then**
    Set: $\boldsymbol{p}_1$ to be a uniform distribution.
  else
    Find: c such that $\sum_{i=1}^N \exp\left([s_{t-1,i}]_-^2/c\right) = Ne$.
    Set: $p_{t,i} \propto -[s_{t-1,i}]_- \exp\left([s_{t-1,i}]_-^2/c\right),\ \forall i$.
  end if
  Invoke the weak learning algorithm to get h_t with edge $\gamma_t = \frac{1}{2}\sum_i p_{t,i} y_i h_t(\boldsymbol{x}_i)$.
  Set: $s_{t,i} = s_{t-1,i} + \frac{1}{2} y_i h_t(\boldsymbol{x}_i) - \gamma_t,\ \forall i$.
end for
Set: $H(\boldsymbol{x}) = \mathrm{sign}(\sum_{t=1}^T h_t(\boldsymbol{x}))$.

Consider using boosting for binary classification. We are given a set of training examples $(\boldsymbol{x}_i, y_i)_{i=1,\ldots,N}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ is an example and $y_i \in \{-1, +1\}$ is its label. A boosting algorithm proceeds for T rounds. On each round, a distribution $\boldsymbol{p}_t$ over the examples is computed and fed into a weak learning algorithm which returns a "weak" hypothesis $h_t : \mathbb{R}^d \to \{-1, +1\}$ with a guaranteed small edge, that is, $\gamma_t = \frac{1}{2}\sum_i p_{t,i} y_i h_t(\boldsymbol{x}_i) \geq \gamma > 0$. At the end, a linear combination of all h_t is computed as the final "strong" hypothesis which is expected to have low training error and potentially low generalization error.

The conversion of an expert algorithm into a boosting algorithm is to treat each example as an expert and set $\ell_{t,i} = \mathbf{1}\{h_t(\boldsymbol{x}_i) = y_i\}$ so that the booster tends to increase weights for those "hard" examples. The final hypothesis is a simple majority vote of all h_t, that is, $H(\boldsymbol{x}) = \mathrm{sign}(\sum_t h_t(\boldsymbol{x}))$. The *margin* of example $\boldsymbol{x}_i$ is defined as $\frac{1}{T}\sum_{t=1}^T y_i h_t(\boldsymbol{x}_i)$, that is, the difference between the fractions of correct hypotheses and incorrect hypotheses on this example. The boosting algorithms derived from NormalHedge.DT and NormalHedge in this way are given in Algorithm 4 and 5.

Studying the training error rate and the distribution of margins is essential in understanding the prediction accuracy of a boosting algorithm. Previous derivations of boosting algorithms using nonadaptive expert algorithms only show the number of rounds needed to achieve zero training error, and the minimum margin of examples at each time (see [98]). However, thanks to the adaptivity of NormalHedge.DT, here we can derive training error rate at each time and also the distribution of margins at each time for NH-Boost.DT.

Theorem 3. *After T rounds, the training error of NH-Boost.DT is of order $\tilde{O}(\exp(-\frac{1}{3}T\gamma^2))$, and the fraction of training examples with margin at most $\theta(\leq 2\gamma)$ is of order $\tilde{O}(\exp(-\frac{1}{3}T(\theta-2\gamma)^2))$.*

Proof. Let $(\tilde{\boldsymbol{x}}_i, \tilde{y}_i)_{i=1,\dots,N}$ be a permutation of the training examples such that their margins are sorted from smallest to largest: $\sum_t \tilde{y}_1 h_t(\tilde{\boldsymbol{x}}_1) \leq \cdots \leq \sum_t \tilde{y}_N h_t(\tilde{\boldsymbol{x}}_N)$, which also implies $\sum_t \mathbf{1}\{h_t(\tilde{\boldsymbol{x}}_1) = \tilde{y}_1\} \leq \cdots \leq \sum_t \mathbf{1}\{h_t(\tilde{\boldsymbol{x}}_N) = \tilde{y}_N\}$. Recall that NH-Boost.DT is essentially solving the expert problem using NormalHedge.DT with loss $\ell_{t,i} = \mathbf{1}\{h_t(\boldsymbol{x}_i) = y_i\}$. Therefore, the ϵ-regret bound together with the assumption on the weak learning algorithm implies: $\forall j \in \{1, \dots, N\}$,

$$\frac{1}{2} + \gamma \leq \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N} \boldsymbol{p}_{t,i}\mathbf{1}\{h_t(\boldsymbol{x}_i) = y_i\} \leq \frac{1}{T}\sum_{t=1}^{T}\mathbf{1}\{h_t(\tilde{\boldsymbol{x}}_j) = \tilde{y}_j\} + \frac{R_T^{j/N}}{T}, \tag{2.4}$$

where $R_T^{j/N} = \tilde{O}(\sqrt{3T\ln(N/j)})$ is the j/N-regret bound for NormalHedge.DT. So if j is such that $\gamma > R_T^{j/N}/T$, we have $\frac{1}{T}\sum_{t=1}^{T}\mathbf{1}\{h_t(\tilde{\boldsymbol{x}}_j) = \tilde{y}_j\} > \frac{1}{2}$, which is saying that example $(\tilde{\boldsymbol{x}}_j, \tilde{y}_j)$ will eventually be classified correctly by $H(\boldsymbol{x})$ due to the fact that $H(\boldsymbol{x})$ is taking a majority vote of all h_t. This is in fact true for all examples $(\tilde{\boldsymbol{x}}_i, \tilde{y}_i)$ such that $i \geq j$ and thus the training error rate will be at most $(j-1)/N$, which is of order $\tilde{O}(\exp(-\frac{1}{3}T\gamma^2))$.

For the margin bound, by plugging $\mathbf{1}\{h_t(\tilde{\boldsymbol{x}}_j) = \tilde{y}_j\} = (\tilde{y}_j h_t(\tilde{\boldsymbol{x}}_j) + 1)/2$, we rewrite Eq. (2.4) as:

$$2\left(\gamma - \frac{R_T^{j/N}}{T}\right) \leq \frac{1}{T}\sum_{t=1}^{T} \tilde{y}_j h_t(\tilde{\boldsymbol{x}}_j).$$

Therefore, if j is such that $\theta < 2(\gamma - R_T^{j/N}/T)$, then the fraction of examples with margin at most θ is again at most $(j-1)/N$, which is of order $\tilde{O}(\exp(-\frac{1}{3}T(\theta - 2\gamma)^2))$. □

Thus, the training error decreases at roughly the same rate as AdaBoost. In addition, this theorem implies that the fraction of examples with margin smaller than 2γ eventually goes to zero as T gets large, which means NH-Boost.DT converges to the optimal margin 2γ; this is known not to be true for AdaBoost (see [98]). Also, like AdaBoost, NH-Boost.DT is an adaptive boosting algorithm that does not require γ or T as a parameter. However, unlike AdaBoost, NH-Boost.DT has the striking property that it completely ignores many examples on each round (by assigning zero weight), which is very helpful for the weak learning algorithm in terms of computational efficiency.

2.2.1 Experiments

We conducted experiments to compare the performance of three boosting algorithms for binary classification: AdaBoost [38], NH-Boost (Algorithm 5) and NH-Boost.DT (Algorithm 4), using a set of benchmark data available from the UCI repository[2] and LIBSVM datasets[3]. Some datasets are preprocessed according to [92]. The number of features, training examples and test examples can be found in Appendix D.

All features are binary. The weak learning algorithm is a simple (exhaustive) decision stump (see for instance [98, Chap 1.2]). On each round, the weak learning algorithm enumerates all features, and for each feature computes the weighted error of

[2] `http://archive.ics.uci.edu/ml/`

[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

the corresponding stump on the weighted training examples. Therefore, if the number of examples with zero weight is relatively large, then the weak learning algorithm would be faster in computing the weighted error and thus faster in finding the best feature.

All boosting algorithms are run for two hundred rounds. The results are summarized in Table 2.2, with bold entries being the best ones among the three (AB, NB and NBDT stand for AdaBoost, NH-Boost and NH-Boost.DT respectively). As we can see, in terms of training error and test error, all three algorithms have similar performance. However, our NH-Boost.DT algorithm is always the fastest one. The average fraction of examples with zero weights for NH-Boost.DT is significantly higher than the one for NH-Boost (note that AdaBoost does not assign zero weight at all). We plot the change of this fraction over rounds in Figure 2.2 (using three datasets). As both algorithms proceed, they tend to ignore more and more examples on each round, but NH-Boost.DT consistently ignores more examples than NH-Boost.

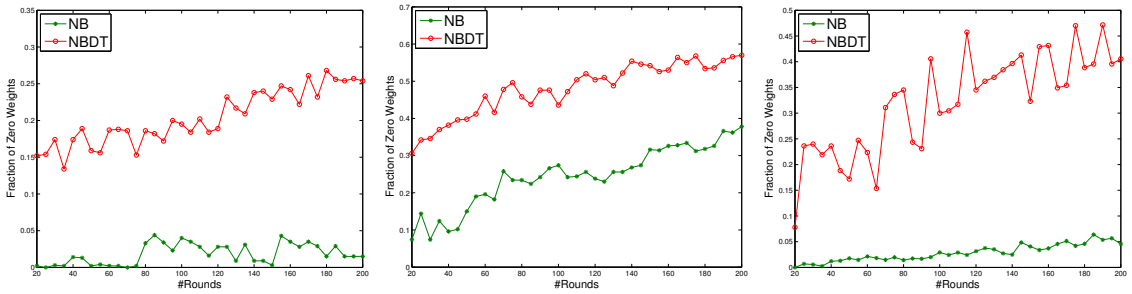
Since $s_{t,i}$ is positively correlated to the margin of example i ($\frac{1}{t}\sum_{\tau=1}^{t} y_i h_\tau(\boldsymbol{x}_i)$) and large $s_{t,i}$ leads to zero weight, the above phenomenon in fact implies that the examples' margins should be larger for NH-Boost.DT than for NH-Boost. This is confirmed by Figure 2.3, where we plot the final cumulative margins on three datasets (i.e. each point represents the fraction of examples with at most some fixed margin). One can see that the lines for NH-Boost.DT are below the ones for NH-Boost (and even AdaBoost) most of the time, meaning that NH-Boost.DT achieves larger margins in general. This could explain NH-Boost.DT's better test error on some datasets.

2.3 An Improved Variant: AdaNormalHedge

The regret for NormalHedge.DT is of order $O(\sqrt{T})$, which is always increasing in T and thus too conservative. Ideally we want the algorithm to achieve much smaller

Table 2.2: Training and test error for different boosting algorithms

	Time (s)			Zeros (%)		Training Error (%)			Test Error (%)		
Data	AB	NB	NBDT	NB	NBDT	AB	NB	NBDT	AB	NB	NBDT
a9a	57.5	72.5	**46.2**	1.1	**22.1**	**15.4**	15.8	15.5	**15.0**	15.6	15.2
census	1.7	2.2	**1.4**	2.2	**19.2**	15.6	17.0	**15.4**	18.7	18.6	**18.3**
ocr49	5.1	4.7	**3.0**	17.1	**42.0**	**1.7**	**1.7**	2.4	**5.5**	5.9	5.8
splice	1.6	1.5	**0.9**	22.2	**45.1**	**0.0**	**0.0**	0.4	9.4	8.6	**8.2**
w8a	237.6	244.7	**170.7**	3.0	**29.3**	2.6	**2.2**	2.4	2.7	**2.3**	2.6

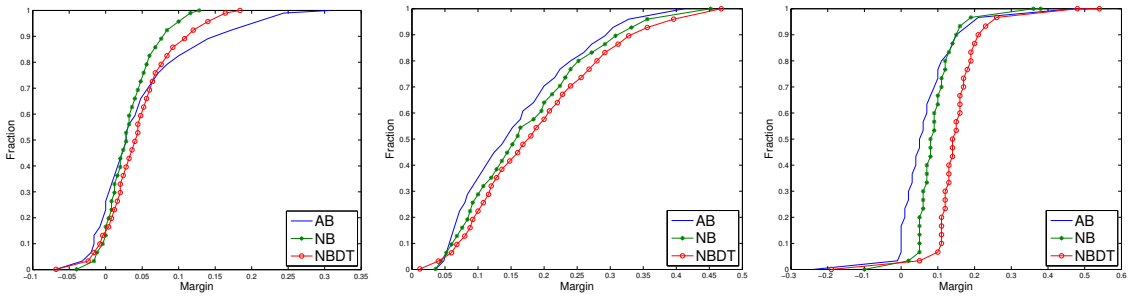


(a) census (b) splice (c) w8a

Figure 2.2: Comparisons of fraction of zero weights between NH-Boost and NH-Boost.DT

regret when the problem is "easy" while still ensuring worst-case robustness, which indeed has been studied in several previous works [32, 104, 42]. Besides this goal, various difficult objectives requiring more adaptive algorithms were also studied in recent years, such as: competing with a sequence of different combinations of the experts [60, 25] and learning with experts who provide confidence-rated advice [16]. However, in most cases these objectives are achieved by different algorithms separately. It is very desirable to have one single algorithm that can achieve all these goals simultaneously.

It turns out that a simple refinement of our NormalHedge.DT algorithm can indeed lead to such a highly adaptive algorithm achieving all the goals, with even improved results in some cases compared to previous work. To describe the algorithm, we first introduce some more notation to ease the presentation. Denote the learner's loss



(a) census (b) splice (c) w8a

Figure 2.3: Comparisons of cumulative margins between AdaBoost, NH-Boost and NH-Boost.DT

$\boldsymbol{p}_t \cdot \boldsymbol{\ell}_t$ by $\hat{\ell}_t$, the *instantaneous regret* to expert i on round t by $r_{t,i} = \hat{\ell}_t - \ell_{t,i}$, the cumulative regret by $R_{t,i} = \sum_{\tau=1}^{t} r_{\tau,i}$, and the cumulative loss by $L_{t,i} = \sum_{\tau=1}^{t} \ell_{\tau,i}$. We also define $\tilde{L}_{t,i} = \sum_{\tau=1}^{t} [\ell_{\tau,i} - \hat{\ell}_\tau]_+$. Many bounds discussed below will be in terms of $\tilde{L}_{T,i}$, which is always at most $L_{T,i}$ since trivially $[\ell_{t,i} - \hat{\ell}_t]_+ \leq \ell_{t,i}$. We will use bold letters to denote vectors with N corresponding coordinates. For example, $\boldsymbol{r}_t$, $\mathbf{R}_t$ and $\mathbf{L}_t$ represent $(r_{t,1}, \ldots, r_{t,N})$, $(R_{t,1}, \ldots, R_{t,N})$ and $(L_{t,1}, \ldots, L_{t,N})$ respectively.

We consider a more general regret definition where the learner wants to have performance close to an arbitrary convex combination of experts: $R_T(\boldsymbol{u}) = \sum_{t=1}^{T} \boldsymbol{u} \cdot \boldsymbol{r}_t = \sum_{t=1}^{T} \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t - \boldsymbol{u} \cdot \boldsymbol{\ell}_t$ where the competitor $\boldsymbol{u}$ is a fixed unknown distribution over the experts. Let $\epsilon \in (0, 1]$ and competitor $\boldsymbol{u}^*_\epsilon$ be a distribution that puts all the mass on the $\lceil N\epsilon \rceil$-th best expert. Then it is clear that $R_T(\boldsymbol{u}^*_\epsilon)$ corresponds to the previous notation R^ϵ_T. When it is clear from the context, we drop the subscript T in $R_T(\boldsymbol{u})$.

Now we rewrite NormalHedge.DT in the following way. Define potential function $\Phi(R, C) = \exp\left(\frac{[R]_+^2}{3C}\right)$ with $\Phi(0, 0)$ defined to be 1, and also a weight function with respect to this potential: $w(R, C) = \frac{1}{2}\left(\Phi(R + 1, C + 1) - \Phi(R - 1, C + 1)\right)$. Then the prediction of NormalHedge.DT is simply to set $p_{t,i}$ to be proportional to $w(R_{t-1,i}, C_{t-1})$ where $C_t = t$ for all t. Note that C_t is closely related to the regret. In fact, the regret is roughly of order $\sqrt{C_T}$ as shown in Section 2.1.4. Therefore, in order to get an expert-wise and more refined bound, we replace C_t by $C_{t,i}$ for each

Algorithm 6 AdaNormalHedge

Input: A prior distribution $\boldsymbol{q} \in \Delta_N$ over experts (uniform if no prior available).
Initialize: $\forall i \in [N], R_{0,i} = 0, C_{0,i} = 0$.
for $t = 1$ **to** T **do**
    Predict $p_{t,i} \propto q_i w(R_{t-1,i}, C_{t-1,i})$.
    Adversary reveals loss vector $\boldsymbol{\ell}_t$ and player suffers loss $\hat{\ell}_t = \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t$.
    Set $\forall i \in [N], r_{t,i} = \hat{\ell}_t - \ell_{t,i}, R_{t,i} = R_{t-1,i} + r_{t,i}, C_{t,i} = C_{t-1,i} + |r_{t,i}|$.
end for

expert so that it captures some useful information for each expert i. There are several possible choices for $C_{t,i}$, but for now we focus on the one used in our new algorithm: $C_{t,i} = \sum_{\tau=1}^{t} |r_{\tau,i}|$, that is, the cumulative magnitude of the instantaneous regrets up to time t. We call this algorithm AdaNormalHedge and summarize it in Algorithm 6. Note that we even allow the player to have a prior distribution $\boldsymbol{q}$ over the experts, which will be useful in some applications as we will see in Section 2.3.3. The theoretical guarantee of AdaNormalHedge is stated below.

Theorem 4. *The regret of AdaNormalHedge to any competitor $\boldsymbol{u} \in \Delta_N$ is bounded as follows:*

$$R(\boldsymbol{u}) \le \sqrt{3(\boldsymbol{u} \cdot \mathbf{C}_T)\left(\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q}) + \ln B + \ln(1 + \ln N)\right)} = \hat{O}(\sqrt{(\boldsymbol{u} \cdot \mathbf{C}_T)\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q})}), \tag{2.5}$$

where $\mathbf{C}_T = (C_{T,1}, \ldots, C_{T,N})$, $B = 1 + \frac{3}{2}\sum_i q_i \left(1 + \ln(1 + C_{T,i})\right) \le \frac{5}{2} + \frac{3}{2}\ln(1 + T)$. *Moreover, if $\boldsymbol{u}$ is a uniform distribution over a subset of $[N]$, then the regret can be improved to*

$$R(\boldsymbol{u}) \le \sqrt{3(\boldsymbol{u} \cdot \mathbf{C}_T)\left(\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q}) + \ln B + 1\right)}. \tag{2.6}$$

Before we prove this theorem, we discuss some implications of the regret bounds and why they are interesting. First of all, the relative entropy term $\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q})$ captures how close the player's prior is to the competitor. A bound in terms of $\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q})$ can be obtained, for example, using the classic exponential weights algorithm but re-

quires carefully tuning the learning rate as a function of $\boldsymbol{u}$. AdaNormalHedge achieves this bound without knowing $\boldsymbol{u}$.

On the other hand, if $\boldsymbol{q}$ is a uniform distribution, then using bound (2.6) and the fact $C_{T,i} \le T$, we get an ϵ-regret bound similar to the one of NormalHedge.DT: $R(\boldsymbol{u}_\epsilon^*) \le R(\boldsymbol{u}_{S_\epsilon}) \le \sqrt{3T\left(\ln\left(\frac{1}{\epsilon}\right) + \ln B + 1\right)}$ where $\boldsymbol{u}_{S_\epsilon}$ is uniform over the top $\lceil N\epsilon \rceil$ experts in terms of their total loss $L_{T,i}$.

However, the power of a bound in terms of $\mathbf{C}_T$ is far more than this. Gaillard et al. [42] introduced a new second order bound that implies much smaller regret when the problem is easy. It turns out that our seemingly weaker first order bound is also enough to get the exact same results! We state these implications in the following theorem which is essentially a restatement of Theorems 9 and 11 of Gaillard et al. [42] with weaker conditions.

Theorem 5. *Suppose an expert algorithm guarantees $R(\boldsymbol{u}) \le \sqrt{(\boldsymbol{u} \cdot \mathbf{C}_T)A(\boldsymbol{u})}$ where $A(\boldsymbol{u})$ is some function of $\boldsymbol{u}$. Then it also satisfies the following:*

1. *Recall $L_{T,i} = \sum_{t=1}^T \ell_{t,i}$ and $\tilde{L}_{T,i} = \sum_{t=1}^T [\ell_{t,i} - \hat{\ell}_t]_+$. We have*

$$R(\boldsymbol{u}) \le \sqrt{2(\boldsymbol{u} \cdot \tilde{\boldsymbol{L}}_T)A(\boldsymbol{u})} + A(\boldsymbol{u}) \le \sqrt{2(\boldsymbol{u} \cdot \mathbf{L}_T)A(\boldsymbol{u})} + A(\boldsymbol{u}).$$

2. *Suppose the loss vector $\boldsymbol{\ell}_t$ are independent random variables and there exists an i^* and some $\alpha \in (0,1]$ such that $\mathbb{E}[\ell_{t,i} - \ell_{t,i^*}] \ge \alpha$ for any t and $i \ne i^*$. Let $\boldsymbol{e}_{i^*}$ be a distribution that puts all the mass on expert i^*. Then we have $\mathbb{E}[R_{T,i^*}] \le \frac{A(\boldsymbol{e}_{i^*})}{\alpha}$, and with probability at least $1-\delta$, $R_{T,i^*} \le \hat{O}\left(\frac{A(\boldsymbol{e}_{i^*})}{\alpha} + \frac{1}{\alpha}\sqrt{A(\boldsymbol{e}_{i^*}) \ln \frac{1}{\delta}}\right)$.*

Proof. For the first result, the key observation is $\boldsymbol{u} \cdot \mathbf{C}_T = R(\boldsymbol{u}) + 2\boldsymbol{u} \cdot \tilde{\boldsymbol{L}}_T$. We only consider the case when $R(\boldsymbol{u}) \ge 0$ since otherwise the statement is trivial. By the condition of the theorem we thus have $R(\boldsymbol{u})^2 \le (R(\boldsymbol{u}) + 2\boldsymbol{u} \cdot \tilde{\boldsymbol{L}}_T)A(\boldsymbol{u})$, which by solving for $R(\boldsymbol{u})$ gives $R(\boldsymbol{u}) \le \frac{1}{2}(A(\boldsymbol{u}) + \sqrt{A(\boldsymbol{u})^2 + 8(\boldsymbol{u} \cdot \tilde{\boldsymbol{L}}_T)A(\boldsymbol{u})}) \le \sqrt{2(\boldsymbol{u} \cdot \tilde{\boldsymbol{L}}_T)A(\boldsymbol{u})} + A(\boldsymbol{u})$, proving the bound we want.

For the second result, let $\mathbb{E}_t$ denote the expectation conditioning on all the randomness up to round t. So by the condition, we have $\mathbb{E}_t[r_{t,i^*}] = \sum_{i=1}^N p_{t,i}\mathbb{E}_t[\ell_{t,i}-\ell_{t,i^*}] \geq \alpha(1-p_{t,i^*})$, and thus $\mathbb{E}[R_{T,i^*}] \geq \alpha S$ where we define $S = \sum_{t=1}^T \mathbb{E}[1-p_{t,i^*}]$. On the other hand, by convexity we also have $|r_{t,i^*}| \leq \sum_{i=1}^N p_{t,i}|\ell_{t,i}-\ell_{t,i^*}| \leq 1-p_{t,i^*}$ and thus $\mathbb{E}[R_{T,i^*}] \leq \mathbb{E}[\sqrt{A(\boldsymbol{e}_{i^*})C_{T,i^*}}] \leq \sqrt{A(\boldsymbol{e}_{i^*})S}$ by the concavity of the square root function. Combining the above two statements gives $S \leq \frac{A(\boldsymbol{e}_{i^*})}{\alpha^2}$, and plugging this back shows $\mathbb{E}[R_{T,i^*}] \leq \frac{A(\boldsymbol{e}_{i^*})}{\alpha}$. The high probability statement follows from the exact same argument of Gaillard et al. [42] using a martingale concentration lemma. □

For AdaNormalHedge, the term $A(\boldsymbol{u})$ is $3(\mathrm{RE}(\boldsymbol{u} \mid\mid \boldsymbol{q}) + \ln B + \ln(1+\ln N))$ in general (or smaller for special $\boldsymbol{u}$ as stated in Theorem 4). Applying Theorem 5 we have $R(\boldsymbol{u}) = \hat{O}\Big(\sqrt{(\boldsymbol{u}\cdot\tilde{L}_T)\mathrm{RE}(\boldsymbol{u} \mid\mid \boldsymbol{q})}\Big)$.[4] Specifically, if $\boldsymbol{q}$ is uniform and assuming without loss of generality that $L_{T,1} \leq \cdots \leq L_{T,N}$, then by a similar argument, we have for AdaNormalHedge, $R(\boldsymbol{u}_\epsilon^*) \leq \hat{O}\Big(\sqrt{\tilde{L}_{T,\lceil N\epsilon\rceil}\ln\left(\frac{1}{\epsilon}\right)}\Big)$ for any ϵ. This answers the open question (in the affirmative) asked by Chaudhuri et al. [26] and Chernov and Vovk [29] on whether an improvement for small loss can be obtained for ϵ-regret without knowing ϵ.

On the other hand, when we are in a stochastic setting as stated in Theorem 5, AdaNormalHedge ensures $R_{T,i^*} \leq \hat{O}\left(\frac{1}{\alpha}\ln\left(\frac{1}{q_{i^*}}\right)\right)$ in expectation (or with high probability with an extra confidence term), which does not grow with T. Therefore, the new regret bound in terms of $\mathbf{C}_T$ actually leads to significant improvements compared to NormalHedge.DT.

Comparison to Adapt-ML-Prod [42]. Adapt-ML-Prod enjoys a second order bound in terms of $\sum_{t=1}^T r_{t,i}^2$, which is always at most the term $\sum_{t=1}^T |r_{t,i}|$ that appears in our bounds. On the one hand, as discussed above, these two bounds have the

[4] We see $O(\mathrm{RE}(\boldsymbol{u} \mid\mid \boldsymbol{q}))$ as a minor term and hide it in the big O notation, which is not completely rigorous but will ease the presentation. We treat other first-order bounds in the same way in this work.

same improvements when the problem is easy in several senses; on the other hand, Adapt-ML-Prod does not provide a bound in terms of RE($\boldsymbol{u}$ || $\boldsymbol{q}$) for an unknown $\boldsymbol{u}$. In fact, as discussed at the end of Section A.3 of Gaillard et al. [42], Adapt-ML-Prod cannot improve by exploiting a good prior $\boldsymbol{q}$ (or at least its current analysis cannot). Specifically, while the regret for AdaNormalHedge does not have an explicit dependence on N and is much smaller when the prior $\boldsymbol{q}$ is close to the competitor $\boldsymbol{u}$, the regret for Adapt-ML-Prod always has a $\ln N$ multiplicative term for $\sum_{t=1}^{T} r_{t,i}^2$, which means even a good prior results in the same regret as a uniform prior! More advantages of AdaNormalHedge over Adapt-ML-Prod will be discussed in concrete examples in the following sections.

2.3.1 Analysis of AdaNormalHedge

We prove Theorem 4 in this section. We need the following two lemmas. The first one is an improved version of Lemma 2.

Lemma 3. *For any $R \in \mathbb{R}, C \geq 0$ and $r \in [-1, 1]$, we have*

$$\Phi(R + r, C + |r|) \leq \Phi(R, C) + w(R, C)r + \frac{3|r|}{2(C + 1)}.$$

Proof. We first argue that $\Phi(R+r, C+|r|)$, as a function of r, is piecewise-convex on $[-1, 0]$ and $[0, 1]$. Since the value of the function is 1 when $R+r < 0$ and is at least 1 otherwise, it suffices to only consider the case when $R + r \geq 0$. On the interval $[0, 1]$, we can rewrite the exponent (ignoring the constant $\frac{1}{3}$) as:

$$\frac{(R + r)^2}{C + r} = (C + r) + \frac{(R - C)^2}{C + r} + 2(R - C),$$

which is convex in r. Combining with the fact that "if $g(x)$ is convex then $\exp(g(x))$ is also convex" proves that $\Phi(R + r, C + |r|)$ is convex on $[0, 1]$. Similarly when

$r \in [-1, 0]$, rewriting the exponent as

$$\frac{(R+r)^2}{C-r} = (C-r) + \frac{(R+C)^2}{C-r} - 2(R+C)$$

completes the argument.

Now define function $f(r) = \Phi(R+r, C+|r|) - w(R,C)r$. Since $f(r)$ is clearly also piecewise-convex on $[-1, 0]$ and $[0, 1]$, we know that the curve of $f(r)$ is below the segment connecting points $(-1, f(-1))$ and $(0, f(0))$ on $[-1, 0]$, and also below the segment connecting points $(1, f(1))$ and $(0, f(0))$ on $[0, 1]$. This can be mathematically expressed as:

$$f(r) \leq \max\{f(0) + (f(0) - f(-1))r, f(0) + (f(1) - f(0))r\} = f(0) + (f(1) - f(0))|r|,$$

where we use the fact $f(-1) = f(1)$. Now by Lemma 2, we have

$$\begin{aligned} f(1) - f(0) &= \tfrac{1}{2}\left(\Phi(R+1, C+1) + \Phi(R-1, C+1)\right) - \Phi(R, C) \\ &\leq \tfrac{1}{2}\left(\exp\left(\tfrac{4}{3(C+1)}\right) - 1\right), \end{aligned}$$

which is at most $\frac{e^{4/3}-1}{2(C+1)}$ since C is nonnegative and $e^x - 1 \leq \frac{e^a-1}{a}x$ for any $x \in [0, a]$. Noting that $e^{4/3} - 1 \leq 3$ completes the proof. □

The second lemma makes use of Lemma 3 to show that the weighted sum of potentials does not increase much and thus the final potential is relatively small.

Lemma 4. *AdaNormalHedge ensures*

$$\sum_{i=1}^{N} q_i \Phi(R_{T,i}, C_{T,i}) \leq B = 1 + \frac{3}{2} \sum_{i=1}^{N} q_i \left(1 + \ln(1 + C_{T,i})\right).$$

Proof. First note that since AdaNormalHedge predicts $p_{t,i} \propto q_i w(R_{t-1,i}, C_{t-1,i})$, we have

$$\sum_{i=1}^N q_i w(R_{t-1,i}, C_{t-1,i}) r_{t,i} = 0. \tag{2.7}$$

Now applying Lemma 3 with $R = R_{t-1,i}, C = C_{t-1,i}$ and $r = r_{t,i}$, multiplying the inequality by q_i on both sides and summing over i gives $\sum_{i=1}^N q_i \Phi(R_{t,i}, C_{t,i}) \le \sum_{i=1}^N q_i \Phi(R_{t-1,i}, C_{t-1,i}) + \frac{3}{2} \sum_{i=1}^N \frac{q_i |r_{t,i}|}{C_{t-1,i}+1}$. We then sum over $t \in [T]$ and telescope to show $\sum_{i=1}^N q_i \Phi(R_{T,i}, C_{T,i}) \le 1 + \frac{3}{2} \sum_{i=1}^N q_i \sum_{t=1}^T \frac{|r_{t,i}|}{C_{t-1,i}+1}$. Finally applying Lemma 14 of [42] to show $\sum_{t=1}^T \frac{|r_{t,i}|}{C_{t-1,i}+1} \le 1 + \ln(1 + C_{T,i})$ completes the proof. □

We are now ready to prove Theorem 4.

Proof of Theorem 4. Assume $q_1 \Phi(R_{T,1}, C_{T,1}) \ge \cdots \ge q_N \Phi(R_{T,N}, C_{T,N})$ without loss of generality. Then by Lemma 4, it must be true that $q_i \Phi(R_{T,i}, C_{T,i}) \le \frac{B}{i}$ for all i, which, by solving for $R_{T,i}$, gives $R_{T,i} \le \sqrt{3 C_{T,i} \ln\left(\frac{B}{i q_i}\right)}$. Multiplying both sides by u_i, summing over N and applying the Cauchy-Schwarz inequality, we arrive at $R(\boldsymbol{u}) \le \sum_{i=1}^N \sqrt{3 u_i C_{T,i} \cdot u_i \ln\left(\frac{B}{i q_i}\right)} \le \sqrt{3(\boldsymbol{u} \cdot \mathbf{C}_T)(D(\boldsymbol{u} \,||\, \boldsymbol{q}) + \ln B)}$, where we define $D(\boldsymbol{u} \,||\, \boldsymbol{q}) = \sum_{i=1}^N u_i \ln\left(\frac{1}{i q_i}\right)$. It remains to show that $D(\boldsymbol{u} \,||\, \boldsymbol{q})$ and $\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q})$ are close. Indeed, we have $D(\boldsymbol{u} \,||\, \boldsymbol{q}) - \mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q}) = \sum_{i=1}^N u_i \ln\left(\frac{1}{i u_i}\right)$, which, by standard analysis, can be shown to reach its maximum when $u_i \propto \frac{1}{i}$ and the maximum value is $\ln \sum_i \frac{1}{i} \le \ln(1 + \ln N)$. This completes the proof for Eq. (2.5).

Finally, when $\boldsymbol{u}$ is in the special form as described in Theorem 4, we have $D(\boldsymbol{u} \,||\, \boldsymbol{q}) - \mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q}) = \ln |S| + \frac{1}{|S|} \sum_{i \in S} \ln\left(\frac{1}{i}\right) \le \ln |S| - \frac{1}{|S|} \ln(|S|!)$. By Stirling's formula $x! \ge \sqrt{2\pi x} \left(\frac{x}{e}\right)^x$, we arrive at $D(\boldsymbol{u} \,||\, \boldsymbol{q}) - \mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q}) \le 1 - (\ln \sqrt{2\pi |S|})/|S| \le 1$, proving Eq. (2.6). □

The algorithm and the proof can be generalized to $C_{t,i} = \sum_{\tau=1}^t |r_{\tau,i}|^d$ for any $d \in [0, 1]$. Indeed, the only extra work is to prove the convexity of $\Phi(R + r, C + |r|^d)$. When $d = 0$, we recover NormalHedge.DT exactly and get a bound on $R(\boldsymbol{u})$ for any $\boldsymbol{u}$

(in terms of $\sqrt{T}$), instead of just $R(\boldsymbol{u}^*_\epsilon)$ as in Section 2.1.4. It is clear that $d = 1$ gives the smallest bound, which is why we use it in AdaNormalHedge. The ideal choice, however, should be $d = 2$ so that a second order bound similar to the one of Gaillard et al. [42] can be obtained. Unfortunately, the function $\Phi(R + r, C + r^2)$ turns out to not always be piecewise-convex, which breaks our analysis. Whether $d = 2$ gives a low-regret algorithm and how to analyze it remain an open question.

2.3.2 Confidence-rated Advice and Sleeping Experts

We further generalize AdaNormalHedge to deal with experts that make confidence-rated advice, a setting that subsumes many interesting applications as studied by Blum [15] and Freund et al. [40]. In this general setting, on each round t, each expert first reports its *confidence* $\mathcal{I}_{t,i} \in [0, 1]$ for the current task. The player then predicts $\boldsymbol{p}_t$ as usual with an extra yet natural restriction that if $\mathcal{I}_{t,i} = 0$ then $p_{t,i} = 0$. That is, the player has to ignore those experts who abstain from making advice (by reporting zero confidence). After that, the loss $\ell_{t,i}$ for those experts who did not abstain (i.e. $\mathcal{I}_{t,i} \neq 0$) are revealed and the player still suffers loss $\hat{\ell}_t = \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t$. We redefine the instantaneous regret $r_{t,i}$ to be $\mathcal{I}_{t,i}(\hat{\ell}_t - \ell_{t,i})$, that is, the difference between the loss of the player and expert i weighted by the confidence. The goal of the player is, as before, to minimize cumulative regret to any competitor $\boldsymbol{u}$: $R(\boldsymbol{u}) = \sum_{t \le T} \boldsymbol{u} \cdot \boldsymbol{r}_t$. Clearly, the classic expert problem that we have studied in previous sections is just a special case of this general setting with $\mathcal{I}_{t,i} = 1$ for all t and i.

Moreover, with this general form of $r_{t,i}$, AdaNormalHedge can be used to deal with this general setting with only one simple change of scaling the weights by the confidence:

$$p_{t,i} \propto q_i \mathcal{I}_{t,i} w(R_{t-1,i}, C_{t-1,i}), \tag{2.8}$$

where $R_{t,i}$ and $C_{t,i}$ are still defined to be $\sum_{\tau=1}^{t} r_{\tau,i}$ and $\sum_{\tau=1}^{t} |r_{\tau,i}|$ respectively. The constraint $\mathcal{I}_{t,i} = 0 \Rightarrow p_{t,i} = 0$ is clearly satisfied. In fact, Algorithm 6 can be seen as a special case of this general form of AdaNormalHedge with $I_{t,i} \equiv 1$. Furthermore, the regret bounds in Theorem 4 still hold without any changes, which are summarized below.

Theorem 6. *For the confidence-rated expert problem, regret bounds* (2.5) *and* (2.6) *still hold for general AdaNormalHedge (Eq.* (2.8)*).*

Proof. It suffices to point out that $r_{t,i}$ is still in the interval $[-1, 1]$ and Eq. (2.7) in the proof of Lemma 4 still holds by the new prediction rule Eq. (2.8). The entire proof for Theorem 4 applies here exactly. □

Previously, Gaillard et al. [42] studied a general reduction from an expert algorithm to a confidence-rated expert algorithm. Applying those results here gives the exact same algorithm and regret guarantee mentioned above. However, we point out that the general reduction is not always applicable. Specifically, it is invalid if there is an unknown number of experts in the confidence-rated setting (explained more in the next paragraph) while the expert algorithm in the standard setting requires knowing the number of experts as a parameter. This is indeed the case for most algorithms (including Adapt-ML-Prod and even the original NormalHedge by Chaudhuri et al. [26]). AdaNormalHedge naturally avoids this problem since it does not depend on N at all.

Sleeping Experts. We are especially interested in the case when $\mathcal{I}_{t,i} \in \{0, 1\}$, also called the specialist/sleeping expert problem where $\mathcal{I}_{t,i} = 0$ means that expert i is "asleep" for round t and not giving any advice. This is a natural setting where the total number of experts is unknown ahead of time. Indeed, the number of awake experts can be dynamically changing over time. An expert that has never appeared before should be thought of as being asleep for all previous rounds.

AdaNormalHedge is a very suitable algorithm to deal with this case due to its independence of the total number of experts. If an expert i appears for the first time on round t, then by definition it will naturally start with $R_{t-1,i} = 0$ and $C_{t-1,i} = 0$. Although we state the prior q as a distribution, which seems to require knowing the total number of experts, it is not an issue algorithmically since $\boldsymbol{q}$ is only used to scale the unnormalized weights (Eq. (2.8)). For example, if we want $\boldsymbol{q}$ to be a uniform distribution over N experts where N is unknown beforehand, then to run AdaNormalHedge we can simply treat q_i in Eq. (2.8) to be 1 for all i, which clearly will not change the behavior of the algorithm anyway. In this case, if we let N_t denote the total number of distinct experts that have been seen up to time t and the competitor $\boldsymbol{u}$ concentrates on any of these experts, then the relative entropy term in the regret (up to time t) will be $\ln N_t$ (instead of $\ln N$), which is changing over time.

Using the adaptivity of AdaNormalHedge in both the number of experts and the competitor, we provide improved results for two instances of the sleeping expert problem in the next two sections.

2.3.3 Time-Varying Competitors

In this section, we study a more challenging goal of competing with time-varying competitors in the standard expert setting (that is, each expert is always awake and again $r_{t,i} = \hat{\ell}_t - \ell_{t,i}$), which turns out to be reducible to a sleeping expert problem. Results for this section are summarized in Table 2.3.

Special Cases: Adaptive Regret and Tracking the Best Expert

We start from a special case: adaptive regret, introduced by Hazan and Seshadhri [56] to better capture changing environments. Formally, consider any time interval $t = t_1, \ldots, t_2$, and let $R_{[t_1,t_2],i} = \sum_{t=t_1}^{t_2} r_{t,i}$ be the regret to expert i on this interval (similarly define $L_{[t_1,t_2],i} = \sum_{t=t_1}^{t_2} \ell_{t,i}$ and $C_{[t_1,t_2],i} = \sum_{t=t_1}^{t_2} |r_{t,i}|$). The goal of the

player is to obtain relatively small regret on any interval. Freund et al. [40] essentially introduced a way to reduce this problem to a sleeping expert problem, which was later improved by Adamskiy et al. [4]. Specifically, for every pair of time t and expert i, we create a sleeping expert, denoted by (t, i), who is only awake after (and including) round t and after that point suffers the same loss as the original expert i. So we have Nt sleeping experts in total on round t. The prediction $p_{t,i}$ is set to be the sum of all the weights of sleeping expert (τ, i) $(\tau = 1, \dots, t)$. It is clear that doing this ensures that the cumulative regret up to time t_2 with respect to sleeping expert (t_1, i) is exactly $R_{[t_1,t_2],i}$ in the original problem.

This is a sleeping expert problem for which AdaNormalHedge is very suitable, since the number of sleeping experts keeps increasing and the total number of experts is in fact unknown if the horizon T is unknown. Theorem 6 implies that the resulting algorithm gives the following adaptive regret:

$$R_{[t_1,t_2],i} = \hat{O}\left(\sqrt{\left(\sum_{t=t_1}^{t_2} |r_{t,i}|\right) \ln\left(\frac{1}{q_{(t_1,i)}}\right)}\right) = \hat{O}\left(\sqrt{\left(\sum_{t=t_1}^{t_2} |r_{t,i}|\right) \ln\left(Nt_1\right)}\right),$$

where $\boldsymbol{q}$ is a prior over the Nt_2 experts and the last step is by setting the prior to be $q_{(t,i)} \propto 1/t^2$ for all t and i.[5] This prior is better than a simple uniform distribution which leads to a term $\ln(Nt_2)$ instead of $\ln(Nt_1)$. We call this algorithm AdaNormalHedge.TV.[6] To be concrete, on round t AdaNormalHedge.TV predicts $p_{t,i} \propto \sum_{\tau=1}^{t} \frac{1}{\tau^2} w\left(R_{[\tau,t-1],i}, C_{[\tau,t-1],i}\right)$.

Again, Theorem 5 can be applied to get the bound $\hat{O}\left(\sqrt{\tilde{L}_{[t_1,t_2],i} \ln\left(Nt_1\right)}\right)$ where $\tilde{L}_{[t_1,t_2],i} = \sum_{t=t_1}^{t_2} [\ell_{t,i} - \hat{\ell}_t]_+ \le L_{[t_1,t_2],i}$, and a much smaller bound $\hat{O}\left(\frac{\ln(Nt_1)}{\alpha}\right)$ if the losses are stochastic on interval $[t_1, t_2]$ in the sense stated in Theorem 5.

One drawback of AdaNormalHedge.TV is that its time complexity per round is $O(Nt)$ and the overall space is $O(NT)$. However, the data streaming technique used

[5] Note that as discussed before, the fact that t_2 is unknown and thus $\boldsymbol{q}$ is unknown does not affect the algorithm.

[6] "TV" stands for "time-varying".

in Hazan and Seshadhri [56] can be directly applied here to reduce the time and space complexity to $O(N \ln t)$ and $O(N \ln T)$ respectively, with only an extra multiplicative $O(\sqrt{\ln(t_2 - t_1)})$ factor in the regret.

Tracking the best expert. In fact, AdaNormalHedge.TV is a solution for one of the open problems proposed by Warmuth and Koolen [106]. Adaptive regret immediately implies the so-called K-shifting regret for the problem of tracking the best expert in a changing environment. Formally, define the K-shifting regret $R_{K\text{-Shift}}$ to be $\max \sum_{k=1}^{K} R_{[t_{k-1}+1,t_k],i_k}$ where the max is taken over all $i_1, \cdots, i_K \in [N]$ and $0 = t_0 < t_1 < \cdots < t_K = T$. In other words, the player is competing with the best K-partition of the whole game and the best expert on each of these partitions. Let $L^*_{K\text{-Shift}} = \min \sum_{k=1}^{K} L_{[t_{k-1}+1,t_k],i_k}$ be the total loss of such best partition (that is, the min is taken over the same space), and similarly define $\tilde{L}^*_{K\text{-Shift}} = \min \sum_{k=1}^{K} \tilde{L}_{[t_{k-1}+1,t_k],i_k} \leq L^*_{K\text{-Shift}}$. Since essentially $R_{K\text{-Shift}}$ is just the sum of K adaptive regrets, using the bounds discussed above and the Cauchy-Schwarz inequality, we conclude that AdaNormalHedge.TV ensures $R_{K\text{-Shift}} = \hat{O}\Big(\sqrt{K \tilde{L}^*_{K\text{-Shift}} \ln(NT)}\Big)$. Also, if the loss vectors are generated randomly on these K intervals, each satisfying the condition stated in Theorem 5, then the regret is $R_{K\text{-Shift}} = \hat{O}\left(\frac{K \ln(NT)}{\alpha}\right)$ in expectation (high probability bound is similar). These bounds are optimal up to logarithmic factors [56]. This is exactly what was asked in Warmuth and Koolen [106]: whether there is an algorithm that can do optimally for both adversarial and stochastic losses in the problem of tracking the best expert. AdaNormalHedge.TV achieves this goal without knowing K or any other information, while the solution provided by Sani et al. [95] needs to know K , $L^*_{K\text{-Shift}}$ and α to get the same adversarial bound and a worse stochastic bound of order $O(1/\alpha^2)$.

Comparison to previous work. For adaptive regret, the FLH algorithm by Hazan and Seshadhri [56] treats any standard expert algorithm as a sleeping expert, and has

Table 2.3: Comparisons of different algorithms for time-varying competitors

Algorithm	$R_{[t_1,t_2],i}$	$R_{K\text{-Shift}}$	$R(\boldsymbol{u}_{1:T})$
AdaNormalHedge.TV (this work)	$\sqrt{\tilde{L}_{[t_1,t_2],i} \ln (Nt_1)}$	$\sqrt{K\tilde{L}^*_{K\text{-Shift}} \ln(NT)}$	$\sqrt{V(\boldsymbol{u}_{1:T})\tilde{L}(\boldsymbol{u}_{1:T}) \ln (NT)}$
FLH [56]	$\sqrt{t_2} \ln t_2$	$K\sqrt{T} \ln T$	unknown
Fixed Share [25]	$\sqrt{t_2 \ln(Nt_2)}$	$K\sqrt{T \ln(NT)}$	unknown

an additive term $O(\sqrt{t_2} \ln t_2)$ in addition to the base algorithm's regret (when no prior information is available), which adds up to a large $O(K\sqrt{T} \ln T)$ term for K-shifting regret. Due to this extra additive regret, FLH also does not enjoy first order bounds nor small regret in the stochastic setting, even if the base algorithm that it builds on provides these guarantees. On the other hand, FLH was proposed to achieve adaptive regret for any general online convex optimization problem. We point out that using AdaNormalHedge as the master algorithm in their framework will give similar improvements as discussed here.

Adapt-ML-Prod is not directly applicable here for the corresponding sleeping expert problem since the total number of experts is unknown.

Another well-studied algorithm for this problem is "fixed share". Several works on fixed share for the simpler "log loss" setting were studied before [59, 18, 4, 67]. Cesa-Bianchi et al. [25] studied a generalized fixed share algorithm for the bounded loss setting considered here. When $t_2 - t_1$ and $L_{[t_1,t_2],i}$ are known, their algorithm ensures $R_{[t_1,t_2],i} = O\left(\sqrt{L_{[t_1,t_2],i} \ln(N(t_2 - t_1))}\right)$ for adaptive regret, and when K, T and $L^*_{K\text{-Shift}}$ are known, they have $R_{K\text{-Shift}} = O\left(\sqrt{KL^*_{K\text{-Shift}} \ln(NT/K)}\right)$. No better result is provided for the stochastic setting. More importantly, when no prior information is known, which is the case in practice, the best results one can extract from their analysis are $R_{[t_1,t_2],i} = O(\sqrt{t_2 \ln(Nt_2)})$ and $R_{K\text{-Shift}} = O(K\sqrt{T \ln(NT)})$, which are much worse than our results.

General Time-Varying Competitors

We finally discuss the most general goal: compete with different $\boldsymbol{u}_t$ on different rounds. Recall $R(\boldsymbol{u}_{1:T}) = \sum_{t=1}^T \boldsymbol{u}_t \cdot \boldsymbol{r}_t$ where $\boldsymbol{u}_t \in \mathbb{R}_+^N$ for all t (note that $\boldsymbol{u}_t$ does not even have to be a distribution). Clearly, adaptive regret and K-shifting regret are special cases of this general notion. Intuitively, how large this regret is should be closely related to how much the competitor's sequence $\boldsymbol{u}_{1:T}$ varies. Cesa-Bianchi et al. [25] introduced a distance measurement to capture this variation: $V(\boldsymbol{u}_{1:T}) = \sum_{t=1}^T \sum_{i=1}^N [u_{t,i} - u_{t-1,i}]_+$ where we define $u_{0,i} = 0$ for all i. Also let $\|\boldsymbol{u}_{1:T}\| = \sum_{t=1}^T \|\boldsymbol{u}_t\|_1$ and $L(\boldsymbol{u}_{1:T}) = \sum_{t=1}^T \boldsymbol{u}_t \cdot \boldsymbol{\ell}_t$. Fixed share is shown to ensure the following regret [25]: $R(\boldsymbol{u}_{1:T}) = O\big(\sqrt{V(\boldsymbol{u}_{1:T}) L(\boldsymbol{u}_{1:T}) \ln\left(N\|\boldsymbol{u}_{1:T}\|/V(\boldsymbol{u}_{1:T})\right)}\big)$ when $V(\boldsymbol{u}_{1:T}), L(\boldsymbol{u}_{1:T})$ and $\|\boldsymbol{u}_{1:T}\|$ are known. No result was provided otherwise.[7] Here, we show that our parameter-free algorithm AdaNormalHedge.TV actually achieves almost the same bound without knowing any information beforehand. Moreover, while the results in Cesa-Bianchi et al. [25] are specific for the fixed share algorithm, we prove the following results which are independent of the concrete algorithms and may be of independent interest.

Theorem 7. *Suppose an expert algorithm ensures $R_{[t_1,t_2],i} \le \sqrt{A \sum_{t=t_1}^{t_2} z_{t,i}}$ for any t_1, t_2 and i, where $z_{t,i} \ge 0$ can be anything depending on t and i (e.g. $|r_{t,i}|$, $[\ell_{t,i} - \hat{\ell}_t]_+$, $\ell_{t,i}$ or constant 1), and A is a term independent of t_1, t_2 and i. Then this algorithm also ensures*

$$R(\boldsymbol{u}_{1:T}) \le \sqrt{AV(\boldsymbol{u}_{1:T}) \sum_{t=1}^T \boldsymbol{u}_t \cdot \boldsymbol{z}_t}.$$

Specifically, for AdaNormalHedge.TV, plugging $A = \hat{O}(\ln(NT))$ and $z_{t,i} = [\ell_{t,i} - \hat{\ell}_t]_+$ gives $R(\boldsymbol{u}_{1:T}) = \hat{O}\Big(\sqrt{V(\boldsymbol{u}_{1:T}) \tilde{L}(\boldsymbol{u}_{1:T}) \ln(NT)}\Big)$, where $\tilde{L}(\boldsymbol{u}_{1:T}) = \sum_{t=1}^T \sum_{i=1}^N u_{t,i}[\ell_{t,i} - \hat{\ell}_t]_+$.

[7] Although in Section 7.3 of Cesa-Bianchi et al. [25], the authors mentioned online tuning technique for the parameters, it only works for special cases (e.g. adaptive regret).

Proof. We use $[s,t]$ to denote the set $\{s, s+1, \ldots, t-1, t\}$ for $1 \le s \le t \le T$ and $s, t \in \mathbb{N}^+$. First fix an expert i and consider the regret to this expert $\sum_{t=1}^{T} u_{t,i} r_{t,i}$. Let $a_j \ge 0$ and $U_j = [s_j, t_j]$ for $j = 1, \ldots, M$ be M positive numbers and M corresponding time intervals such that $u_{t,i} = \sum_{j=1}^{M} a_j \mathbf{1}\{t \in U_j\}$. Note that this is always possible, with a trivial choice being $a_j = u_{t,j}$, $s_j = t_j = j$ and $M = T$; we will however need a more sophisticated construction specified later. By the adaptive regret guarantee, we have

$$\sum_{t=1}^{T} u_{t,i} r_{t,i} = \sum_{j=1}^{M} a_j \sum_{t=1}^{T} \mathbf{1}\{t \in U_j\} r_{t,i} = \sum_{j=1}^{M} a_j R_{[s_j,t_j],i} \le \sum_{j=1}^{M} a_j \sqrt{A \sum_{t=s_j}^{t_j} z_{t,i}},$$

which, by the Cauchy-Schwarz inequality, is at most

$$\begin{aligned}\sqrt{\sum_{j=1}^{M} a_j} \cdot \sqrt{A \sum_{j=1}^{M} a_j \sum_{t=s_j}^{t_j} z_{t,i}} &= \sqrt{\sum_{j=1}^{M} a_j} \cdot \sqrt{A \sum_{j=1}^{M} a_j \sum_{t=1}^{T} \mathbf{1}\{t \in U_j\} z_{t,i}} \\ &= \sqrt{\sum_{j=1}^{M} a_j} \cdot \sqrt{A \sum_{t=1}^{T} u_{t,i} z_{t,i}}.\end{aligned}$$

Therefore, we need a construction of a_j and U_j such that $\sum_{j=1}^{M} a_j$ is minimized. This is addressed in Lemma 5 below which shows that there is in fact always an (optimal) construction such that $\sum_{j=1}^{M} a_j$ is exactly $\sum_{t=1}^{T} [u_{t,i} - u_{t-1,i}]_+$. Now summing the resulting bound over all experts and applying the Cauchy-Schwarz inequality again proves the theorem. □

Lemma 5. *Let $v_1, \ldots, v_T$ be T nonnegative numbers and $h(\{v_1, \ldots, v_T\}) = \min \sum_{j=1}^{M} a_j$ where the minimum is taken over the set of all possible choices of $M \in \mathbb{N}^+, a_j > 0, U_j = [s_j, t_j]$ with $1 \le s_j \le t_j \le T, s_j, t_j \in \mathbb{N}^+$ $(j = 1, \ldots, M)$ such that $v_t = \sum_{j=1}^{M} a_j \mathbf{1}\{t \in U_j\}$ for all t. Then with v_0 defined to be 0 we have $h(\{v_1, \ldots, v_T\}) = \sum_{t=1}^{T} [v_t - v_{t-1}]_+$.*

The proof of this technical lemma is deferred to Appendix A.4.

Theorem 7 tells us that while playing with time-varying competitors seems to be a harder problem, it is in fact not any harder than its special case: achieving adaptive regret on any interval. Although the result is independent of the algorithms, one still cannot derive bounds on $R(\boldsymbol{u}_{1:T})$ for FLH or fixed share based on their adaptive regret bounds, because when no prior information is available, the bounds on $R_{[t_1,t_2],i}$ for these algorithms are of order $O(\sqrt{t_2})$ instead of $O(\sqrt{t_2 - t_1})$, which is not good enough. We refer the reader to Table 2.3 for a summary of this section.

2.3.4 Competing with the Best Pruning Tree

We now turn to our second application of AdaNormalHedge on predicting almost as well as the best pruning tree within a template tree. This problem was studied in the context of online learning by Helmbold and Schapire [58] using the approach of Willems et al. [107, 108]. It is also called the tree expert problem in Cesa-Bianchi and Lugosi [22]. Freund et al. [40] proposed a generic reduction from a tree expert problem to a sleeping expert problem. Using this reduction with our new algorithm, we provide much better results compared to previous work (summarized in Table 2.4).

Specifically, consider a setting where on each round t, the predictor has to make a decision y_t from some convex set $\mathcal{Y}$ given some side information x_t, and then a convex loss function $f_t : \mathcal{Y} \to [0, 1]$ is revealed and the player suffers loss $f_t(y_t)$. The predictor is given a *template tree* $\mathcal{G}$ to consult. Starting from the root, each node of $\mathcal{G}$ performs a test on x_t to decide which of its children should perform the next test, until a leaf is reached. In addition to a test, each node (except the root) also makes a prediction based on x_t. A *pruning tree* $\mathcal{P}$ is a tree induced by replacing zero or more nodes (and associated subtrees) of $\mathcal{G}$ by leaves. The prediction of a pruning tree $\mathcal{P}$ given x_t, denoted by $\mathcal{P}(x_t)$, is naturally defined as the prediction of the leaf that x_t reaches by traversing $\mathcal{P}$. The player's goal is thus to predict almost as well as the best pruning tree in hindsight, that is, to minimize $R_{\mathcal{G}} = \sum_{t=1}^{T} f_t(y_t) - \min_{\mathcal{P}} \sum_{t=1}^{T} f_t(\mathcal{P}(x_t))$.

The idea of the reduction introduced by Freund et al. [40] is to view each edge of $\mathcal{G}$ as a sleeping expert (indexed by e), who is awake only when traversed by x_t, and in that case predicts $y_{t,e}$, the same prediction as the child node that it connects to. The predictor runs a sleeping expert algorithm with loss $\ell_{t,e} = f_t(y_{t,e})$, and eventually predicts $y_t = \sum_{e \in E} p_{t,e} y_{t,e}$ where E denotes the set of edges of $\mathcal{G}$ and $p_{t,e}$ ($e \in E$) is the output of the expert algorithm; thus by convexity of f_t, we have $f_t(y_t) \le \sum_{e \in E} p_{t,e} f_t(y_{t,e}) = \hat{\ell}_t$. Note that we only care about the predictions of those awake experts since otherwise $p_{t,e}$ is required to be zero. Now let $\mathcal{P}^*$ be one of the best pruning trees, that is, $\mathcal{P}^* \in \arg\min_{\mathcal{P}} \sum_{t=1}^{T} f_t(\mathcal{P}(x_t))$, and m be the number of leaves of $\mathcal{P}^*$. In the expert problem, we will set the competitor $\boldsymbol{u}$ to be a uniform distribution over the m terminal edges (that is, the ones connecting the leaves) of $\mathcal{P}^*$, and the prior $\boldsymbol{q}$ to be a uniform distribution over all the edges. Since on each round, one and only one of those m experts is awake, and its prediction is exactly $\mathcal{P}^*(x_t)$, we have $R(\boldsymbol{u}) = \sum_{t=1}^{T} \frac{1}{m}(\hat{\ell}_t - f_t(\mathcal{P}^*(x_t)))$, and therefore $R_{\mathcal{G}} \le mR(\boldsymbol{u})$.

It remains to pick a concrete sleeping algorithm to apply. There are two reasons that make AdaNormalHedge very suitable for this problem. First, since m is clearly unknown ahead of time, we are competing with an unknown competitor $\boldsymbol{u}$, which is exactly what AdaNormalHedge can deal with. Second, the number of awake experts is dynamically changing, and as discussed before, in this case AdaNormalHedge enjoys a regret bound that is adaptive in the number of experts seen so far. Formally, recall the notation N_t, which in this case represents the total number of distinct traversed edges up to round t. Then by Theorem 6, we have

$$R_{\mathcal{G}} = \hat{O}(m\sqrt{(\boldsymbol{u} \cdot \mathbf{C}_T)\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q})}) = \hat{O}\left(\sqrt{m\left(\sum_{t=1}^{T} |\hat{\ell}_t - f_t(\mathcal{P}^*(x_t))|\right)\ln\left(\tfrac{N_T}{m}\right)}\right),$$

which, by Theorem 5, implies $R_{\mathcal{G}} = \hat{O}\left(\sqrt{m\tilde{L}^* \ln(N_T/m)}\right)$ where similarly we define $\tilde{L}^* = \sum_{t=1}^{T} [f_t(\mathcal{P}^*(x_t)) - \hat{\ell}_t]_+$, which is at most the total loss of the best pruning

Table 2.4: Comparisons of different algorithms for the tree expert problem

Algorithm	$R_{\mathcal{G}}$	Time (per round)	Space (overall)	Need to know L^* and m?
AdaNormalHedge (this work)	$\hat{O}\left(\sqrt{m\tilde{L}^* \ln\left(\frac{N_T}{m}\right)}\right)$	$O\left(\|x_t\|_{\mathcal{G}}\right)$	$O\left(N_T\right)$	No
Adapt-ML-Prod [42]	$\hat{O}\left(\sqrt{m\tilde{L}^* \ln N}\right)$	$O\left(\|x_t\|_{\mathcal{G}}\right)$	$O\left(N_T\right)$	No
Exponential Weights [58]	$O\left(\sqrt{mL^*}\right)$	$O\left(d\|x_t\|_{\mathcal{G}}\right)$	$O\left(N_T\right)$	Yes

tree $L^* = \sum_{t=1}^{T} f_t(\mathcal{P}^*(x_t))$. Moreover, the algorithm is efficient: the overall space requirement is $O(N_T)$, and the running time on round t is $O(\|x_t\|_{\mathcal{G}})$ where we use $\|x_t\|_{\mathcal{G}}$ to denote the number of edges that x_t traverses.

Comparison to other solutions. The work by Freund et al. [40] considers a variant of the exponential weights algorithm in a "log loss" setting, and is not directly applicable here (specifically it is not clear how to tune the learning rate appropriately). A better choice is the Adapt-ML-Prod algorithm by Gaillard et al. [42] (the version for the sleeping expert problem). However, there is still one issue for this algorithm: it does not give a bound in terms of $\mathrm{RE}(\boldsymbol{u} \,||\, \boldsymbol{q})$ for an unknown $\boldsymbol{u}$. So to get a bound on $R(\boldsymbol{u})$, the best thing to do is to use the definition $R(\boldsymbol{u}) = \boldsymbol{u} \cdot \mathbf{R}_T$ and a bound on each $R_{T,i}$. In short, one can verify that Adapt-ML-Prod ensures regret $R(\boldsymbol{u}) = \hat{O}\left(\sqrt{m\tilde{L}^* \ln N}\right)$ where $N = |E|$ is the total number of edges/experts. We emphasize that N can be much larger than N_T when the tree is huge. Indeed, while N_T is at most T times the depth of $\mathcal{G}$, N could be exponentially large in the depth. The running time and space of Adapt-ML-Prod for this problem, however, is the same as AdaNormalHedge.

We finally compare with a totally different approach [58], where one simply treats each pruning tree as an expert and run the exponential weights algorithm. Clearly the number of experts is exponentially large, and thus the running time and space

are unacceptable by a naive implementation. This issue is avoided by using a clever dynamic programming technique. If L^* and m are known ahead of time, then the regret for this algorithm is $O(\sqrt{mL^*})$ by tuning the learning rate optimally. As discussed in Freund et al. [40], the linear dependence on m in this bound is much better than the one of the form $O\left(\sqrt{mL^* \ln N}\right)$, which, in the worst case, is linear in m. This was considered as the main drawback of using the sleeping expert approach. However, the bound for AdaNormalHedge is $\hat{O}\left(\sqrt{m\tilde{L}^* \ln(N_T/m)}\right)$, which is much smaller as discussed previously and in fact comparable to $O(\sqrt{mL^*})$. More importantly, L^* and m are unknown in practice. In this case, no sublinear regret is known for this dynamic programming approach, since it relies heavily on the fact that the algorithm is using a fixed learning rate and thus the usual time-varying learning rate methods cannot be applied here. Therefore, although theoretically this approach gives small regret, it is not a practical method.

The running time is also slightly worse than the sleeping expert approach. For simplicity, suppose every internal node has d children. Then the time complexity per round is $O(d\|x\|_{\mathcal{G}})$. The overall space requirement is $O(N_t)$, the same as other approaches. Again, see Table 2.4 for a summary of this section.

Finally, as mentioned in Freund et al. [40], the sleeping expert approach can be easily generalized to predicting with a decision graph. In that case, AdaNormalHedge still enjoys all the improvements discussed in this section (details omitted).

2.4 Conclusion and Future Directions

In this chapter, we mostly focused on learning by combining expert advice, one of the fundamental online learning problems. We first showed a strong connection between this problem and a variant of the drifting game, and made use of this connection to develop a general recipe to design expert algorithms. Using this framework, we not

only recovered previous algorithms with direct regret guarantees, but also developed an algorithm (NormalHedge.DT) with strong adaptivity in the quantile that we try to compete with. Base on this algorithm, a further improvement AdaNormalHedge provides even stronger adaptivity and enjoys several useful properties simultaneously, such as constant regret when the losses are stochastic. Applications of these algorithms discussed in this thesis include more adaptive and efficient boosting algorithms, parameter-free expert algorithms that can compete with time-varying competitors, and also more efficient algorithms for learning the best pruning tree.

There are still many open problems in this direction. One of the main questions is how to extend all these adaptivity results to the partial information setting, which seems to be highly non-trivial after initial attempts. It is not even clear what kind of adaptive regret bounds that one should be looking for in this setting. Another orthogonal direction is to extend these results to the more general OCO setting. We discuss some preliminary results in Appendix A.3, but the algorithms discussed there are not efficient. Getting both efficiency and adaptivity is a challenging and important future direction.

Chapter 3

Online Boosting

In this chapter, we study online boosting, the task of converting any weak online learner into a strong online learner. While boosting is well-studied in the offline setting, there is very little work on the theory of online boosting. The main question is whether we can improve the accuracy of existing online prediction algorithms in a similar way as we do in the offline case. For example, given an online weak spam filter which on average only has about 51% accuracy, can we somehow build a very accurate spam filter (say, 99% accuracy) by combining several copies of these weak spam filters in an online fashion?

Based on a novel and natural definition of weak online learnability, we develop two online boosting algorithms. The first algorithm is an online version of boost-by-majority. By proving a matching lower bound, we show that this algorithm is essentially optimal in terms of the number of weak learners and the sample complexity needed to achieve a specified accuracy. This optimal algorithm is not adaptive, however. Using tools from online loss minimization, we derive an adaptive online boosting algorithm that is also parameter-free, but not optimal. Both algorithms work with base learners that can handle example importance weights directly, as well as by rejection sampling examples with probability defined by the booster.

We next further extend the theory to online regression problems. The notion of a weak learning algorithm is modeled as an online learning algorithm with linear loss functions that competes with a base class of regression functions, while a strong learning algorithm is an online learning algorithm with smooth convex loss functions that competes with a larger class of regression functions. Our main result is an online gradient boosting algorithm that converts a weak online learning algorithm into a strong one where the larger class of functions is the linear span of the base class. We also give a simpler boosting algorithm that converts a weak online learning algorithm into a strong one where the larger class of functions is the convex hull of the base class, and prove its optimality.

Both results are complemented with experimental studies.

This chapter includes joint work with Alina Beygelzimer and Satyen Kale [14], and joint work with Alina Beygelzimer, Elad Hazan and Satyen Kale [13].

3.1 Online Boosting for Binary Classification

We start with online boosting for binary classification in this section. The theory of boosting in the batch setting has been studied extensively in the literature and has led to huge practical success (see the book by Schapire and Freund [98] for a thorough discussion). Given the success of boosting in the batch setting, it is natural to ask about the possibility of applying boosting to online learning. Indeed, there has already been some work on online boosting [90, 48, 71, 49, 27, 28].

From a theoretical viewpoint, recent work by Chen et al. [27] is perhaps most interesting. They generalized the batch weak learning assumption to the online setting, and made a connection between online boosting and batch boosting that produces smooth distributions over the training examples. The resulting algorithm is guaranteed to achieve an arbitrarily small error rate as long as the number of weak learners

and the number of examples are sufficiently large. No assumptions need to be made about how the data is generated. Indeed, the data can even be generated by an adversary.

We present a new online boosting algorithm, based on the boost-by-majority (BBM) algorithm of [37]. This algorithm, called Online BBM, improves upon the work of Chen et al. [27] in several respects:

1. our assumption on online weak learners is weaker and can be seen as a direct online analogue of the weak learning assumption in standard batch boosting;
2. our algorithm does not require weak learners to deal with weighted examples, instead using a sampling technique to decide whether or not to pass an example to the weak learners as feedback (similar to boosting by filtering in the batch setting [see for example, 36, 19]); and
3. our algorithm is optimal in the sense that no online boosting algorithm can achieve the same error rate with less weak learners or less examples asymptotically (see the lower bounds in Section 3.1.3).

A quantitative comparison of our results with those of Chen et al. [27] appears in Table 3.1, where N and T represent the number of weak learners and examples needed to achieve error rate ϵ, and γ stands for a similar concept of the "edge" of the weak learning oracle as in the batch setting (smaller γ means more inaccurate weak learners).

A clear drawback of all the algorithms mentioned above is lack of adaptivity. A simple interpretation of this drawback is that all these algorithms require using γ, an unknown quantity, as a parameter. More importantly, this also means that the algorithm treats each weak learner equally and ignores the fact that some weak learners are actually doing better than the others. The best example of an adaptive boosting algorithm is the well-known parameter-free AdaBoost algorithm [38], where each weak learner is naturally weighted by how accurate it is. In fact, adaptivity is

Table 3.1: Comparisons of different online boosting algorithms

Algorithm	N	T	Optimal?	Adaptive?
Online BBM (Section 3.1.2)	$O(\frac{1}{\gamma^2}\ln\frac{1}{\epsilon})$	$\tilde{O}(\frac{1}{\epsilon\gamma^2})$	$\surd$	$\times$
AdaBoost.OL (Section 3.1.4)	$O(\frac{1}{\epsilon\gamma^2})$	$\tilde{O}(\frac{1}{\epsilon^2\gamma^4})$	$\times$	$\surd$
OSBoost [27]	$O(\frac{1}{\epsilon\gamma^2})$	$\tilde{O}(\frac{1}{\epsilon\gamma^2})$	$\times$	$\times$

known to be one of the key features that lead to the practical success of AdaBoost, and therefore should also be essential to the performance of online boosting algorithms. In Section 3.1.4, we thus propose AdaBoost.OL, an adaptive and parameter-free online boosting algorithm. As shown in Table 3.1, AdaBoost.OL is theoretically suboptimal in terms of N and T. However, empirically it generally outperforms OSBoost and sometimes even beats the optimal algorithm Online BBM (see Section 3.1.5).

Our techniques are also very different from those of Chen et al. [27], which rely on the smooth boosting algorithm of Servedio [100]. As far as we know, all other work on smooth boosting [20, 19, 9] cannot be easily generalized to the online setting, necessitating completely different methods not relying on smooth distributions. Our Online BBM algorithm builds on top of a potential based family that arises naturally in the batch setting as approximate minimax optimal algorithms for so-called drifting games [97, 73]. The decomposition of each example in that framework naturally allows us to generalize it to the online setting where examples come one by one. On the other hand, AdaBoost.OL is derived by viewing boosting from a different angle: loss minimization [78, 98]. The theory of online loss minimization is the key tool for developing AdaBoost.OL.

3.1.1 Setup, Assumptions and Main Theorem

At each time step $t = 1, \ldots, T$, an adversary chooses an example $(\boldsymbol{x}_t, y_t) \in \mathcal{X} \times \{-1, 1\}$, where $\mathcal{X}$ is the domain, and reveals $\boldsymbol{x}_t$ to the online learner. The learner makes a prediction on its label $\hat{y}_t \in \{-1, 1\}$, and suffers the 0-1 loss $\mathbf{1}\{\hat{y}_t \neq y_t\}$. As is usual with online algorithms, this prediction may be randomized.

For parameters $\gamma \in (0, \frac{1}{2})$, $\delta \in (0, 1)$, a constant $S > 0$ and a set of examples U, the learner is said to be a *weak* online learner with edge γ and *excess loss* S with respect to U if, for any T and for any input sequence of examples $(\boldsymbol{x}_t, y_t) \in U$ for $t = 1, 2, \ldots, T$ chosen adaptively, it generates predictions $\hat{y}_t$ such that with probability at least $1 - \delta$,

$$\sum_{t=1}^{T} \mathbf{1}\{\hat{y}_t \neq y_t\} \leq (\tfrac{1}{2} - \gamma)T + S. \tag{3.1}$$

The excess loss requirement is necessary since an online learner cannot be expected to predict with any accuracy with too few examples. Essentially, the excess loss S yields a kind of sample complexity bound: the weak learner starts obtaining a distinct edge of $\Omega(\gamma)$ over random guessing when $T \gg \frac{S}{\gamma}$. Typically, the dependence of the high probability bound on δ is polylogarithmic in $\frac{1}{\delta}$; thus in the following we will avoid explicitly mentioning δ.

For a given parameter $\epsilon > 0$, the learner is said to be a *strong* online learner with error rate ϵ if it satisfies the same conditions as a weak online learner except that its edge is $\frac{1}{2} - \epsilon$, or in other words, the fraction of mistakes made, asymptotically, is ϵ. Just as for the weak learner, the excess loss S yields a sample complexity bound: the fraction of mistakes made by the strong learner becomes $O(\epsilon)$ when $T \gg \frac{S}{\epsilon}$.

Our main theorem is stated below, which is a direct implication of Theorem 9 in Section 3.1.2 and Theorem 10 in Section 3.1.3.

Theorem 8. *Given a weak online learning algorithm with edge γ and excess loss S and any target error rate $\epsilon > 0$, there is a strong online learning algorithm with error*

rate ϵ which uses $O(\frac{1}{\gamma^2}\ln(\frac{1}{\epsilon}))$ copies of the weak online learner, and has excess loss $\tilde{O}(\frac{S}{\gamma}+\frac{1}{\gamma^2})$; thus its sample complexity is $\tilde{O}(\frac{1}{\epsilon}(\frac{S}{\gamma}+\frac{1}{\gamma^2}))$. Furthermore, if $S \geq \tilde{\Omega}(\frac{1}{\gamma})$, then the number of weak online learners is optimal up to constant factors, and the sample complexity is optimal up to polylogarithmic factors.

The requirement that $S \geq \tilde{\Omega}(\frac{1}{\gamma})$ in the lower bound is not very stringent; this is precisely the excess loss one obtains when using standard online learning algorithms with regret bound $O(\sqrt{T})$, as is explained in the discussion following Lemma 7. Furthermore, since we require the bound (3.1) to hold with high probability, typical analyses of online learning algorithms will have an $\tilde{O}(\sqrt{T})$ deviation term, which also leads to $S \geq \tilde{\Omega}(\frac{1}{\gamma})$.

As the theorem indicates, the strong online learner (hereafter referred to as "booster") works by maintaining N copies of the weak online learner, for some positive integer N to be specified later. Denote the weak online learners WL^i for $i = 1, 2, \ldots, N$. At time step t, the prediction of the i-th weak online learner is given by $\mathrm{WL}^i(\boldsymbol{x}_t) \in \{-1, 1\}$. Note the slight abuse of notation here: WL^i is *not* a function; rather it is an algorithm with an internal state that is updated as it is fed training examples. Thus, the prediction $\mathrm{WL}^i(\boldsymbol{x}_t)$ depends on the internal state of WL^i, and for notational convenience we avoid reference to the internal state.

In each round t, the booster works by taking a weighted majority vote of the weak learners' predictions. Specifically, the booster maintains weights $\alpha_t^i \in \mathbb{R}$ for $i = 1, \ldots, N$ corresponding to each weak learner, and its final prediction will then be[8] $\hat{y}_t = \mathrm{sign}(\sum_{i=1}^N \alpha_t^i \mathrm{WL}^i(\boldsymbol{x}_t))$. After making the prediction, the true label y_t is revealed by the environment. The booster then updates WL^i by passing the training example $(\boldsymbol{x}_t, y_t)$ to WL^i with a carefully chosen sampling probability p_t^i (and not passing the example with the remaining probability). The sampling probability p_t^i is obtained by

[8] In Section 3.1.4 a slightly different final prediction will be used.

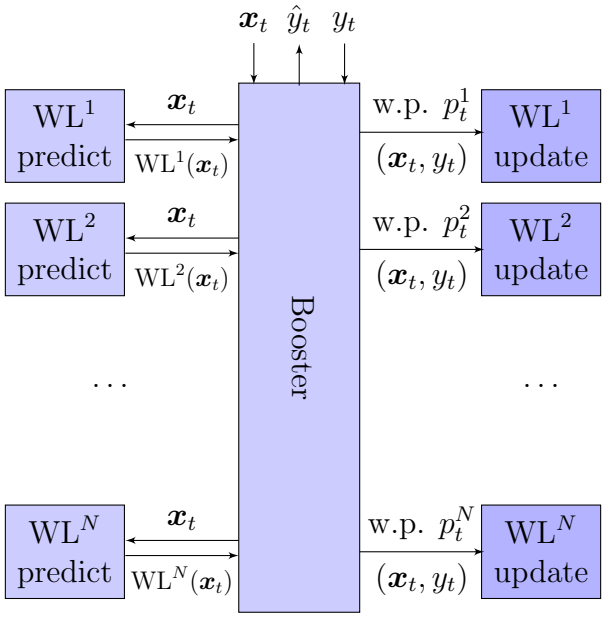


Figure 3.1: An illustration of the online boosting algorithm on round t

computing a weight w_t^i and setting[9] $p_t^i = \frac{w_t^i}{\|\boldsymbol{w}^i\|_\infty}$, where $\boldsymbol{w}^i = \langle w_1^i, w_2^i, \ldots, w_T^i \rangle$. At the same time the booster updates α_t^i as well, and then it is ready to make a prediction for the next round. See Figure 3.1 for an illustration.

We introduce some more notation to ease the presentation. Let $z_t^i = y_t \mathrm{WL}^i(\boldsymbol{x}_t)$ and $s_t^i = s_t^{i-1} + \alpha_t^i z_t^i$ with $s_t^0 = 0$. Define $\boldsymbol{z}^i = \langle z_1^i, z_2^i, \ldots, z_T^i \rangle$. Finally, a martingale concentration bound using (3.1) yields the following bound (proof deferred to Appendix B.1). The bound can be seen as a weighted version of (3.1) which is necessary for the rest of the analysis.

Lemma 6. *There is a constant $\tilde{S} = 2S + \tilde{O}(\frac{1}{\gamma})$ such that for any T, with high probability, for every weak learner WL^i we have $\boldsymbol{w}^i \cdot \boldsymbol{z}^i \geq \gamma \|\boldsymbol{w}^i\|_1 - \tilde{S}\|\boldsymbol{w}^i\|_\infty$.*

Handling Importance Weights

Typical online learning algorithms can handle *importance weighted* examples: each example $(\boldsymbol{x}_t, y_t)$ comes with a weight $p_t \in [0, 1]$, and the loss on that example is scaled

[9] In the algorithm we simply use a tight-enough upper bound on $\|\boldsymbol{w}^i\|_\infty$ (such as the bound from Lemma 9) to compute the values p_t^i; we abuse notation here and use $\|\boldsymbol{w}^i\|_\infty$ to also denote this upper bound.

by p_t, i.e. the loss for predicting $\hat{y}_t$ is $p_t \mathbf{1}\{\hat{y}_t \neq y_t\}$. Consider the following natural extension to the definition of online weak learners which incorporates importance weighted examples: we now require that for any sequence of weighted examples $(\boldsymbol{x}_t, y_t)$ with weight $p_t \in [0,1]$ for $t = 1, 2, \ldots, T$, the online learner generates predictions $\hat{y}_t$ such that with probability at least $1 - \delta$,

$$\sum_{t=1}^{T} p_t \mathbf{1}\{\hat{y}_t \neq y_t\} \leq (\tfrac{1}{2} - \gamma) \sum_{t=1}^{T} p_t + S. \tag{3.2}$$

Having access to such a weak learner makes the boosting algorithm simpler: we now simply pass every example $(\boldsymbol{x}_t, y_t)$ to every weak learner WL^i setting the importance weight equal to the probability value $p_t^i = \frac{w_t^i}{\|\boldsymbol{w}^i\|_\infty}$ computed in the algorithm. The advantage is that the bound (3.2) immediately implies the following inequality for any weak learner WL^i, which can be seen as a (stronger) analogue of Lemma 6.

$$\boldsymbol{w}^i \cdot \boldsymbol{z}^i \geq 2\gamma \|\boldsymbol{w}^i\|_1 - 2S\|\boldsymbol{w}^i\|_\infty. \tag{3.3}$$

Since the analysis depends only on the bound in Lemma 6, if we use the importance-weighted version of the boosting algorithm, then we can simply use inequality (3.3) instead in the analysis, which gives a slightly tighter version of Theorem 8, viz. the excess loss can now be bounded by $O(\frac{S}{\gamma})$.

In the rest of the paper, for simplicity of exposition we assume that the p_t^i's are used as sampling probabilities rather than importance weights, and give the analysis using the bound from Lemma 6. In experiments, however, using the p_t^i's as importance weights rather than sampling probabilities led to better performance.

Discussion of Weak Online Learning Assumption

We now justify our definition of weak online learning, viz. inequality (3.1). In the standard batch boosting case, the corresponding weak learning assumption (see for

example Schapire and Freund [98]) is that there is an algorithm which, given a training set of examples and an arbitrary distribution on it, generates a hypothesis that has error at most $\frac{1}{2} - \gamma$ on the training data under the given distribution. This statement can be interpreted as making the following two implicit assumptions:

1. (Richness.) Given an edge parameter $\gamma \in (0, \frac{1}{2})$, there is a set of hypotheses, $\mathcal{H}$, such that, given any training set (possibly, a multiset) of examples U, there is some hypothesis $h \in \mathcal{H}$ with error at most $\frac{1}{2} - \gamma$, i.e.

$$\sum_{(\boldsymbol{x},y)\in U} \mathbf{1}\{h(\boldsymbol{x}) \neq y\} \leq (\tfrac{1}{2} - \gamma)|U|.$$

2. (Agnostic Learnability.) For any $\epsilon \in (0, 1)$, there is an algorithm which, given any training set (possibly, a multiset) of examples U, can compute a nearly optimal hypothesis $h \in \mathcal{H}$, i.e.

$$\sum_{(\boldsymbol{x},y)\in U} \mathbf{1}\{h(\boldsymbol{x}) \neq y\} \leq \inf_{h'\in\mathcal{H}} \sum_{(\boldsymbol{x},y)\in U} \mathbf{1}\{h'(\boldsymbol{x}) \neq y\} + \epsilon|U|.$$

Our weak online learning assumption can be seen as arising from a direct generalization of the above two assumptions to the online setting. Namely, the richness assumption stays the same, whereas the agnostic learnability of $\mathcal{H}$ assumption is replaced by an agnostic online learnability of $\mathcal{H}$ assumption (c.f. Ben-David et al. [12]). That is, there is an online learning algorithm which, given any sequence of examples, $(\boldsymbol{x}_t, y_t)$ for $t = 1, 2, \ldots, T$, generates predictions $\hat{y}_t$ such that

$$\sum_{t=1}^{T} \mathbf{1}\{\hat{y}_t \neq y_t\} \leq \inf_{h\in\mathcal{H}} \sum_{t=1}^{T} \mathbf{1}\{h(\boldsymbol{x}_t) \neq y_t\} + R(T),$$

where $R : \mathbb{N} \to \mathbb{R}_+$ is the regret, a non-decreasing, sublinear function of the number of prediction periods T. Since online learning algorithms are typically randomized, we

assume the above bound holds with high probability. The following lemma shows that richness and agnostic online learnability immediately imply our online weak learning assumption (3.1):

Lemma 7. *Suppose the sequence of examples $(\boldsymbol{x}_t, y_t)$ is chosen from a data set for which there exists a hypothesis class $\mathcal{H}$ that is both rich for edge parameter 2γ and agnostically online learnable with regret $R(\cdot)$. Then, the agnostic online learning algorithm for $\mathcal{H}$ satisfies the weak learning assumption (3.1), with edge γ and excess loss $S = \max_T(R(T) - \gamma T)$.*

Proof. For the given sequence of examples $(\boldsymbol{x}_t, y_t)$ for $t = 1, 2, \ldots, T$, the richness with edge parameter 2γ and agnostic online learnability assumptions on $\mathcal{H}$ imply that with high probability, the predictions $\hat{y}_t$ generated by the agnostic online learning algorithm for $\mathcal{H}$ satisfy

$$\sum_{t=1}^{T} \mathbf{1}\{\hat{y}_t \neq y_t\} \leq (\tfrac{1}{2} - 2\gamma)T + R(T).$$

It only remains to show that $(\frac{1}{2} - 2\gamma)T + R(T) \leq (\frac{1}{2} - \gamma)T + S$, or equivalently, $R(T) \leq \gamma T + S$, which is true by the definition of S. This concludes the proof. □

Various agnostic online learning algorithms are known that have a regret bound of $O(\sqrt{T \ln(\frac{1}{\delta})})$, such as a standard experts algorithm on a finite hypothesis space such as the exponential weights algorithm. If we use such an online learning algorithm as a weak online learner, then a simple calculation implies, via Lemma 7, that it has excess loss $\Theta(\frac{\ln(\frac{1}{\delta})}{\gamma})$. Thus, by Theorem 8, we obtain an online boosting algorithm with near-optimal sample complexity.

3.1.2 An Optimal Algorithm

We will again make use of drifting-games and generalize its analysis to obtain a potential based family of online boosting algorithms. Specifically, pick a sequence of

$N+1$ non-increasing potential functions $\Phi_i(s)$ such that

$$\begin{aligned} \Phi_N(s) &\geq \mathbf{1}\{s \leq 0\}, \\ \Phi_{i-1}(s) &\geq (\tfrac{1}{2} - \tfrac{\gamma}{2})\Phi_i(s-1) + (\tfrac{1}{2} + \tfrac{\gamma}{2})\Phi_i(s+1). \end{aligned} \tag{3.4}$$

(See [98, Chapter 13] for intuition behind these potential functions.) Then the algorithm is simply to set $\alpha_t^i = 1$ and $w_t^i = \frac{1}{2}(\Phi_i(s_t^{i-1} - 1) - \Phi_i(s_t^{i-1} + 1))$. The following theorem states the error rate bound of this general scheme.

Lemma 8. *For any T and N, with high probability, the number of mistakes made by the algorithm described above is bounded as follows (recall that $\tilde{S}$ is defined in Lemma 6):*

$$\sum_{t=1}^{T} \mathbf{1}\{\hat{y}_t \neq y_t\} \leq \Phi_0(0)T + \tilde{S} \sum_i \|\boldsymbol{w}^i\|_\infty.$$

Proof. The key property of the algorithm is that for any fixed i and t, one can verify the following:

$$\begin{aligned} \Phi_i(s_t^i) + w_t^i(z_t^i - \gamma) &= \Phi_i(s_t^{i-1} + z_t^i) + w_t^i(z_t^i - \gamma) \\ &= (\tfrac{1}{2} - \tfrac{\gamma}{2})\Phi_i(s_t^{i-1} - 1) + (\tfrac{1}{2} + \tfrac{\gamma}{2})\Phi_i(s_t^{i-1} + 1) \leq \Phi_{i-1}(s_t^{i-1}) \end{aligned}$$

by plugging the formula of w_t^i, realizing that z_t^i can only be -1 or 1, and using the definition of $\Phi_{i-1}(s)$ from Eq. (3.4). Summing over t we get

$$\sum_{t=1}^{T} \Phi_i(s_t^i) + \boldsymbol{w}^i \cdot \boldsymbol{z}^i - \gamma\|\boldsymbol{w}^i\|_1 \;\leq\; \sum_{t=1}^{T} \Phi_{i-1}(s_t^{i-1}).$$

Using Lemma 6, we get $\sum_{t=1}^{T} \Phi_i(s_t^i) \;\leq\; \sum_{t=1}^{T} \Phi_{i-1}(s_t^{i-1}) + \tilde{S}\|\boldsymbol{w}^i\|_\infty$ which relates the sums of all examples' potential for two successive weak learners. We can therefore apply this inequality iteratively to arrive at: $\sum_{t=1}^{T} \Phi_N(s_t^N) \leq \sum_{t=1}^{T} \Phi_0(0) + \tilde{S} \sum_i \|\boldsymbol{w}^i\|_\infty$.

The proof is completed by noting that $\Phi_N(s_t^N) \geq \mathbf{1}\{s_t^N \leq 0\} = \mathbf{1}\{\hat{y}_t \neq y_t\}$ since $y_t\hat{y}_t = \text{sign}(s_t^N)$ by definition. □

Note that the term $\tilde{S}\sum_i \|\boldsymbol{w}^i\|_\infty$ becomes a penalty for the final error rate. Therefore, we naturally want this penalty term to be relatively small. This is not necessarily true for any choice of the potential function. For example, if $\Phi_i(s)$ is the exponential potential that leads to a variant of AdaBoost in the batch setting [see 98, Chap. 13], then the weight w_t^i could be exponentially large.

Fortunately, there is indeed a set of potential functions that produce small weights, which, in the batch setting, corresponds to an algorithm called boost-by-majority (BBM) [37]. All we need to do is to let Eq. (3.4) hold with equality, and direct calculation shows:

$$\Phi_i(s) = \sum_{k=0}^{\lfloor \frac{N-i-s}{2} \rfloor} \binom{N-i}{k} \left(\frac{1}{2}+\frac{\gamma}{2}\right)^k \left(\frac{1}{2}-\frac{\gamma}{2}\right)^{N-i-k},$$

and

$$w_t^i = \frac{1}{2}\binom{N-i}{k_t^i} \left(\frac{1}{2}+\frac{\gamma}{2}\right)^{k_t^i} \left(\frac{1}{2}-\frac{\gamma}{2}\right)^{N-i-k_t^i} \tag{3.5}$$

where $k_t^i = \lfloor \frac{N-i-s_t^{i-1}+1}{2} \rfloor$ and $\binom{n}{k}$ is defined to be 0 if $k < 0$ or $k > n$. In other words, imagine flipping a biased coin whose probability of heads is $\frac{1}{2}+\frac{\gamma}{2}$ for $N-i$ times. Then $\Phi_i(s)$ is exactly the probability of seeing at most $(N-i-s)/2$ heads and w_t^i is half of the probability of seeing k_t^i heads. We call this algorithm *Online BBM*. The pseudocode is given in Algorithm 7.

One can see that the weights produced by this algorithm are small since trivially $w_t^i \leq 1/2$. To get a better result, however, we need a better estimate of $\|\boldsymbol{w}^i\|_\infty$ stated in the following lemma.

Lemma 9. *If w_t^i is defined as in Eq. (3.5), then we have $w_t^i = O(1/\sqrt{N-i})$ for any $i < N$.*

Algorithm 7 Online BBM

for $t = 1$ **to** T **do**
  Receive example $\boldsymbol{x}_t$.
  Predict $\hat{y}_t = \text{sign}(\sum_{i=1}^{N} \text{WL}^i(\boldsymbol{x}_t))$, receive label y_t.
  Set $s_t^0 = 0$.
  for $i = 1$ **to** N **do**
    Set $s_t^i = s_t^{i-1} + y_t \text{WL}^i(\boldsymbol{x}_t)$.
    Set $k_t^i = \lfloor \frac{N-i-s_t^{i-1}+1}{2} \rfloor$.
    Set $w_t^i = \binom{N-i}{k_t^i} \left(\frac{1}{2} + \frac{\gamma}{2}\right)^{k_t^i} \left(\frac{1}{2} - \frac{\gamma}{2}\right)^{N-i-k_t^i}$.
    Pass training example $(\boldsymbol{x}_t, y_t)$ to WL^i with probability $p_t^i = \frac{w_t^i}{\|\boldsymbol{w}^i\|_\infty}$.
  end for
end for

Proof. Let $X \sim B(m, p)$ be a binomial random variable where $m = N - i$ and $p = 1/2 + \gamma/2$. Also let $q = 1 - p$ and F_X be the CDF of X. By the definition of w_t^i, we have $w_t^i \leq \frac{1}{2} \max_k \Pr\{X = k\}$. We will approximate X by a Gaussian random variable $G \sim N(mp, mpq)$ with density function f and CDF F_G. Note that

$$\begin{aligned}\left|\Pr\{X = k\} - \int_{k-1}^{k} f(G)dG\right| &= |\,(F_X(k) - F_X(k-1)) - (F_G(k) - F_G(k-1))\,| \\ &\leq |F_X(k) - F_G(k)| + |F_X(k-1) - F_G(k-1)|.\end{aligned}$$

So by applying Berry-Esseen theorem to the above two CDF differences between X and G, we arrive at

$$\left|\Pr\{X = k\} - \int_{k-1}^{k} f(G)dG\right| \leq \frac{2C(p^2 + q^2)}{\sqrt{mpq}},$$

where C is the universal constant stated in Berry-Esseen theorem. It remains to point out that

$$\begin{aligned}\Pr\{X = k\} &\leq \int_{k-1}^{k} f(G)dG + \frac{2C(p^2+q^2)}{\sqrt{mpq}} \\ &\leq \max_{G\in R} f(G) + \frac{2C(p^2+q^2)}{\sqrt{mpq}} \\ &= \frac{1}{\sqrt{2\pi mpq}} + \frac{2C(p^2+q^2)}{\sqrt{mpq}} = O\left(\frac{1}{\sqrt{m}}\right),\end{aligned}$$

since $pq = 1/4 - \gamma^2/4 \geq 3/16$. □

We are now ready to state the main results of Online BBM.

Theorem 9. *For any T and N, with high probability, the number of mistakes made by the Online BBM algorithm is bounded as follows:*

$$\exp(-\tfrac{1}{2}N\gamma^2)T + \tilde{O}(\sqrt{N}(S + \tfrac{1}{\gamma})). \tag{3.6}$$

Thus, in order to achieve error rate ϵ, it suffices to use $N = \Theta(\frac{1}{\gamma^2}\ln\frac{1}{\epsilon})$ weak learners, which gives an excess loss bound of $\tilde{\Theta}(\frac{S}{\gamma} + \frac{1}{\gamma^2})$.

Proof. A direct application of Hoeffding's inequality gives $\Phi_0(0) \leq \exp(-N\gamma^2/2)$. With Lemma 9 we have $\sum_i \|\boldsymbol{w}^i\|_\infty = O\left(\sum_{i=1}^{N-1}\frac{1}{\sqrt{N-i}}\right) = O(\sqrt{N})$. Applying Lemma 8 proves Eq. (3.6). Now if we set $N = \frac{2}{\gamma^2}\ln\frac{1}{\epsilon}$, we arrive at

$$\sum_{t=1}^{T} \mathbf{1}\{\hat{y}_t \neq y_t\} \leq \epsilon T + \tilde{O}(\sqrt{N}(S + \tfrac{1}{\gamma})) = \epsilon T + \tilde{O}(\tfrac{S}{\gamma} + \tfrac{1}{\gamma^2}).$$

□

3.1.3 Matching Lower Bounds

We give lower bounds for the number of weak learners and the sample complexity in this section that show that our Online BBM algorithm is optimal up to logarithmic factors. To show this, we first need to formally define online boosting algorithms. Recall that in the online prediction setting, at each time step $t = 1, \ldots, T$, an adversary chooses an example $(\boldsymbol{x}_t, y_t) \in \mathcal{X} \times \{-1, 1\}$ adaptively, and reveals $\boldsymbol{x}_t$ to the online learner. The online learner is called an online boosting algorithm if on each round it makes at most N calls to some weak learners, and its prediction $\hat{y}_t$ depends only on the current example $\boldsymbol{x}_t$, the predictions of the weak learners for $\boldsymbol{x}_t$, and the entire history (including $(\boldsymbol{x}_s, y_s)$ for $s = 1, \ldots, t-1$ and the weak learners' predictions on these examples).

Note that while our online boosting algorithms always predict by considering some weighted majority vote of the weak learners, our lower bound statement holds for a much wider range of algorithms defined above.

Theorem 10. *For any $\gamma \in (0, \frac{1}{4})$, $S \geq \frac{\ln(\frac{1}{\delta})}{\gamma}$, $\delta \in (0, 1)$ and $\epsilon \in (0, 1)$, there is a weak online learning algorithm with edge γ and excess loss S satisfying (3.1) with probability at least $1 - \delta$, such that to achieve error rate ϵ, an online boosting algorithm needs at least $\Omega(\frac{1}{\gamma^2} \ln \frac{1}{\epsilon})$ weak learners and a sample complexity of $\Omega(\frac{S}{\epsilon\gamma}) = \Omega(\frac{1}{\epsilon}(\frac{S}{\gamma} + \frac{1}{\gamma^2}))$.*

Proof. The proof of both lower bounds uses a similar construction. In either case, all examples' labels are generated uniformly at random from $\{-1, 1\}$, and in time period t, each weak learner outputs the correct label y_t independently of all other weak learners and other examples with a certain probability p_t to be specified later. Thus, for any T, by the Azuma-Hoeffding inequality, with probability at least $1 - \delta$, the predictions $\hat{y}_t$ made by the weak learner satisfy

$$\sum_{t=1}^{T} \mathbf{1}\{y_t \neq \hat{y}_t\} \leq \sum_{t=1}^{T}(1 - p_t) + \sqrt{2T \ln(\tfrac{1}{\delta})} \leq \sum_{t=1}^{T}(1 - p_t) + \gamma T + \frac{\ln(\frac{1}{\delta})}{2\gamma} \tag{3.7}$$

where the last inequality follows by the arithmetic mean-geometric mean inequality. We will now carefully choose p_t so that inequality (3.7) implies inequality (3.1).

For the lower bound on the number of weak learners, we set $p_t = \frac{1}{2} + 2\gamma$, so that inequality (3.7) implies that with probability at least $1 - \delta$, the predictions $\hat{y}_t$ made by the weak learner satisfy

$$\sum_{t=1}^{T} \mathbf{1}\{y_t \neq \hat{y}_t\} \leq (\tfrac{1}{2} - \gamma)T + \frac{\ln(\frac{1}{\delta})}{2\gamma} \leq (\tfrac{1}{2} - \gamma)T + S.$$

Thus, the weak online learner has edge γ with excess loss S. In this case, the Bayes optimal output of a booster using N weak learners is to simply take a majority vote of all the weak learners [see for instance 98, Chap. 13.2.6], and the probability that the majority vote is incorrect is $\Theta(\exp(-8N\gamma^2))$. Setting this error to ϵ and solving for N gives the desired lower bound.

Now we turn to the lower bound on the sample complexity. We divide the whole process into two phases: for $t \leq T_0 = \frac{S}{4\gamma}$, we set $p_t = \frac{1}{2}$, and for $t > T_0$, we set $p_t = \frac{1}{2} + 2\gamma$. Now, if $T \leq T_0$, inequality (3.7) implies that with probability at least $1 - \delta$, the predictions $\hat{y}_t$ made by the weak learner satisfy

$$\sum_{t=1}^{T} \mathbf{1}\{y_t \neq \hat{y}_t\} \leq (\tfrac{1}{2} + \gamma)T + \frac{\ln(\frac{1}{\delta})}{2\gamma} \leq (\tfrac{1}{2} - \gamma)T + S \tag{3.8}$$

using the fact that $T \leq T_0 = \frac{S}{4\gamma}$ and $S \geq \frac{\ln(\frac{1}{\delta})}{\gamma}$. Next, if $T > T_0$, let $T' = T - T_0$, and again inequality (3.7) implies that with probability at least $1 - \delta$, the predictions $\hat{y}_t$ made by the weak learner satisfy

$$\sum_{t=1}^{T} \mathbf{1}\{y_t \neq \hat{y}_t\} \leq \tfrac{1}{2}T_0 + (\tfrac{1}{2} - 2\gamma)T' + \gamma T + \frac{\ln(\frac{1}{\delta})}{2\gamma} = (\tfrac{1}{2} - \gamma)T + 2\gamma T_0 + \frac{\ln(\frac{1}{\delta})}{2\gamma} \leq (\tfrac{1}{2} - \gamma)T + S, \tag{3.9}$$

since $S \geq \frac{\ln(\frac{1}{\delta})}{\gamma}$. Inequalities (3.8) and (3.9) imply that the weak online learner has edge γ with excess loss S.

However, in the first phase (i.e. $t \leq T_0$), since the predictions of the weak learners are uncorrelated with the true labels, it is clear that no matter what the booster does, it makes a mistake with probability $\frac{1}{2}$. Thus, it will make $\Omega(T_0)$ mistakes with high probability in the first phase, and thus to achieve ϵ error rate, it needs at least $\Omega(T_0/\epsilon) = \Omega(\frac{S}{\epsilon\gamma})$ examples. □

3.1.4 An Adaptive Algorithm

Although the Online BBM algorithm is optimal, it is unfortunately not adaptive since it requires knowledge of γ as a parameter, which is unknown ahead of time. As discussed in the introduction, adaptivity is essential to the practical performance of boosting algorithms such as AdaBoost.

In this section we thus study adaptive online boosting algorithms using the theory of online loss minimization as the main tool. It is known that boosting can be viewed as trying to find a linear combination of weak hypotheses to minimize the total loss of the training examples, usually using functional gradient descent [see for details 98, Chap. 7]. AdaBoost, for instance, minimizes the exponential loss. Here, as discussed before, we intuitively want to avoid using exponential loss since it could lead to large weights. Instead, we will consider logistic loss $\ell(s) = \ln(1 + \exp(-s))$, which results in an algorithm called AdaBoost.L in the batch setting [98, Chap. 7].

In the online setting, we conceptually define N different "experts" giving advice on what to predict on the current example $\boldsymbol{x}_t$. In round t, expert i predicts by combining the first i weak learners: $\hat{y}_t^i = \text{sign}(\sum_{j=1}^{i} \alpha_t^j \text{WL}^j(\boldsymbol{x}_t))$. Now, as in AdaBoost.L, the weight w_t^i for the example $(\boldsymbol{x}_t, y_t)$ used for computing the sampling probability p_t^i for WL^i is obtained by computing the logistic loss of the prediction of expert $i-1$, i.e.

$\ell(s_t^{i-1})$, and then setting w_t^i to be the negative derivative of the loss:

$$w_t^i = -\ell'(s_t^{i-1}) = \frac{1}{1+\exp(s_t^{i-1})} \in [0,1].$$

In terms of the weight α_t^i of WL^i, ideally we wish to mimic AdaBoost.L and use a fixed α^i for all t such that the total logistic loss is minimized: $\alpha^i = \arg\min_\alpha \sum_{t=1}^T \ell(s_t^{i-1} + \alpha z_t^i)$. Of course this is not possible because α^i depends on the future unknown examples. Nevertheless, it turns out that we can almost achieve that using tools from online learning theory. Indeed, one of the fundamental topics in online learning is exactly how to perform almost as well as the best fixed choice (α^i) in hindsight.

Specifically, it turns out that it suffices to restrict α to the feasible set $[-2,2]$. Then consider the following simple one dimensional online learning problem: on each round t, the algorithm predicts α_t^i from a feasible set $[-2,2]$; the environment then reveals loss function $f_t(\alpha) = \ell(s_t^{i-1} + \alpha z_t^i)$ and the algorithm suffers loss $f_t(\alpha_t^i)$. There are many so-called "low-regret" algorithms in the literature (see the survey by Shalev-Shwartz [101]) for this problem ensuring

$$\sum_{t=1}^T f_t(\alpha_t^i) - \min_{\alpha\in[-2,2]} \sum_{t=1}^T f_t(\alpha) \le R_T^i,$$

where R_T^i is sublinear in T so that on average it goes to 0 when T is large and the algorithm is thus doing almost as well as the best constant choice α^i. The simplest low-regret algorithm in this case is perhaps *online gradient descent* [112]:

$$\alpha_{t+1}^i = \Pi\left(\alpha_t^i - \eta_t f_t'(\alpha_t^i)\right) = \Pi\left(\alpha_t^i + \frac{\eta_t z_t^i}{1+\exp(s_t^i)}\right),$$

where η_t is a time-varying learning rate and Π represents projection onto the set $[-2,2]$, i.e., $\Pi(\cdot) = \max\{-2, \min\{2, \cdot\}\}$. Since the loss function is actually 1-Lipschitz ($|f_t'(\alpha)| \le 1$), if we set η_t to be $4/\sqrt{t}$, then standard analysis shows $R_T^i = 4\sqrt{T}$.

Finally, it remains to specify the algorithm's final prediction $\hat{y}_t$. In Online BBM, we simply used the advice of expert N. Unfortunately the algorithm described in this section cannot guarantee that expert N will always make highly accurate predictions. However, as we will show in the proof of Theorem 11, the algorithm does ensure that at least *one of the N experts* will have high accuracy. Therefore, what we really need to do is to decide which expert to follow on each round, and try to predict almost as well as the best fixed expert in hindsight. This is exactly the expert problem that we studied in Chapter 2, and can be solved, for instance, by the exponential algorithm.

We call the final resulting algorithm AdaBoost.OL (O stands for online and L stands for logistic loss), and summarize it in Algorithm 8. Note that as promised, AdaBoost.OL is an adaptive online boosting algorithm and does not require knowing γ in advance. In fact, in the analysis we do not even assume that the weak learners satisfy the bound (3.1). Instead, define the quantities $\gamma_i \triangleq \frac{\boldsymbol{w}^i \cdot \boldsymbol{z}^i}{2\|\boldsymbol{w}^i\|_1}$ for each weak learner WL^i. This can be interpreted as the (weighted) edge over random guessing that WL^i obtains. Note that γ_i may even be negative, which means flipping the sign of WL^i's predictions performs better than random guessing. Nevertheless, the algorithm can still make accurate predictions even with negative γ_i since it will end up choosing negative weights α_t^i in that case. The performance of AdaBoost.OL is provided below.

Theorem 11. *For any T and N, with high probability, the number of mistakes made by AdaBoost.OL is bounded by $\frac{2T}{\sum_i \gamma_i^2} + \tilde{O}\left(\frac{N^2}{\sum_i \gamma_i^2}\right)$.*

Proof. Let the number of mistakes made by expert i be $M_i \triangleq \sum_{t=1}^{T} \mathbf{1}\{y_t \neq \hat{y}_t^i\}$, also define $M_0 = T$ for convenience. Note that AdaBoost.OL is using a variant of the expert algorithm with $\mathbf{1}\{y_t \neq \hat{y}_t^i\}$ being the loss of expert i on round t (Line 7 and 15). So by standard analysis [see e.g. 22, Corollary 2.3], and the Azuma-Hoeffding

Algorithm 8 AdaBoost.OL

1: **Initialize:** $\forall i : v_1^i = 1, \alpha_1^i = 0$.
2: **for** $t = 1$ **to** T **do**
3: Receive example $\boldsymbol{x}_t$.
4: **for** $i = 1$ **to** N **do**
5: Set $\hat{y}_t^i = \text{sign}(\sum_{j=1}^{i} \alpha_t^j \text{WL}^j(\boldsymbol{x}_t))$.
6: **end for**
7: Randomly pick i_t with $\Pr[i_t = i] \propto v_t^i$.
8: Predict $\hat{y}_t = \hat{y}_t^{i_t}$, receive label y_t.
9: Set $s_t^0 = 0$.
10: **for** $i = 1$ **to** N **do**
11: Set $z_t^i = y_t \text{WL}^i(\boldsymbol{x}_t)$.
12: Set $s_t^i = s_t^{i-1} + \alpha_t^i z_t^i$.
13: Set $\alpha_{t+1}^i = \Pi\left(\alpha_t^i + \frac{\eta_t z_t^i}{1+\exp(s_t^i)}\right)$ with $\eta_t = 4/\sqrt{t}$.
14: Pass example $(\boldsymbol{x}_t, y_t)$ to WL^i with probability $p_t^i = w_t^i = 1/(1+\exp(s_t^{i-1}))$.
15: Set $v_{t+1}^i = v_t^i \cdot \exp(-\mathbf{1}\{y_t \neq \hat{y}_t^i\})$.
16: **end for**
17: **end for**

inequality, we have with high probability

$$\sum_{t=1}^{T} \mathbf{1}\{y_t \neq \hat{y}_t\} \leq 2 \min_i M_i + 2\ln(N) + \tilde{O}(\sqrt{T}). \tag{3.10}$$

Now, whenever expert $i-1$ makes a mistake (i.e. $s_t^{i-1} \leq 0$), we have $w_t^i = 1/(1+\exp(s_t^{i-1})) \geq 1/2$ and therefore

$$\|\boldsymbol{w}^i\|_1 \geq M_{i-1}/2. \tag{3.11}$$

Note that Eq. (3.11) holds even for $i = 1$ by the definition of M_0. We now bound the difference between the logistic loss of two successive experts, $\Delta_i \triangleq \sum_{t=1}^{T} \left(\ell(s_t^i) - \ell(s_t^{i-1})\right)$. Online gradient descent (Line 13) ensures that

$$\sum_{t=1}^{T} \ell(s_t^i) \leq \min_{\alpha \in [-2,2]} \sum_{t=1}^{T} \ell(s_t^{i-1} + \alpha z_t^i) + 4\sqrt{T}, \tag{3.12}$$

as discussed previously. On the other hand, direct calculation shows $\ell(s_t^{i-1} + \alpha z_t^i) - \ell(s_t^{i-1}) = \ln\left(1 + w_t^i(e^{-\alpha z_t^i} - 1)\right) \leq w_t^i(e^{-\alpha z_t^i} - 1)$. With $\sigma_i \triangleq \sum_{t=1}^T \frac{w_t^i}{\|\boldsymbol{w}^i\|_1}\mathbf{1}\{z_t^i = 1\} = \frac{1}{2} + \gamma_i$, we thus have

$$\begin{aligned}
\min_{\alpha\in[-2,2]} \sum_{t=1}^T \left(\ell(s_t^{i-1} + \alpha z_t^i) - \ell(s_t^{i-1})\right) &\leq \min_{\alpha\in[-2,2]} \|\boldsymbol{w}^i\|_1(\sigma_i e^{-\alpha} + (1-\sigma_i)e^{\alpha} - 1) \\
&\leq -\tfrac{1}{2}\|\boldsymbol{w}^i\|_1(2\sigma_i - 1)^2 \qquad (3.13) \\
&= -2\gamma_i^2\|\boldsymbol{w}^i\|_1 \qquad (3.14) \\
&\leq -\gamma_i^2 M_{i-1}. \qquad (3.15)
\end{aligned}$$

Here, inequality (3.13) follows from Lemma 10 (proven below) and inequality (3.15) from inequality (3.11). The above inequality and inequality (3.12) imply that $\Delta_i \leq -\gamma_i^2 M_{i-1} + 4\sqrt{T}$. Summing over $i = 1, \ldots, N$ and rearranging gives

$$\sum_{i=1}^N \gamma_i^2 M_{i-1} + \sum_{t=1}^T \ell(s_t^N) \leq \sum_{t=1}^T \ell(0) + 4N\sqrt{T}$$

which implies that

$$\min_i M_i \leq \min_i M_{i-1} \leq \frac{\ln(2)}{\sum_i \gamma_i^2}T + \frac{4N}{\sum_i \gamma_i^2}\sqrt{T}$$

since $M_i \leq M_0$ for all i, $\ell(s_t^N) \geq 0$ for all t and $\ell(0) = \ln(2)$. Using this bound in inequality (3.10), we get

$$\sum_{t=1}^T \mathbf{1}\{y_t \neq \hat{y}_t\} \leq \frac{2\ln(2)T}{\sum_i \gamma_i^2} + \tilde{O}\left(\frac{N\sqrt{T}}{\sum_i \gamma_i^2} + \ln(N)\right) \leq \frac{2T}{\sum_i \gamma_i^2} + \tilde{O}\left(\frac{N^2}{\sum_i \gamma_i^2}\right),$$

where the last inequality follows from the bound $\frac{cN\sqrt{T}}{\sum_i \gamma_i^2} \leq \frac{T}{2\sum_i \gamma_i^2} + \frac{c^2N^2}{2\sum_i \gamma_i^2}$, where c is the hidden $\tilde{O}(1)$ factor in the $\tilde{O}(\frac{N\sqrt{T}}{\sum_i \gamma_i^2})$ term, using the arithmetic mean-geometric mean inequality. □

For the case when the weak learners do satisfy the bound (3.1), we get the following bound on the number of errors:

Theorem 12. *If the weak learners satisfy (3.1), then for any T and N, with high probability, the number of mistakes made by AdaBoost.OL is bounded by*

$$\frac{8T}{\gamma^2 N} + \tilde{O}\left(\frac{N}{\gamma^2} + \frac{S}{\gamma}\right),$$

Thus, in order to achieve error rate ϵ, it suffices to use $N \geq \frac{8}{\epsilon\gamma^2}$ weak learners, which gives an excess loss bound of $\tilde{O}(\frac{S}{\gamma} + \frac{1}{\epsilon\gamma^4})$.

Proof. The proof is along the same lines as that of Theorem 11. The only change is that in inequality (3.14), we use the bound $\gamma_i^2 \geq \frac{\gamma^2}{4} - \frac{\gamma\tilde{S}}{2\|\boldsymbol{w}^i\|_1}$ which follows from Lemma 6 using the fact that $a \geq b - c$ implies $a^2 \geq b^2 - 2bc$ for non-negative a, b and c, and the fact that $\|\boldsymbol{w}^i\|_\infty \leq 1$. This leads to the following change in inequality (3.15):

$$\min_{\alpha\in[-2,2]} \sum_{t=1}^{T} \left(\ell(s_t^{i-1} + \alpha z_t^i) - \ell(s_t^{i-1})\right) \leq -\frac{\gamma^2}{4} M_{i-1} + \gamma\tilde{S}.$$

Continuing using this bound in the proof and simplifying, we get the stated bound on the number of errors. □

The following lemma is a simple calculation:

Lemma 10. *For any $\sigma \in [0,1]$, $\min_{\alpha\in[-2,2]} \sigma e^{-\alpha} + (1-\sigma)e^{\alpha} \leq 1 - \frac{1}{2}(2\sigma - 1)^2$.*

Proof. It suffice to prove the bound for $\sigma \geq \frac{1}{2}$; the bound for $\sigma < \frac{1}{2}$ follows by simply using the bound for $1 - \sigma$. For $\sigma \in [0.5, 0.95]$, setting $\alpha = \frac{1}{2}\ln(\frac{\sigma}{1-\sigma}) \in [-2, 2]$ gives

$$\sigma e^{-\alpha} + (1-\sigma)e^{\alpha} = \sqrt{4\sigma(1-\sigma)} \leq 1 - \tfrac{1}{2}(2\sigma - 1)^2,$$

since $\sqrt{1-x} \leq 1 - \frac{1}{2}x$ for $x \in [0,1]$. For $\sigma \in (0.95, 1]$, setting $\alpha = \frac{1}{2}\ln(\frac{0.95}{0.05}) \in [-2, 2]$ we have $\sigma e^{-\alpha} + (1-\sigma)e^{\alpha} \leq 0.95e^{-\alpha} + 0.05e^{\alpha} = \sqrt{0.19} \leq \frac{1}{2} \leq 1 - \frac{1}{2}(2\sigma - 1)^2$. □

Although the number of weak learners and excess loss for Adaboost.OL are sub-optimal, the adaptivity of AdaBoost.OL is an appealing feature and leads to good performance in experiments. The possibility of obtaining an algorithm that is both adaptive and optimal is left as an open question.

3.1.5 Experiments

We perform experiments to evaluate our algorithms. We extended the Vowpal Wabbit open source machine learning system [105] to include the algorithms proposed here. We used VW's default base learning algorithm as our weak learner, tuning only the learning rate. The online boosting algorithms implemented were Online BBM, AdaBoost.OL, OSBoost (using uniform weighting on the weak learners) and OSBoost.OCP from [27], all using importance weighted examples in VW. We also implemented AdaBoost.OL.S, which is the version of AdaBoost.OL where examples sent to VW are sampled rather than weighted.

All experiments were done on a diverse collection of 13 publicly available datasets (see Appendix D for detailed descriptions). For each dataset, we performed a random split with 80% of the data used for training and the remaining 20% for testing. We tuned the learning rate, the number of weak learners, and the edge parameter γ (for all but AdaBoost.OL) using progressive validation 0-1 loss on the training set. Reported is the 0-1 loss on the test set.

It should be noted that the VW baseline is already a strong learner. The results obtained are given in Table 3.2. As can be seen, for most datasets, Online BBM had the best performance. The average improvement of Online BBM over the baseline was 5.14%. For AdaBoost.OL, it was 2.57%. Using sampling in AdaBoost.OL (i.e. AdaBoost.OL.S) boosts the average to 2.67%. The average improvement for OSBoost.OCP was 1.98%, followed by OSBoost with 1.13%.

Table 3.2: Experimental performance of various online boosting algorithms

Dataset	VW baseline	Online BBM	AdaBoost.OL	AdaBoost.OL.S	OSBoost.OCP	OSBoost
20news	0.0812	**0.0775**	0.0777	0.0777	0.0791	0.0801
a9a	0.1509	**0.1495**	0.1497	0.1497	0.1509	0.1505
activity	0.0133	**0.0114**	0.0128	0.0127	0.0130	0.0133
adult	0.1543	**0.1526**	0.1536	0.1536	0.1539	0.1544
bio	0.0035	**0.0031**	0.0032	0.0032	0.0033	0.0034
census	0.0471	**0.0469**	**0.0469**	**0.0469**	**0.0469**	0.0470
covtype	0.2563	**0.2347**	0.2495	0.2450	0.2470	0.2521
letter	0.2295	**0.1923**	0.2078	0.2078	0.2148	0.2150
maptaskcoref	0.1091	**0.1077**	0.1083	0.1083	0.1093	0.1091
nomao	0.0641	**0.0627**	0.0635	0.0635	**0.0627**	0.0633
poker	0.4555	**0.4312**	0.4555	0.4555	0.4555	0.4555
rcv1	0.0487	0.0485	**0.0484**	**0.0484**	0.0488	0.0488
vehv2binary	0.0292	0.0286	0.0291	0.0291	**0.0284**	0.0286

3.2 Online Boosting for General Regression

In this section we extend the online boosting theory to more general regression problem, which has been elusive and escaped theoretical guarantees thus far. Specifically we extend the very commonly used gradient boosting methods [41, 77] to the online setting, providing an online boosting algorithm that competes with any linear combination the base functions, given an online linear learning algorithm over the base class. This algorithm is the online analogue of the batch boosting algorithm of Zhang and Yu [111], and in fact our algorithmic technique, when specialized to the batch boosting setting, provides exponentially better convergence guarantees.

We also give an online boosting algorithm that competes with the best convex combination of base functions. This is a simpler algorithm which is analyzed along the lines of the Frank-Wolfe algorithm [35]. While the algorithm has weaker theoretical guarantees, it can still be useful in practice. We also prove that this algorithm obtains the optimal regret bound (up to constant factors) for this setting.

Related Work. The foundational theory of boosting for regression can be found in the statistics literature [51, 52], where boosting is understood as a greedy stagewise algorithm for fitting of additive models. The goal is to achieve the performance of linear combinations of base models, and to prove convergence to the performance of the best such linear combination.

While the earliest works on boosting for regression such as [41] do not have such convergence proofs, later works such as [77, 30] do have convergence proofs but without a bound on the speed of convergence. Bounds on the speed of convergence have been obtained by Duffy and Helmbold [34] relying on a somewhat strong assumption on the performance of the base learning algorithm. A different approach to boosting for regression was taken by Freund and Schapire [38], who give an algorithm that reduces the regression problem to classification and then applies AdaBoost; the corresponding proof of convergence relies on an assumption on the induced classification problem which may be hard to satisfy in practice. The strongest result is that of Zhang and Yu [111], who prove convergence to the performance of the best linear combination of base functions, along with a bound on the rate of convergence, making essentially no assumptions on the performance of the base learning algorithm. Telgarsky [103] proves similar results for logistic (or similar) loss using a slightly simpler boosting algorithm.

Our results are a generalization of the results of Zhang and Yu [111] to the online setting. However, we emphasize that this generalization is nontrivial and requires different algorithmic ideas and proof techniques. Indeed, we were not able to directly generalize the analysis in [111] by simply adapting the techniques used in Section 3.1, but we made use of the classical Frank-Wolfe algorithm [35]. On the other hand, while an important part of the convergence analysis for the batch setting is to show statistical consistency of the algorithms [111, 10, 103], in the online setting we only

need to study the empirical convergence (that is, the regret), which makes our analysis much more concise.

3.2.1 Problem Setup

For a regression problem, we assume that examples are chosen from a feature space $\mathcal{X}$, and the prediction space is $\mathbb{R}^d$. Let $\|\cdot\|$ denote some norm in $\mathbb{R}^d$. On each round t for $t = 1, 2, \ldots, T$, an adversary selects an example $\boldsymbol{x}_t \in \mathcal{X}$ and a loss function $\ell_t : \mathbb{R}^d \to \mathbb{R}$, and presents $\boldsymbol{x}_t$ to the online learner. The online learner outputs a prediction $\boldsymbol{y}_t \in \mathbb{R}^d$, obtains the loss function ℓ_t, and incurs loss $\ell_t(\boldsymbol{y}_t)$.

Let $\mathcal{F}$ denote a reference class of regression functions $f : \mathcal{X} \to \mathbb{R}^d$, and let $\mathbf{C}$ denote a class of loss functions $\ell : \mathbb{R}^d \to \mathbb{R}$. Also, let $R : \mathbb{N} \to \mathbb{R}_+$ be a non-decreasing function. We say that the function class $\mathcal{F}$ is *online learnable* for losses in $\mathbf{C}$ with regret R if there is an online learning algorithm $\mathcal{A}$, that for every $T \in \mathbb{N}$ and every sequence $(\boldsymbol{x}_t, \ell_t) \in \mathcal{X} \times \mathbf{C}$ for $t = 1, 2, \ldots, T$ chosen by the adversary, generates predictions[10] $\mathcal{A}(\boldsymbol{x}_t) \in \mathbb{R}^d$ such that

$$\sum_{t=1}^{T} \ell_t(\mathcal{A}(\boldsymbol{x}_t)) \;\leq\; \inf_{f \in \mathcal{F}} \sum_{t=1}^{T} \ell_t(f(\boldsymbol{x}_t)) + R(T). \tag{3.16}$$

If the online learning algorithm is randomized, we require the above bound to hold with high probability.

The above definition is simply the online generalization of standard empirical risk minimization (ERM) in the batch setting. A concrete example is 1-dimensional regression, i.e. the prediction space is $\mathbb{R}$. For a labeled data point $(\boldsymbol{x}, y^\star) \in \mathcal{X} \times \mathbb{R}$, the loss for the prediction $y \in \mathbb{R}$ is given by $\ell(y^\star, y)$ where $\ell(\cdot, \cdot)$ is a fixed loss function that is convex in the second argument (such as squared loss, logistic loss, etc). Given a batch of T labeled data points $\{(\boldsymbol{x}_t, y_t^\star) \mid t = 1, 2, \ldots, T\}$ and a base class of regression

[10]There is a slight abuse of notation here. $\mathcal{A}(\cdot)$ is not a function but rather the output of the online learning algorithm $\mathcal{A}$ computed on the given example using its internal state.

functions $\mathcal{F}$ (say, the set of bounded norm linear regressors), an ERM algorithm finds the function $f \in \mathcal{F}$ that minimizes $\sum_{t=1}^{T} \ell(y_t^\star, f(\boldsymbol{x}_t))$.

In the online setting, the adversary reveals the data $(\boldsymbol{x}_t, y_t^\star)$ in an online fashion, only presenting the true label $y_t^\star$ after the online learner $\mathcal{A}$ has chosen a prediction y_t. Thus, setting $\ell_t(y_t) = \ell(y_t^\star, y_t)$, we observe that if $\mathcal{A}$ satisfies the regret bound (3.16), then it makes predictions with total loss almost as small as that of the empirical risk minimizer, up to the regret term. If $\mathcal{F}$ is the set of all bounded-norm linear regressors, for example, the algorithm $\mathcal{A}$ could be online gradient descent [112] or online Newton Step [57].

At a high level, in the batch setting, "boosting" is understood as a procedure that, given a batch of data and access to an ERM algorithm for a function class $\mathcal{F}$ (this is called a "weak" learner), obtains an approximate ERM algorithm for a richer function class $\mathcal{F}'$ (this is called a "strong" learner). Generally, $\mathcal{F}'$ is the set of finite linear combinations of functions in $\mathcal{F}$. The efficiency of boosting is measured by how many times, N, the base ERM algorithm needs to be called (i.e., the number of boosting steps) to obtain an ERM algorithm for the richer function within the desired approximation tolerance. Convergence rates [111] give bounds on how quickly the approximation error goes to 0 and $N \to \infty$.

We now extend this notion of boosting to the online setting in the natural manner. To capture the full generality of the techniques, we also specify a class of loss functions that the online learning algorithm can work with. Informally, an online boosting algorithm is a reduction that, given access to an online learning algorithm $\mathcal{A}$ for a function class $\mathcal{F}$ and loss function class $\mathbf{C}$ with regret R, and a bound N on the total number of calls made in each iteration to copies of $\mathcal{A}$, obtains an online learning algorithm $\mathcal{A}'$ for a richer function class $\mathcal{F}'$, a richer loss function class $\mathbf{C}'$, and (possibly larger) regret R'. The bound N on the total number of calls made to all the copies of $\mathcal{A}$ corresponds to the number of boosting stages in the batch setting, and in the online

setting it may be viewed as a resource constraint on the algorithm. The efficacy of the reduction is measured by R' which is a function of R, N, and certain parameters of the comparator class $\mathcal{F}'$ and loss function class $\mathbf{C}'$. We desire online boosting algorithms such that $\frac{1}{T}R'(T) \to 0$ quickly as $N \to \infty$ and $T \to \infty$. We make the notions of richness in the above informal description more precise now.

Comparator function classes. A given function class $\mathcal{F}$ is said to be D-bounded if for all $\boldsymbol{x} \in \mathcal{X}$ and all $f \in \mathcal{F}$, we have $\|f(\boldsymbol{x})\| \leq D$. Throughout this chapter, we assume that $\mathcal{F}$ is closed under negation:[11] i.e. if $f \in \mathcal{F}$, then $-f \in \mathcal{F}$, and it contains the constant zero function, which we denote, with some abuse of notation, by $\mathbf{0}$.

Given $\mathcal{F}$, we define two richer function classes $\mathcal{F}'$: the convex hull of $\mathcal{F}$, denoted $\mathrm{CH}(\mathcal{F})$, is the set of convex combinations of a finite number of functions in $\mathcal{F}$, and the span of $\mathcal{F}$, denoted $\mathrm{span}(\mathcal{F})$, is the set of linear combinations of finitely many functions in $\mathcal{F}$. For any $f \in \mathrm{span}(\mathcal{F})$, define $\|f\|_1 := \inf\left\{\max\{1, \sum_{g\in S}|w_g|\} : \ f = \sum_{g\in S} w_g g,\ S \subseteq \mathcal{F},\ |S| < \infty,\ w_g \in \mathbb{R}\right\}$. Since functions in $\mathrm{span}(\mathcal{F})$ are not bounded, it is not possible to obtain a uniform regret bound for all functions in $\mathrm{span}(\mathcal{F})$: rather, the regret of an online learning algorithm $\mathcal{A}$ for $\mathrm{span}(\mathcal{F})$ is specified in terms of regret bounds for individual comparator functions $f \in \mathrm{span}(F)$, viz.

$$R_f(T) \ := \ \sum_{t=1}^{T} \ell_t(\mathcal{A}(\boldsymbol{x}_t)) - \sum_{t=1}^{T} \ell_t(f(\boldsymbol{x}_t)).$$

Loss function classes. The base loss function class we consider is $\mathcal{L}$, the set of all linear functions $\ell : \mathbb{R}^d \to \mathbb{R}$, with Lipschitz constant bounded by 1. A function class $\mathcal{F}$ that is online learnable with the loss function class $\mathcal{L}$ is called *online linear*

[11] This is without loss of generality; as will be seen momentarily, our base assumption only requires an online learning algorithm $\mathcal{A}$ for $\mathcal{F}$ for linear losses ℓ_t. By running the Hedge algorithm on two copies of $\mathcal{A}$, one of which receives the actual loss functions ℓ_t and the other recieves $-\ell_t$, we get an algorithm which competes with negations of functions in $\mathcal{F}$ and the constant zero function as well. Furthermore, since the loss functions are convex (indeed, linear) this can be made into a deterministic reduction by choosing the convex combination of the outputs of the two copies of $\mathcal{A}$ with mixing weights given by the Hedge algorithm.

learnable for short. The richer loss function class we consider is denoted by $\mathcal{C}$ and is a set of convex loss functions $\ell : \mathbb{R}^d \to \mathbb{R}$ satisfying some regularity conditions specified in terms of certain parameters described below.

We define a few parameters of the class $\mathcal{C}$. For any $b > 0$, let $\mathbb{B}^d(b) = \{\boldsymbol{y} \in \mathbb{R}^d : \|\boldsymbol{y}\| \leq b\}$ be the ball of radius b. The class $\mathcal{C}$ is said to have Lipschitz constant L_b on $\mathbb{B}^d(b)$ if for all $\ell \in \mathcal{C}$ and all $\boldsymbol{y} \in \mathbb{B}^d(b)$ there is an efficiently computable subgradient $\nabla\ell(\boldsymbol{y})$ with norm at most L_b. Next, $\mathcal{C}$ is said to be β_b-smooth on $\mathbb{B}^d(b)$ if for all $\ell \in \mathcal{C}$ and all $\boldsymbol{y}, \boldsymbol{y}' \in \mathbb{B}^d(b)$ we have

$$\ell(\boldsymbol{y}') \;\leq\; \ell(\boldsymbol{y}) + \nabla\ell(\boldsymbol{y}) \cdot (\boldsymbol{y}' - \boldsymbol{y}) + \frac{\beta_b}{2}\|\boldsymbol{y} - \boldsymbol{y}'\|^2.$$

Next, define the projection operator $\Pi_b : \mathbb{R}^d \to \mathbb{B}^d(b)$ as $\Pi_b(\boldsymbol{y}) := \arg\min_{\boldsymbol{y}' \in \mathbb{B}^d(b)} \|\boldsymbol{y} - \boldsymbol{y}'\|$, and define $\epsilon_b := \sup_{\boldsymbol{y} \in \mathbb{R}^d,\ \ell \in \mathcal{C}} \frac{\ell(\Pi_b(\boldsymbol{y})) - \ell(\boldsymbol{y})}{\|\Pi_b(\boldsymbol{y}) - \boldsymbol{y}\|}$.

3.2.2 Online Gradient Boosting

The setup is that we are given a D-bounded reference class of functions $\mathcal{F}$ with an online linear learning algorithm $\mathcal{A}$ with regret bound $R(\cdot)$. For normalization, we also assume that the output of $\mathcal{A}$ at any time is bounded in norm by D, i.e. $\|\mathcal{A}(\boldsymbol{x}_t)\| \leq D$ for all t. We further assume that for every $b > 0$, we can compute[12] a Lipschitz constant L_b, a smoothness parameter β_b, and the parameter ϵ_b for the class $\mathcal{C}$ over $\mathbb{B}^d(b)$. Furthermore, the online boosting algorithm may make up to N calls per iteration to any copies of $\mathcal{A}$ it maintains, for a given a budget parameter N.

Given this setup, our main result is an online boosting algorithm, Algorithm 9, competing with $\text{span}(\mathcal{F})$. The algorithm maintains N copies of $\mathcal{A}$, denoted $\mathcal{A}^i$, for $i = 1, 2, \ldots, N$. Each copy corresponds to one stage in boosting. When it receives a new example $\boldsymbol{x}_t$, it passes it to each $\mathcal{A}^i$ and obtains their predictions $\mathcal{A}^i(\boldsymbol{x}_t)$, which

[12] It suffices to compute upper bounds on these parameters.

Algorithm 9 Online Gradient Boosting for span($\mathcal{F}$)

Input: Number of weak learners N, step size parameter $\eta \in [\frac{1}{N}, 1]$,

1: Let $B = \min\{\eta N D,\ \inf\{b \geq D :\ \eta\beta_b b^2 \geq \epsilon_b D\}\}$.
2: Maintain N copies of the algorithm $\mathcal{A}$, denoted $\mathcal{A}^i$ for $i = 1, 2, \ldots, N$.
3: For each i, initialize $\sigma_1^i = 0$.
4: **for** $t = 1$ **to** T **do**
5: Receive example $\boldsymbol{x}_t$.
6: Define $\boldsymbol{y}_t^0 = \mathbf{0}$.
7: **for** $i = 1$ **to** N **do**
8: Define $\boldsymbol{y}_t^i = \Pi_B((1 - \sigma_t^i\eta)\boldsymbol{y}_t^{i-1} + \eta\mathcal{A}^i(\boldsymbol{x}_t))$.
9: **end for**
10: Predict $\boldsymbol{y}_t = \boldsymbol{y}_t^N$.
11: Obtain loss function ℓ_t and suffer loss $\ell_t(\boldsymbol{y}_t)$.
12: **for** $i = 1$ **to** N **do**
13: Pass loss function $\ell_t^i(\boldsymbol{y}) = \frac{1}{L_B}\nabla\ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}$ to $\mathcal{A}^i$.
14: Set $\sigma_{t+1}^i = \max\{\min\{\sigma_t^i + \alpha_t\nabla\ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}_t^{i-1}, 1\}, 0\}$, where $\alpha_t = \frac{1}{L_B B\sqrt{t}}$.
15: **end for**
16: **end for**

it then combines into a prediction for $\boldsymbol{y}_t$ using a linear combination. At the most basic level, this linear combination is simply the sum of all the predictions scaled by a step size parameter η. Two tweaks are made to this sum in step 8 to facilitate the analysis:

1. While constructing the sum, the partial sum $\boldsymbol{y}_t^{i-1}$ is multiplied by a *shrinkage* factor $(1 - \sigma_t^i\eta)$. This shrinkage term is tuned using an online gradient descent algorithm in step 14. The goal of the tuning is to induce the partial sums $\boldsymbol{y}_t^{i-1}$ to be aligned with a descent direction for the loss functions, as measured by the inner product $\nabla\ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}_t^{i-1}$.

2. The partial sums $\boldsymbol{y}_t^i$ are made to lie in $\mathbb{B}^d(B)$, for some parameter B, by using the projection operator Π_B. This is done to ensure that the Lipschitz constant and smoothness of the loss function are suitably bounded.

Once the boosting algorithm makes the prediction $\boldsymbol{y}_t$ and obtains the loss function ℓ_t, each $\mathcal{A}^i$ is updated using a suitably scaled linear approximation to the loss function

at the partial sum $\boldsymbol{y}_t^{i-1}$, i.e. the linear loss function $\frac{1}{L_B}\nabla\ell_t(\boldsymbol{y}_t^{i-1})\cdot\boldsymbol{y}$. This forces $\mathcal{A}^i$ to produce predictions that are aligned with a descent direction for the loss function.

We provide the analysis of the algorithm in Section 3.2.3. The analysis yields the following regret bound for the algorithm:

Theorem 13. *Let $\eta \in [\frac{1}{N}, 1]$ be a given parameter. Let $B = \min\{\eta ND,\ \inf\{b \geq D : \eta\beta_b b^2 \geq \epsilon_b D\}\}$. Algorithm 9 is an online learning algorithm for span($\mathcal{F}$) and losses in $\mathcal{C}$ with the following regret bound for any $f \in$ span($\mathcal{F}$):*

$$R'_f(T) \ \leq \ \left(1 - \frac{\eta}{\|f\|_1}\right)^N \Delta_0 + 3\eta\beta_B B^2\|f\|_1 T + L_B\|f\|_1 R(T) + 2L_B B\|f\|_1\sqrt{T},$$

where $\Delta_0 := \sum_{t=1}^T \ell_t(\mathbf{0}) - \ell_t(f(\boldsymbol{x}_t))$.

The regret bound in this theorem depends on several parameters such as B, β_B and L_B. In applications of the algorithm for 1-dimensional regression with commonly used loss functions, however, these parameters are essentially modest constants; see the end of this section for calculations of the parameters for various loss functions. Furthermore, if η is appropriately set (e.g. $\eta = (\log N)/N$), then the average regret $R'_f(T)/T$ clearly converges to 0 as $N \to \infty$ and $T \to \infty$. While the requirement that $N \to \infty$ may raise concerns about computational efficiency, this is in fact analogous to the guarantee in the batch setting: the algorithms converge only when the number of boosting stages goes to infinity. Moreover, our lower bound (Theorem 15) shows that this is indeed necessary.

We also present a simpler boosting algorithm, Algorithm 10, that competes with CH($\mathcal{F}$). Algorithm 10 is similar to Algorithm 9, with some simplifications: the final prediction is simply a convex combination of the predictions of the base learners, with no projections or shrinkage necessary. While Algorithm 9 is more general, Algorithm 10 may still be useful in practice when a bound on the norm of the comparator function is known in advance, using the observations in Section 3.2.4. Furthermore,

its analysis is cleaner and easier to understand for readers who are familiar with the Frank-Wolfe method, and this serves as a foundation for the analysis of Algorithm 9. This algorithm has an optimal (up to constant factors) regret bound as given in the following theorem, proven in Section 3.2.3. The upper bound in this theorem is proved along the lines of the Frank-Wolfe [35] algorithm, and the lower bound using information-theoretic arguments.

Theorem 14. *Algorithm 10 is an online learning algorithm for CH($\mathcal{F}$) for losses in $\mathcal{C}$ with the regret bound $R'(T) \leq \frac{8\beta_D D^2}{N} T + L_D R(T)$. Furthermore, the dependence of this regret bound on N is optimal up to constant factors.*

The dependence of the regret bound on $R(T)$ is unimprovable without additional assumptions: otherwise, Algorithm 10 will be an online linear learning algorithm over $\mathcal{F}$ with better than $R(T)$ regret.

Algorithm 10 Online Gradient Boosting for CH($\mathcal{F}$)

Maintain N copies of the algorithm $\mathcal{A}$, denoted $\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^N$, and let $\eta_i = \frac{2}{i+1}$ for $i = 1, 2, \ldots, N$.
for $t = 1$ **to** T **do**
    Receive example $\boldsymbol{x}_t$.
    Define $\boldsymbol{y}_t^0 = \boldsymbol{0}$.
    for $i = 1$ **to** N **do**
        Define $\boldsymbol{y}_t^i = (1 - \eta_i)\boldsymbol{y}_t^{i-1} + \eta_i \mathcal{A}^i(\boldsymbol{x}_t)$.
    end for
    Predict $\boldsymbol{y}_t = \boldsymbol{y}_t^N$.
    Obtain loss function ℓ_t and suffer loss $\ell_t(\boldsymbol{y}_t)$.
    for $i = 1$ **to** N **do**
        Pass loss function $\ell_t^i(\boldsymbol{y}) = \frac{1}{L_D} \nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}$ to $\mathcal{A}^i$.
    end for
end for

Using a deterministic base online linear learning algorithm. If the base online linear learning algorithm $\mathcal{A}$ is deterministic, then our results can be improved, because our online boosting algorithms are also deterministic, and using a standard

simple reduction, we can now allow $\mathcal{C}$ to be any set of convex functions (smooth or not) with a computable Lipschitz constant L_b over the domain $\mathbb{B}^d(b)$ for any $b > 0$.

This reduction converts arbitrary convex loss functions into linear functions: viz. if $\boldsymbol{y}_t$ is the output of the online boosting algorithm, then the loss function provided to the boosting algorithm as feedback is the linear function $\ell'_t(\boldsymbol{y}) = \nabla \ell_t(\boldsymbol{y}_t) \cdot \boldsymbol{y}$. This reduction immediately implies that the base online linear learning algorithm $\mathcal{A}$, when fed loss functions $\frac{1}{L_D}\ell'_t$, is already an online learning algorithm for $\mathrm{CH}(\mathcal{F})$ with losses in $\mathcal{C}$ with the regret bound $R'(T) \leq L_D R(T)$.

As for competing with $\mathrm{span}(\mathcal{F})$, since linear loss functions are 0-smooth, we obtain the following easy corollary of Theorem 13:

Corollary 3. *Let $\eta \in [\frac{1}{N}, 1]$ be a given parameter, and set $B = \eta N D$. Algorithm 9 is an online learning algorithm for span($\mathcal{F}$) for losses in $\mathcal{C}$ with the following regret bound for any $f \in$ span($\mathcal{F}$):*

$$R'_f(T) \;\leq\; \left(1 - \frac{\eta}{\|f\|_1}\right)^N \Delta_0 + L_B \|f\|_1 R(T) + 2 L_B B \|f\|_1 \sqrt{T},$$

where $\Delta_0 := \sum_{t=1}^T \ell_t(\mathbf{0}) - \ell_t(f(\boldsymbol{x}_t))$.

Parameters for several basic loss functions. To be specific, here we consider the application of our results to 1-dimensional regression, where we assume, for normalization, that the true labels of the examples and the predictions of the functions in the class $\mathcal{F}$ are in $[-1, 1]$. In this case $\|\cdot\|$ denotes the absolute value norm. Thus, in each round, the adversary chooses a labeled data point $(\boldsymbol{x}_t, y_t^\star) \in \mathcal{X} \times [-1, 1]$, and the loss for the prediction $y_t \in [-1, 1]$ is given by $\ell_t(y_t) = \ell(y_t^\star, y_t)$ where $\ell(\cdot, \cdot)$ is a fixed loss function that is convex in the second argument. Note that $D = 1$ in this setting. We give examples of several such loss functions below, and compute the parameters L_b, β_b and ϵ_b for every $b > 0$, as well as B from Theorem 13.

1. Linear loss: $\ell(y^\star, y) = -y^\star y$. We have $L_b = 1$, $\beta_b = 0$, $\epsilon_b = 1$, and $B = \eta N$.

2. p-norm loss, for some $p \geq 2$: $\ell(y^\star, y) = |y^\star - y|^p$. We have $L_b = p(b+1)^{p-1}$, $\beta_b = p(p-1)(b+1)^{p-2}$, $\epsilon_b = \max\{p(1-b)^{p-1}, 0\}$, and $B = 1$.

3. Modified least squares: $\ell(y^\star, y) = \frac{1}{2}\max\{1 - y^\star y, 0\}^2$. We have $L_b = b + 1$, $\beta_b = 1$, $\epsilon_b = \max\{1 - b, 0\}$, and $B = 1$.

4. Logistic loss: $\ell(y^\star, y) = \ln(1 + \exp(-y^\star y))$. We have $L_b = \frac{\exp(b)}{1+\exp(b)}$, $\beta_b = \frac{1}{4}$, $\epsilon_b = \frac{\exp(-b)}{1+\exp(-b)}$, and $B = \min\{\eta N, \ln(4/\eta)\}$.

3.2.3 Analysis

We first give the analysis of Algorithm 10 before that of Algorithm 9 since it is easier to understand and provides the foundation for the analysis of Algorithm 9.

Proof of Theorem 14. First, note that for any $i = 1, 2, \ldots, N$, since ℓ_t^i is a linear function, we have

$$\inf_{f \in \mathrm{CH}(\mathcal{F})} \sum_{t=1}^{T} \ell_t^i(f(\boldsymbol{x}_t)) \;=\; \inf_{f \in \mathcal{F}} \sum_{t=1}^{T} \ell_t^i(f(\boldsymbol{x}_t)).$$

Let f be any function in $\mathrm{CH}(\mathcal{F})$. The equality above and the fact that $\mathcal{A}^i$ is an online learning algorithm for $\mathcal{F}$ with regret bound $R(\cdot)$ for the 1-Lipschitz linear loss functions $\ell_t^i(\boldsymbol{y}) = \frac{1}{L_D}\nabla\ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}$ imply that

$$\sum_{t=1}^{T} \frac{1}{L_D}\nabla\ell_t(\boldsymbol{y}_t^{i-1}) \cdot \mathcal{A}^i(\boldsymbol{x}_t) \;\leq\; \sum_{t=1}^{T} \frac{1}{L_D}\nabla\ell_t(\boldsymbol{y}_t^{i-1}) \cdot f(\boldsymbol{x}_t) + R(T). \tag{3.17}$$

Now define, for $i = 0, 1, 2, \ldots, N$, $\Delta_i = \sum_{t=1}^{T} \ell_t(\boldsymbol{y}_t^i) - \ell_t(f(\boldsymbol{x}_t))$. We have

$$\begin{aligned}
\Delta_i &= \sum_{t=1}^{T} \ell_t(\boldsymbol{y}_t^{i-1} + \eta_i(\mathcal{A}^i(\boldsymbol{x}_t) - \boldsymbol{y}_t^{i-1})) - \ell_t(f(\boldsymbol{x}_t)) \\
&\leq \sum_{t=1}^{T} \ell_t(\boldsymbol{y}_t^{i-1}) - \ell_t(f(\boldsymbol{x}_t)) + \eta_i \nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot (\mathcal{A}^i(\boldsymbol{x}_t) - \boldsymbol{y}_t^{i-1}) + \frac{\eta_i^2 \beta_D}{2} \|\mathcal{A}^i(\boldsymbol{x}_t) - \boldsymbol{y}_t^{i-1}\|^2 \\
&\quad \text{(by } \beta_D\text{-smoothness of } \mathcal{C}) \\
&\leq \left[\sum_{t=1}^{T} \ell_t(\boldsymbol{y}_t^{i-1}) - \ell_t(f(\boldsymbol{x}_t)) + \eta_i \nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot (f(\boldsymbol{x}_t) - \boldsymbol{y}_t^{i-1}) + 2\eta_i^2 \beta_D D^2 \right] + \eta_i L_D R(T) \\
&\quad \text{(by (3.17) and using the bound } \|\mathcal{A}^i(\boldsymbol{x}_t) - \boldsymbol{y}_t^{i-1}\| \leq 2D) \\
&\leq \left[\sum_{t=1}^{T} \ell_t(\boldsymbol{y}_t^{i-1}) - \ell_t(f(\boldsymbol{x}_t)) - \eta_i(\ell_t(\boldsymbol{y}_t^{i-1}) - \ell_t(f(\boldsymbol{x}_t))) + 2\eta_i^2 \beta_D D^2 \right] + \eta_i L_D R(T) \\
&\quad \big(\text{by convexity, } \ell_t(\boldsymbol{y}_t^{i-1}) + \nabla \ell(\boldsymbol{y}_t^{i-1}) \cdot (f(\boldsymbol{x}_t) - \boldsymbol{y}_t^{i-1}) \leq \ell_t(f(\boldsymbol{x}_t))\big) \\
&= (1 - \eta_i)\Delta_{i-1} + 2\eta_i^2 \beta_D D^2 T + \eta_i L_D R(T).
\end{aligned}$$

For $i = 1$, since $\eta_1 = 1$, the above bound implies that $\Delta_1 \leq 2\beta_D D^2 T + L_D R(T)$. Starting from this base case, an easy induction on $i \geq 1$ proves that $\Delta_i \leq \frac{8\beta_D D^2}{i} T + L_D R(T)$. Applying this bound for $i = N$ completes the proof. □

We now show that the dependence of the regret bound of Algorithm 10 on the parameter N is optimal up to constant factors. Similarly to Section 3.1.3, here we define an online boosting algorithm as an algorithm whose prediction on each round depends only on the current example, the predictions of different copies of the base online linear learning algorithm and the entire history.

Theorem 15. *Let N be any specified bound on the total number of calls in each iteration to all copies of the base online linear learning algorithm. Then there is a setting of 1-dimensional prediction with a 1-bounded comparator function class $\mathcal{F}$, an online linear optimization algorithm $\mathcal{A}$ over $\mathcal{F}$, and a class $\mathcal{C}$ of loss functions that*

is 1*-smooth on* $\mathbb{R}$ *such that any online boosting algorithm for* CH($\mathcal{F}$) *with losses in* $\mathcal{C}$ *respecting the bound* N *has regret at least* $\Omega(\frac{T}{N})$.

Proof. Consider the following construction. At a high level, the setting is 1-dimensional regression with $\mathcal{C}$ corresponding to squared loss. The domain $\mathcal{X} = \mathbb{N}$ and true labels of examples are in $[0, 1]$.

Define $p_1 = \frac{1}{2} + \epsilon$ and $p_2 = \frac{1}{2} - \epsilon$, where $\epsilon = \frac{1}{10\sqrt{N}}$, and let D_1 and D_2 be two distributions over $\{0, 1\}^N$ where each bit is a Bernoulli random variable with parameter p_1 and p_2 respectively, chosen independently of the other bits. Consider a sequence of examples $(\boldsymbol{x}_t, y_t^\star) \in \mathbb{N} \times [0, 1]$ generated as follows: $\boldsymbol{x}_t = t$, and the label $y_t^\star$ is chosen from $\{p_1, p_2\}$ uniformly at random in each round.

Let $c = \frac{1}{4000}$. The function class $\mathcal{F}$ consists of a large number, $M = \frac{1}{c}N$, of functions f_i, $i \in [M]$. For each t and i, we set $f_i(\boldsymbol{x}_t) = 1$ w.p. $y_t^\star$, and 0 w.p. $1 - y_t^\star$, independently of all other values of t and i.

The base online linear learning algorithm $\mathcal{A}$ is simply Hedge over the M functions. In each round, the Hedge algorithm selects one of the M functions in $\mathcal{F}$ and uses that to predict the label, and for any sequence of T examples, with high probability, incurs regret $R(T) = O(\sqrt{\log(M)T})$.

We set $\mathcal{C}$ to be the set of squared loss functions, i.e. functions of the form $\ell(y) = \frac{1}{2}(y - y^\star)^2$ for $y^\star \in [0, 1]$. Note that these loss functions are 1-smooth and $D = 1$. In round t, the loss function is $\ell_t(y) = \frac{1}{2}(y - y_t^\star)^2$.

Consider the function $\bar{f} = \frac{1}{M}\sum_{i\in[M]} f_i$, which is in CH($\mathcal{F}$). Given any input sequence $(\boldsymbol{x}_t, y_t^\star)$ for $t = 1, 2, \ldots, T$ it is easy to calculate that $\mathbb{E}[\frac{1}{2}(\bar{f}(\boldsymbol{x}_t) - y_t^\star)^2] = \frac{y_t^\star(1-y_t^\star)}{2M} \leq \frac{1}{2M}$, and since the examples and predictions of functions on the examples are independent across iterations, a simple application of the multiplicative Chernoff bound implies that if $T \geq 12M$, then with probability at least 0.9, we have $\sum_{t=1}^{T} \frac{1}{2}(\bar{f}(\boldsymbol{x}_t) - y_t^\star)^2 \leq \frac{T}{M}$.

Now suppose there is an online boosting algorithm making at most N calls total to all copies of $\mathcal{A}$ in each iteration, that for any large enough T and for any sequence $(\boldsymbol{x}_t, y_t^\star)$ for $t = 1, 2, \ldots, T$, outputs predictions y_t such that with high probability, say at least 0.9, we have $\sum_{t=1}^T \frac{1}{2}(y_t - y_t^\star)^2 \leq \sum_{t=1}^T \frac{1}{2}(\bar{f}(\boldsymbol{x}_t) - y_t^\star)^2 + \frac{cT}{N}$. Then by a union bound, with probability at least 0.8, we have $\sum_{t=1}^T \frac{1}{2}(y_t - y_t^\star)^2 \leq \frac{cT}{N} + \frac{T}{M} \leq \frac{2cT}{N}$. By Markov's inequality and a union bound, with probability at least 0.7, for a uniform random time $\tau \in [T]$, we have

$$\frac{1}{2}(y_\tau - y_\tau^\star)^2 \leq \frac{20c}{N} = \frac{\epsilon^2}{2}, \tag{3.18}$$

or in other words, y_τ is on the same side of $\frac{1}{2}$ as $y_\tau^\star$, and thus can be used to identify $y_\tau^\star$. In the rest of the proof, we will use this fact, along with the fact that the total variation distance between D_1 and D_2, denoted $d_{\mathrm{TV}}(D_1, D_2)$, is small, to derive a contradiction.

Define the random variable $Y : \{0,1\}^N \to \mathbb{R}$ as follows. For any bit string $s = \langle s_1, s_2, \ldots, s_N \rangle \in \{0,1\}^N$, choose a random round $\tau \in [T]$, and simulate the online boosting process until round $\tau - 1$ by sampling $y_t^\star$'s and the outputs of $f_i(\boldsymbol{x}_t)$ for all $t \leq \tau - 1$ and $i \in [M]$ from the appropriate distributions. In round τ, let $f_{i_1}, f_{i_2}, \ldots, f_{i_N}$ be the functions that are obtained from the at most N calls to copies of $\mathcal{A}$ (there could be repetitions). Assign $f_{i_j}(\boldsymbol{x}_\tau) = s_j$ for $j \in [N]$ (being careful with repeated functions and repeating outputs appropriately), and run the booster with these outputs to obtain y_τ, and set $Y(s) = y_\tau$. Let $\Pr[\cdot]$ denote probability of events in this process for generating $Y(s)$ given s.

Let $\mathbb{E}_1[X(s)]$ and $\mathbb{E}_2[X(s)]$ denote expectation of a random variable $X : \{0,1\}^N \to \mathbb{R}$ when s is drawn from D_1 and D_2 respectively, and let $\mathbb{E}_0[X(I,s)]$ denote expectation of a random variable $X : \{1,2\} \times \{0,1\}^N \to \mathbb{R}$ when I is chosen from $\{1,2\}$ uniformly at random and then s is sampled from D_I. The above analysis (inequality (3.18))

implies that

$$0.7 \;\leq\; \mathbb{E}_0[\Pr[|Y(s) - p_I| \leq \epsilon]] \;=\; \tfrac{1}{2}\mathbb{E}_1[\Pr[|Y(s) - p_1| \leq \epsilon]] + \tfrac{1}{2}\mathbb{E}_2[\Pr[|Y(s) - p_2| \leq \epsilon]].$$

Now define a random variable $X : \{0,1\}^N \to \mathbb{R}$ as $X(s) = \Pr[Y(s) \geq \frac{1}{2}]$. Since

$$\Pr[Y(s) \geq \tfrac{1}{2}] \;\geq\; \Pr[|Y(s) - p_1| \leq \epsilon] \;\text{ and }\; 1 - \Pr[Y(s) \geq \tfrac{1}{2}] \;\geq\; \Pr[|Y(s) - p_2| \leq \epsilon],$$

we conclude, using the above bound, that $\mathbb{E}_1[X(s)] - \mathbb{E}_2[X(s)] \geq 0.4$. This is a contradiction, since because $X(s) \in [0,1]$, we have

$$\mathbb{E}_1[X(s)] - \mathbb{E}_2[X(s)] \;\leq\; d_{\mathrm{TV}}(D_1, D_2) \;<\; 4\sqrt{\epsilon^2 N} \;=\; 0.4,$$

where the bound on $d_{\mathrm{TV}}(D_1, D_2)$ is standard, for e.g. see [55]. This gives us the desired contradiction. □

The above result can be easily extended to any given parameters β and D so that the $\mathcal{F}$ is D-bounded and $\mathcal{C}$ is β-smooth on $\mathbb{R}$, giving a lower bound of $\Omega(\frac{\beta D^2 T}{N})$ on the regret of an online boosting algorithm for $\mathrm{CH}(\mathcal{F})$ with losses in $\mathcal{C}$: we simply scale all function and label values by D, and consider the loss functions $\ell(y, y^\star) = \frac{\beta}{2}(y - y^\star)^2$. If there were an online boosting algorithm for $\mathrm{CH}(\mathcal{F})$ with these loss functions with regret $o(\frac{\beta D^2 T}{N})$, then by scaling down the predictions by D, we obtain an online boosting algorithm for exactly the setting in the proof of Theorem 15 with a regret bound of $o(\frac{T}{N})$, which is a contradiction.

Next we show that Algorithm 9 satisfies the regret bound claimed in Theorem 13.

Proof of Theorem 13. Let $f = \sum_{g \in S} w_g g$, for some finite subset S of $\mathcal{F}$, where $w_g \in \mathbb{R}$. Since $\mathcal{F}$ is symmetric, we may assume that all $w_g \geq 0$, and let $W := \sum_g w_g$. Furthermore, we may assume that $\mathbf{0} \in S$ with weight $w_{\mathbf{0}} = \max\{1 - \sum_{g \in S,\; g \neq \mathbf{0}} w_g, 0\}$, so that $W \geq 1$. Note that $\|f\|_1$ is exactly the infimum of W over all such ways of

expressing f as a finite weighted sum of functions in $\mathcal{F}$. We now prove that bound stated in the theorem holds with $\|f\|_1$ replaced by W; the theorem then follows simply by taking the infimum of the bound over all such ways of expressing f.

Now, for each $i \in [N]$, the update in line 14 of Algorithm 9 is exactly online gradient descent [112] on the domain $[0, 1]$ with linear loss functions $\sigma \mapsto -\nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}_t^{i-1}\sigma$. Note that the derivative of this loss function is bounded as follows:

$$| - \nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}_t^{i-1}| \leq L_B B.$$

Since $\frac{1}{W} \in [0, 1]$, the standard analysis of online gradient descent then implies that the sequence σ_t^i for $t = 1, 2, \ldots, T$ satisfies

$$\sum_{t=1}^{T} -\nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}_t^{i-1}\sigma_t^i \;\leq\; \sum_{t=1}^{T} -\nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}_t^{i-1}\frac{1}{W} + 2L_B B\sqrt{T}. \tag{3.19}$$

Next, since $f = \sum_{g \in S} w_g g$ with $w_g \geq 0$, we have

$$\frac{1}{W}\sum_{t=1}^{T} \nabla \ell_t(\boldsymbol{y}_t^i) \cdot f(\boldsymbol{x}_t) = \frac{1}{\sum_{g \in S} w_g}\sum_{t=1}^{T}\sum_{g \in S} w_g \nabla \ell_t(\boldsymbol{y}_t^i) \cdot g(\boldsymbol{x}_t) \;\geq\; \min_{g \in S}\sum_{t=1}^{T} \nabla \ell_t(\boldsymbol{y}_t^i) \cdot g(\boldsymbol{x}_t). \tag{3.20}$$

Let $g^\star \in \arg\min_{g \in S} \sum_{t=1}^{T} \nabla \ell_t(\boldsymbol{y}_t^i) \cdot g(\boldsymbol{x}_t)$. Since $\mathcal{A}^i$ is an online learning algorithm for $\mathcal{F}$ with regret bound $R(\cdot)$ for the 1-Lipschitz linear loss functions $\ell_t^i(\boldsymbol{y}) = \frac{1}{L_B}\nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot \boldsymbol{y}$, and $g^\star \in \mathcal{F}$, multiplying the regret bound (3.16) by L_B we have

$$\begin{aligned}\sum_{t=1}^{T} \nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot \mathcal{A}^i(\boldsymbol{x}_t) &\leq \sum_{t=1}^{T} \nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot g^\star(\boldsymbol{x}_t) + L_B R(T) \\ &\leq \frac{1}{W}\sum_{t=1}^{T} \nabla \ell_t(\boldsymbol{y}_t^{i-1}) \cdot f(\boldsymbol{x}_t) + L_B R(T)\end{aligned} \tag{3.21}$$

by (3.20). Now, we analyze how much excess loss is potentially introduced due to the projection in line 8. First, note that if $B = \eta N D$, then the projection has no

effect since $(1-\sigma_t^i\eta)\boldsymbol{y}_t^{i-1}+\eta\mathcal{A}^i(\boldsymbol{x}_t)\in\mathbb{B}^d(B)$, and in this case $\ell_t(\boldsymbol{y}_t^i) = \ell_t((1-\sigma_t^i\eta)\boldsymbol{y}_t^{i-1}+\eta\mathcal{A}^i(\boldsymbol{x}_t))$. If $B<\eta ND$, then by the definition of B, $\eta\beta_B B^2\geq\epsilon_B D$, and since $(1-\sigma_t^i\eta)\boldsymbol{y}_t^{i-1}\in\mathbb{B}^d(B)$ and $\|\eta\mathcal{A}^i(\boldsymbol{x}_t))\|\leq\eta D$, and we have

$$\ell_t(\boldsymbol{y}_t^i) \;=\; \ell_t(\Pi_B((1-\sigma_t^i\eta)\boldsymbol{y}_t^{i-1}+\eta\mathcal{A}^i(\boldsymbol{x}_t))) \;\leq\; \ell_t((1-\sigma_t^i\eta)\boldsymbol{y}_t^{i-1}+\eta\mathcal{A}^i(\boldsymbol{x}_t))+\eta\epsilon_B D.$$

In either case, we have

$$\ell_t(\boldsymbol{y}_t^i) \;\leq\; \ell_t((1-\sigma_t^i\eta)\boldsymbol{y}_t^{i-1}+\eta\mathcal{A}^i(\boldsymbol{x}_t))+\eta^2\beta_B B^2. \tag{3.22}$$

We now move to the main part of the analysis. Define for $i=0,1,2,\ldots,N$, $\Delta_i:=\sum_{t=1}^T\ell_t(\boldsymbol{y}_t^i)-\ell_t(f(\boldsymbol{x}_t))$. We bound Δ_i as follows:

$$\begin{aligned}
&\left[\sum_{t=1}^T\ell_t((1-\sigma_t^i\eta)\boldsymbol{y}_t^{i-1}+\eta\mathcal{A}^i(\boldsymbol{x}_t))-\ell_t(f(\boldsymbol{x}_t))\right]+\eta^2\beta_B B^2T\\
&\leq\Delta_{i-1}+\left[\sum_{t=1}^T\eta\nabla\ell_t(\boldsymbol{y}_t^{i-1})\cdot(\mathcal{A}^i(\boldsymbol{x}_t)-\sigma_t^i\boldsymbol{y}_t^{i-1})+\frac{\beta_B\eta^2}{2}\|\mathcal{A}^i(\boldsymbol{x}_t)-\sigma_t^i\boldsymbol{y}_t^{i-1}\|^2\right]+\eta^2\beta_B B^2T\\
&\quad\text{(by }\beta_B\text{-smoothness)}\\
&\leq\Delta_{i-1}\;+\left[\sum_{t=1}^T\frac{\eta}{W}\nabla\ell_t(\boldsymbol{y}_t^{i-1})\cdot(f(\boldsymbol{x}_t)-\boldsymbol{y}_t^{i-1})\right]+3\eta^2\beta_B B^2T+\eta L_B R(T)+2\eta L_B B\sqrt{T}\\
&\quad\text{(by (3.19), (3.21) and the fact that }\|\mathcal{A}^i(\boldsymbol{x}_t)-\sigma_t^i\boldsymbol{y}_t^{i-1}\|\leq D+B\leq 2B)\\
&\leq\;\left(1-\frac{\eta}{W}\right)\Delta_{i-1}+3\eta^2\beta_B B^2T+\eta L_B R(T)+2\eta L_B B\sqrt{T},
\end{aligned}$$

since, by convexity of ℓ_t we have $\ell_t(\boldsymbol{y}_t^{i-1})+\nabla\ell(\boldsymbol{y}_t^{i-1})\cdot(f(\boldsymbol{x}_t)-\boldsymbol{y}_t^{i-1})\leq\ell_t(f(\boldsymbol{x}_t))$. Applying the above bound iteratively, we get

$$\begin{aligned}
\Delta_N \;&\leq\; \left(1-\frac{\eta}{W}\right)^N\Delta_0+\sum_{i=1}^N\left(1-\frac{\eta}{W}\right)^{i-1}\cdot(3\eta^2\beta_B B^2T+\eta L_B R(T)+2\eta L_B B\sqrt{T})\\
&\leq\; \left(1-\frac{\eta}{W}\right)^N\Delta_0+3\eta\beta_B B^2WT+L_B WR(T)+2L_B BW\sqrt{T}.
\end{aligned}$$

This completes the proof. □

3.2.4 Variants and Generalizations

Our boosting algorithms and the analysis are considerably flexible: it is easy to modify the algorithms to work with a different (and perhaps more natural) kind of base learner which does greedy fitting, or incorporate a scaling of the base functions which improves performance. Also, when specialized to the batch setting, our algorithms provide better convergence rates than previous work.

Fitting to actual loss functions. The choice of an online *linear* learning algorithm over the base function class in our algorithms was made to ease the analysis. In practice, it is more common to have an online algorithm which produces predictions with comparable accuracy to the best function in hindsight for the *actual* sequence of loss functions. In particular, a common heuristic in boosting algorithms such as the original gradient boosting algorithm by Friedman [41] or the matching pursuit algorithm of Mallat and Zhang [76] is to build a linear combination of base functions by iteratively augmenting the current linear combination by greedily choosing a base function and a step size for it that minimizes the loss with respect to the residual label. Indeed, the boosting algorithm of Zhang and Yu [111] also uses this kind of greedy fitting algorithm as the base learner.

In the online setting, we can model greedy fitting as follows. We first fix a step size $\alpha \geq 0$ in advance. Then, in each round t, the base learner $\mathcal{A}$ receives not only the example $\boldsymbol{x}_t$, but also an *offset* $\boldsymbol{y}'_t \in \mathbb{R}^d$ for the prediction, and produces a prediction $\mathcal{A}(\boldsymbol{x}_t) \in \mathbb{R}^d$, after which it receives the loss function ℓ_t and suffers loss $\ell_t(\boldsymbol{y}'_t + \alpha\mathcal{A}(\boldsymbol{x}_t))$. The predictions of $\mathcal{A}$ satisfy

$$\sum_{t=1}^{T} \ell_t(\boldsymbol{y}'_t + \alpha\mathcal{A}(\boldsymbol{x}_t)) \;\leq\; \inf_{f \in \mathcal{F}} \sum_{t=1}^{T} \ell_t(\boldsymbol{y}'_t + \alpha f(\boldsymbol{x}_t)) + R(T),$$

where R is the regret. Our algorithms can be made to work with this kind of base learner as well. The details can be found in Appendix B.2.1.

Improving the regret bound via scaling. Given an online linear learning algorithm $\mathcal{A}$ over the function class $\mathcal{F}$ with regret R, then for any scaling parameter $\lambda > 0$, we trivially obtain an online linear learning algorithm, denoted $\lambda\mathcal{A}$, over a λ-scaling of $\mathcal{F}$, viz. $\lambda\mathcal{F} := \{\lambda f \mid f \in \mathcal{F}\}$, simply by multiplying the predictions of $\mathcal{A}$ by λ. The corresponding regret scales by λ as well, i.e. it becomes λR.

The performance of Algorithm 9 can be improved by using such an online linear learning algorithm over $\lambda\mathcal{F}$ for a suitably chosen scaling $\lambda \geq 1$ of the function class $\mathcal{F}$. The regret bound from Theorem 13 improves because the 1-norm of f measured with respect to $\lambda\mathcal{F}$, i.e. $\|f\|_1' = \max\{1, \frac{\|f\|_1}{\lambda}\}$, is smaller than $\|f\|_1$, but degrades because the parameter $B' = \min\{\eta N \lambda D,\ \inf\{b \geq \lambda D :\ \eta\beta_b b^2 \geq \epsilon_b \lambda D\}\}$ is larger than B. But, as detailed in Appendix B.2.2, in many situations the improvement due to the former compensates for the degradation due to the latter, and overall we can get improved regret bounds using a suitable value of λ.

Improvements for batch boosting Our algorithmic technique can be easily specialized and modified to the standard batch setting with a fixed batch of training examples and a base learning algorithm operating over the batch, exactly as in [111]. The main difference compared to the algorithm of [111] is the use of the σ variables to scale the coefficients of the weak hypotheses appropriately. While a seemingly innocuous tweak, this allows us to derive analogous bounds to those of Zhang and Yu [111] on the optimization error that show that our boosting algorithm converges exponentially faster. A detailed comparison can be found in Appendix B.2.3.

Table 3.3: Improvement of Online Gradient Boosting over different base learners

Base learner	Average improvement		Median improvement	
	Algorithm 9	Algorithm 10	Algorithm 9	Algorithm 10
VW	1.65%	1.33%	0.03%	0.29%
Regression stumps	20.22%	15.90%	10.45%	13.69%
Neural networks	7.88%	0.72%	0.72%	0.33%

3.2.5 Experiments

We again implement and evaluate Algorithms 9 and 10 within the Vowpal Wabbit (VW) open source machine learning system [105]. The three online base learners used were VW's default linear learner (a variant of stochastic gradient descent), two-layer sigmoidal neural networks with 10 hidden units, and regression stumps.

Regression stumps were implemented by doing stochastic gradient descent on each individual feature, and predicting with the best-performing non-zero valued feature in the current example.

All experiments were done on a collection of 14 publically available regression and classification datasets (described in Appendix D) using squared loss. The only parameters tuned were the learning rate and the number of weak learners, as well as the step size parameter for Algorithm 9. Parameters were tuned based on progressive validation loss on half of the dataset; reported is progressive validation loss on the remaining half. Progressive validation is a standard online validation technique, where each training example is used for testing before it is used for updating the model [17].

Table 3.3 reports the average and the median, over the datasets, relative improvement in squared loss over the respective base learner. One can see that boosting is much more effective for weaker base learners such as regression stumps, while still achieves reasonable improvement for stronger base learners such as neural networks and VW default learner.

3.3 Conclusion and Future Directions

In this chapter, we developed a theory of online boosting, the task of improving the accuracy of an existing online learning algorithm by training and combining several copies of it appropriately. Our results are based on a natural weak learning assumption that generalizes the one in the batch setting directly. We proposed two online boosting algorithms and showed that one of them is optimal in terms of the number of weak learners and the number of examples to achieve a specific error, while the other is suboptimal but adaptive in the sense that it is parameter-free and able to exploit the fact that some weak learners might be doing better than others. One obvious yet difficult open question here is whether we can get both optimality and adaptivity at the same time.

In the second part of this chapter we also generalized the theory of boosting for regression problems to the online setting and provided online gradient boosting algorithms with theoretical convergence guarantees. Our algorithmic technique also improves convergence guarantees for batch boosting algorithms. We also provided experimental evidence that our boosting algorithms do improve prediction accuracy over commonly used base learners in practice, with greater improvements for weaker base learners. The main remaining open question here is whether the algorithm we propose for competing with the span of the base functions is optimal in any sense, similar to our proof of optimality for the algorithm for competing with the convex hull of the base functions.

Chapter 4

Efficient Second Order Online Learning via Sketching

In this chapter we study efficient invariant online learning. The performance of classic learning algorithms (such as gradient descent) usually depends on how the data is represented, for instance, whether length is measured in meters or centimeters. In the offline setting where all the data is available in advance, one can simply perform preprocessing (such as whitening) on the data before feeding it to the algorithm, so that the way the data is represented does not affect the algorithm anymore. Since this is only done once before training happens, the overhead is negligible.

In the online setting, however, examples arrive one by one on the fly and data preprocessing is no longer an option. The main question is thus whether in this case we can still have an efficient algorithm whose performance is independent of how the data is represented.

To address this problem, we thus propose an online learning algorithm that is invariant to linear transformation and as a result enjoy a regret guarantee that is independent of the representation of the data. Our algorithm is an adaptive version of Online Newton Step [57] which makes use of second order information. To over-

come the quadratic running time of the algorithm, we combine different sketching techniques with our algorithm to achieve running time that is linear in the number of dimension, which is then even improved to be linear in the sparsity of the data. Moreover, we rigorously study the regret guarantee when using some of the sketching techniques and show that, roughly speaking, the regret is only slightly affected as long as the size of the sketch is larger than the number of significant directions of the data.

This chapter includes joint work with Alekh Agarwal, Nicolò Cesa-Bianchi and John Langford [75].

4.1 Motivation and Main Results

Online learning methods are highly successful at rapidly reducing the test error on large, high-dimensional datasets. First-order methods are particularly attractive in such problems as they typically enjoy computational complexity linear in the input size. However, the convergence of these methods crucially depends on the representation of the data; for instance, running the same algorithm on the same dataset but with different feature representation (e.g. using meter instead of centimeter to measure length) can return vastly inferior results.

One approach to address this problem is to require that the algorithm is invariant to linear transformation, meaning that if an unknown linear transformation is applied to all the data in advance, the algorithm would make the exact same predictions as it would have given the original data. If an algorithm has this invariance property, then we can simply pretend that the algorithm is run on the best represented dataset obtained by preprocessing the original data as if we had the entire data beforehand.

Second order algorithms such as Online Newton Step [57] have this invariant property, but typically require update time which is quadratic in the number of

dimensions. Furthermore, the dependence on dimension is not improved even if the examples are sparse. These issues lead to the key question in our work: *How do we develop (approximately) invariant online learning algorithms with efficient updates?* More concretely, we want algorithms with update cost linear in input sparsity like first-order methods (albeit with a larger constant).

To improve the computational efficiency, we study how sketching techniques can be combined with Online Newton Step and reduce its running time to be linear. While the idea of data sketching is widely studied [110], as far as we know our work is the first one to apply it to the adversarial online learning setting providing rigorous regret guarantees. Three different sketching methods are considered: Random Projections (RP) [61, 3], Frequent Directions [69, 46], and Oja's algorithm [85, 86], all of which allow linear running time per round. For the first two methods, we prove regret bounds similar to the one of the original Online Newton Step whenever the sketch-size is large enough. For example, we show that the regret using the RP sketch is at most $\tilde{O}(\sqrt{Td})$ for convex losses and $\tilde{O}(d \ln T)$ with an additional curvature assumption, where T is the number of examples and d is the number of dimensions. We show that this optimizes a regret bound even in hindsight up to log factors, with a computational overhead of $O(r + \ln T)$ if the data is of rank r. Our analysis makes it easy to plug in other sketching and online PCA methods (e.g. [44]).

For practical implementations, in Section 4.5 we further develop sparse versions of these updates with a running time linear in the sparsity of the examples instead of the dimensions. We believe that these are the first known sparse implementations of the Frequent Directions and Oja's algorithm, and may be of independent interest. Overall, our work provides the first proven approximately invariant and efficient algorithms for online learning.

Related work Our online learning setting is closest to the one proposed in [94], which studies scale-invariant algorithms, a special case of the invariance property considered here (see also [89, Section 5]). Computational efficiency, a main concern in this work, is not a problem there since each coordinate is scaled independently. Orabona and Pál [88] study unrelated notions of invariance.

Another line of work focuses on reducing the effect of how data is represented in stochastic optimization, where the examples are drawn i.i.d. from a distribution. The L-BFGS algorithm [70] has recently been studied in the stochastic setting [99, 21, 81, 102, 82], but the analysis relies on assumptions of smoothness and strong convexity with pessimistic rates in theory and reliance on the use of large mini-batches empirically. Recent work by Gonen and Shalev-Shwartz [47] considers a random projection sketching approach to stochastic optimization, but does not consider other sketches or sparse implementation, and does not extend in an obvious manner to the online setting.

The Frank-Wolfe algorithm [35, 62] is also invariant to linear transformations, but the online regret bounds scale as $O(T^{2/3})$ as opposed to the typical $O(\sqrt{T})$ rates [54]. Garber and Hazan [43] improve the regret to $O(\sqrt{T})$ but with a much more complicated variants.

Notation We introduce some matrix notation. We use $\text{diag}\{\boldsymbol{x}\}$ to represent a diagonal matrix with $\boldsymbol{x}$ on the diagonal, $\lambda_i(A)$ to denote the i-th largest eigenvalue of A, $\|\boldsymbol{w}\|_A$ to denote $\sqrt{\boldsymbol{w}^\top A \boldsymbol{w}}$. The determinant and trace of A are represented by $|A|$ and $\text{TR}(A)$ respectively. Finally, $\langle A, B \rangle$ denotes $\sum_{i,j} A_{i,j} B_{i,j}$, and $A \preceq B$ means that $B - A$ is positive semidefinite.

4.2 Setup and a Meta Algorithm

We consider a general online linear prediction setting. On each round $t = 1, 2 \ldots$

1. the adversary first presents an example $\boldsymbol{x}_t \in \mathbb{R}^d$,

2. the learner chooses $\boldsymbol{w}_t \in \mathbb{R}^d$ and predicts $\boldsymbol{w}_t^\top \boldsymbol{x}_t$,

3. the adversary reveals a loss $f_t(\boldsymbol{w}) = \ell_t(\boldsymbol{w}^\top \boldsymbol{x}_t)$ for some convex differentiable $\ell_t : \mathbb{R} \to \mathbb{R}_+$,

4. the learner suffers loss $f_t(\boldsymbol{w}_t)$ for this round.

As in previous sections, the regret of the learner with respect to a fixed comparator $\boldsymbol{w}$ is defined as

$$R_T(\boldsymbol{w}) = \sum_{t=1}^{T} f_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} f_t(\boldsymbol{w}) \ .$$

Typical results study $R_T(\boldsymbol{w})$ against all $\boldsymbol{w}$ with a bounded norm in some geometry. For an invariant update, we relax this requirement and only put bounds on the predictions $\boldsymbol{w}^\top \boldsymbol{x}_t$. Specifically, we define for some pre-chosen positive constant C,

$$\mathcal{K}_t \stackrel{\text{def}}{=} \left\{ \boldsymbol{w} \ : \ |\boldsymbol{w}^\top \boldsymbol{x}_t| \le C \right\} \ . \tag{4.1}$$

We seek to minimize regret to all comparators that generate bounded predictions on every data point, i.e.

$$R_T = \sup_{\boldsymbol{w} \in \mathcal{K}} R_T(\boldsymbol{w}), \quad \mathcal{K} \stackrel{\text{def}}{=} \bigcap_{t=1}^{T} \mathcal{K}_t \ .$$

Under this setup, if the data were transformed to $M\boldsymbol{x}_t$ for all t and some invertible matrix $M \in \mathbb{R}^{d \times d}$, the optimal $\boldsymbol{w}^*$ would simply move to $(M^{-1})^\top \boldsymbol{w}^*$, ensuring the same predictions. This is exactly the key property we are looking for for invariant learning.

We make two structural assumptions on the loss.

Assumption 1. *(Scalar Lipschitz) The loss function ℓ_t satisfies $|\ell_t'(z)| \le L$ whenever $|z| \le C$.*

Assumption 2. *(Curvature) There exists $\sigma_t \geq 0$ such that for all $\boldsymbol{u}, \boldsymbol{w} \in \mathcal{K}$,*

$$f_t(\boldsymbol{w}) \geq f_t(\boldsymbol{u}) + \nabla f_t(\boldsymbol{u})^\top (\boldsymbol{w} - \boldsymbol{u}) + \frac{\sigma_t}{2} \left(\nabla f_t(\boldsymbol{u})^\top (\boldsymbol{u} - \boldsymbol{w})\right)^2 .$$

When $\sigma_t = 0$, this assumption merely imposes convexity. More generally, it is satisfied by squared loss $f_t(\boldsymbol{w}) = (\boldsymbol{w}^\top \boldsymbol{x}_t - y_t)^2$ with $\sigma_t = \frac{1}{8C^2}$ whenever $|\boldsymbol{w}^\top \boldsymbol{x}_t|$ and $|y_t|$ are bounded by C, as well as for all exp-concave functions (see Hazan et al. [57, Lemma 3]).

To solve the problem, we consider a broad family of updates that generalizes the well-known gradient descent method and is similar to the Online Newton Step of Hazan et al. [57]. At round $t+1$ with some invertible matrix A_t (that will be specified later) and gradient $\boldsymbol{g}_t = \nabla f_t(\boldsymbol{w}_t)$, the algorithm performs the following update *before* making the prediction on the new data point $\boldsymbol{x}_{t+1}$,

$$\begin{aligned} \boldsymbol{u}_{t+1} &= \boldsymbol{w}_t - A_t^{-1} \boldsymbol{g}_t \\ \boldsymbol{w}_{t+1} &= \operatorname*{argmin}_{\boldsymbol{w} \in \mathcal{K}_{t+1}} \|\boldsymbol{w} - \boldsymbol{u}_{t+1}\|_{A_t} . \end{aligned} \tag{4.2}$$

The projection onto the set $\mathcal{K}_{t+1}$ (defined in (4.1)) differs from typical norm-based projections as it only enforces boundedness on $\boldsymbol{x}_{t+1}$ at round $t+1$.

One can think of A_t as a generalized "learning rate". If A_t is a scaled identity matrix, then this simply degenerates to gradient descent. If A_t is some diagonal matrix, updates similar to those of Ross et al. [94] are recovered. We will later see how this generalization helps solve the problem. Different matrices A_t result in different methods in the following sections, all of which satisfy the following regret bound.

Proposition 1. *For any sequence of positive definite matrices A_t and sequence of losses satisfying Assumptions 1 and 2, the regret of updates* (4.2) *against any com-*

parator $\boldsymbol{w} \in \mathcal{K}$ *satisfies*

$$2R_T(\boldsymbol{w}) \leq \|\boldsymbol{w}\|_{A_0}^2 + \underbrace{\sum_{t=1}^{T} \boldsymbol{g}_t^T A_t^{-1} \boldsymbol{g}_t}_{\text{"Gradient Bound" } R_G} + \underbrace{\sum_{t=1}^{T} (\boldsymbol{w}_t - \boldsymbol{w})^\top (A_t - A_{t-1} - \sigma_t \boldsymbol{g}_t \boldsymbol{g}_t^\top)(\boldsymbol{w}_t - \boldsymbol{w})}_{\text{"Diameter Bound" } R_D} \tag{4.3}$$

Proof. Since $\boldsymbol{w}_{t+1}$ is the projection of $\boldsymbol{u}_{t+1}$ onto $\mathcal{K}_{t+1}$, by the property of projections (see for example [54, Lemma 8]), the algorithm ensures

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}\|_{A_t}^2 \leq \|\boldsymbol{u}_{t+1} - \boldsymbol{w}\|_{A_t}^2 = \|\boldsymbol{w}_t - \boldsymbol{w}\|_{A_t}^2 + \boldsymbol{g}_t^\top A_t^{-1} \boldsymbol{g}_t - 2\boldsymbol{g}_t^\top (\boldsymbol{w}_t - \boldsymbol{w})$$

for all $\boldsymbol{w} \in \mathcal{K} \subseteq \mathcal{K}_{t+1}$. By the curvature property in Assumption 1, we then have that

$$\begin{aligned} 2R_T(\boldsymbol{w}) &\leq \sum_{t=1}^{T} 2\boldsymbol{g}_t^\top (\boldsymbol{w}_t - \boldsymbol{w}) - \sigma_t \big(\boldsymbol{g}_t^\top (\boldsymbol{w}_t - \boldsymbol{w})\big)^2 \\ &\leq \sum_{t=1}^{T} \boldsymbol{g}_t^\top A_t^{-1} \boldsymbol{g}_t + \|\boldsymbol{w}_t - \boldsymbol{w}\|_{A_t}^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}\|_{A_t}^2 - \sigma_t \big(\boldsymbol{g}_t^\top (\boldsymbol{w}_t - \boldsymbol{w})\big)^2 \\ &= \|\boldsymbol{w}\|_{A_0}^2 + \sum_{t=1}^{T} \boldsymbol{g}_t^\top A_t^{-1} \boldsymbol{g}_t + (\boldsymbol{w}_t - \boldsymbol{w})^\top (A_t - A_{t-1} - \sigma_t \boldsymbol{g}_t \boldsymbol{g}_t^\top)(\boldsymbol{w}_t - \boldsymbol{w}), \end{aligned}$$

which completes the proof. □

The form of regret bound above is standard in online learning, and is invoked below for several choices of A_t considered next.

4.3 Full second-order updates

We postpone computational considerations to develop intuition for good choices of the matrix A_t. We begin by considering the *best fixed choice* for A in hindsight, before analyzing a related adaptive version.

4.3.1 Best choice in hindsight

Taking a cue from previous works [33, 80, 94], we first consider how to pick the best fixed matrix A, if all the $\boldsymbol{x}_t$ and gradients $\boldsymbol{g}_t$ were known and fixed in advance. Although this is fallacious as a method (since different choices of A produce different gradient sequences) this exercise provides intuition on good matrix design. Setting $A_t \equiv A \succ 0$ for all t, the regret bound (4.3) gives

$$2R_T(\boldsymbol{w}) \leq \|\boldsymbol{w}\|_A^2 + \sum_{t=1}^{T} \boldsymbol{g}_t^\top A^{-1} \boldsymbol{g}_t \ .$$

The best fixed matrix against all comparators $\boldsymbol{w} \in \mathcal{K}$ solves the following minimax problem

$$\inf_{A \succ 0} \sup_{\boldsymbol{w} \in \mathcal{K}} \|\boldsymbol{w}\|_A^2 + \sum_{t=1}^{t} \boldsymbol{g}_t^\top A^{-1} \boldsymbol{g}_t \ .$$

since $\boldsymbol{w} \in \mathcal{K}$ implies $|\boldsymbol{w}^\top \boldsymbol{x}_t| \leq C$ for all t. This minimax problem is identical to that of Ross et al. [94], yet dealing with the set $\mathcal{K}$ for non-diagonal matrices A is challenging. We instead endow the adversary with more power, allowing it to pick from a larger set $\mathcal{K}' = \left\{\boldsymbol{w} \in \mathbb{R}^d \ : \ \sum_{t=1}^{T} (\boldsymbol{w}^\top \boldsymbol{g}_t)^2 \leq TL^2C^2\right\}$. Assumption 1 ensures $\mathcal{K}' \supseteq \mathcal{K}$. With this relaxation, the following result holds.

Proposition 2. *If* $G \stackrel{\text{def}}{=} \sum_{t=1}^{T} \boldsymbol{g}_t \boldsymbol{g}_t^\top$ *is invertible then*

$$\inf_{A \succ 0} \sup_{\boldsymbol{w} \in \mathcal{K}'} \|\boldsymbol{w}\|_A^2 + \sum_{t=1}^{T} \boldsymbol{g}_t^\top A^{-1} \boldsymbol{g}_t = 2CL\sqrt{dT}$$

and the optimal choice is $A = \sqrt{\frac{d}{L^2C^2T}} G$.

Proof. Note that $\boldsymbol{w} \in \mathcal{K}'$ is equivalent to $\left\|G^{1/2}\boldsymbol{w}\right\|_2 \leq \sqrt{T}LC$, where $G = \sum_{t=1}^{T} \boldsymbol{g}_t \boldsymbol{g}_t^\top$. So by the change of variable $\widetilde{\boldsymbol{w}} = \frac{G^{1/2}\boldsymbol{w}}{\sqrt{T}LC}$, we have

$$\sup_{\boldsymbol{w} \in \mathcal{K}'} \|\boldsymbol{w}\|_A^2 = TL^2C^2 \sup_{\|\widetilde{\boldsymbol{w}}\|_2 \leq 1} \widetilde{\boldsymbol{w}}^\top (G^{-1/2} A G^{-1/2}) \widetilde{\boldsymbol{w}} = TL^2C^2 \lambda_1(B)$$

where $B = G^{-1/2}AG^{-1/2}$ and recall that $\lambda_i(\cdot)$ represents the i-th largest eigenvalue of the argument. So we have

$$\begin{aligned}
\sup_{\boldsymbol{w}\in\mathcal{K}'} \|\boldsymbol{w}\|_A^2 + \sum_{t=1}^{T} \boldsymbol{g}_t^\top A^{-1}\boldsymbol{g}_t &= TL^2C^2\lambda_1(B) + \sum_{t=1}^{T} \boldsymbol{g}_t^\top (G^{1/2}BG^{1/2})^{-1}\boldsymbol{g}_t \\
&= TL^2C^2\lambda_1(B) + \left\langle (G^{1/2}BG^{1/2})^{-1},\ \sum_{t=1}^{T} \boldsymbol{g}_t\boldsymbol{g}_t^\top \right\rangle \\
&= TL^2C^2\lambda_1(B) + \textsc{tr}(G^{-1/2}B^{-1}G^{-1/2}G) \\
&= TL^2C^2\lambda_1(B) + \textsc{tr}(B^{-1}) \\
&= TL^2C^2\lambda_1(B) + \sum_{i=1}^{d} \frac{1}{\lambda_i(B)}\ .
\end{aligned}$$

If $\lambda_i(B) < \lambda_1(B)$ for some $i \neq 1$, then by increasing $\lambda_i(B)$ so that $\lambda_i(B) = \lambda_1(B)$, we can always make the upper bound smaller, which means that the optimal B should have d identical eigenvalues, that is, $B = \lambda \boldsymbol{I}_d$ for some $\lambda > 0$. Plugging this form of B leads to $TL^2C^2\lambda + \frac{d}{\lambda} = 2CL\sqrt{Td}$ for $\lambda^* = \sqrt{\frac{d}{TC^2L^2}}$. The best A is thus $G^{1/2}(\lambda^*\boldsymbol{I}_d)G^{1/2} = \frac{1}{LC}\sqrt{\frac{d}{T}}G$. □

Proposition 2 suggests setting the matrices A_t in terms of the sum of gradient outer products. This choice is essentially identical to the Online Newton Step (ONS) algorithm [57], and is also similar to the full-matrix version of the AdaGrad method [33], where $G^{1/2}$ was used.

4.3.2 Adapting the matrix online

To turn the above intuition into an algorithm with the update form (4.2), we set

$$A_t = \alpha \boldsymbol{I}_d + \sum_{s=1}^{t} (\sigma_s + \eta_s)\boldsymbol{g}_s\boldsymbol{g}_s^\top \tag{4.4}$$

for some parameters $\alpha > 0$ and $\eta_t \geq 0$. The parameter α ensures the invertibility of A_t, although it does incorporate a dependence on the ℓ_2 geometry and does not leave the updates truly invariant as discussed later. The update is essentially identical to ONS, except that we project onto the set $\mathcal{K}_t$ at time t, while ONS projects onto a fixed set. The projection step can be performed in closed form as the next lemma shows.

Lemma 11. *For any $\boldsymbol{x} \neq \boldsymbol{0}, \boldsymbol{u} \in \mathbb{R}^d$ and positive definite matrix $A \in \mathbb{R}^{d \times d}$, we have*

$$\operatorname*{argmin}_{\boldsymbol{w} \,:\, |\boldsymbol{w}^\top \boldsymbol{x}| \leq C} \|\boldsymbol{w} - \boldsymbol{u}\|_A = \boldsymbol{u} - \frac{\tau_C(\boldsymbol{u}^\top \boldsymbol{x})}{\boldsymbol{x}^\top A^{-1} \boldsymbol{x}} A^{-1} \boldsymbol{x}$$

where $\tau_C(y) = sign(y) \max\{|y| - C, 0\}$.

The proof of the above lemma (generalized to rank deficient A) is in Appendix C.1. The same projection in a different context than invariant learning appears in Gentile and Orabona [45]. This choice of the matrices A_t simplifies the general regret bound of Proposition 1.

Theorem 16. *Under Assumptions 1 and 2, suppose that $\sigma_t \geq \sigma \geq 0$ for all t, and η_t is non-increasing. Then using the matrices* (4.4) *in the updates* (4.2) *yields for all $\boldsymbol{w} \in \mathcal{K}$*

$$R_T(\boldsymbol{w}) \leq \frac{\alpha}{2} \|\boldsymbol{w}\|_2^2 + 2(CL)^2 \sum_{t=1}^{T} \eta_t + \frac{d}{2(\sigma + \eta_T)} \ln \left(1 + \frac{(\sigma + \eta_T) \sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2}{d\alpha} \right) .$$

Proof. The theorem is proved by applying Proposition 1 with the choice of the matrices A_t in Theorem 16. Recalling that $A_0 = \alpha \boldsymbol{I}_d$ and $A_t = A_{t-1} + (\sigma_t + \eta_t) \boldsymbol{g}_t \boldsymbol{g}_t^T$, we

obtain

$$
\begin{aligned}
2R_T(\boldsymbol{w}) &\leq \|\boldsymbol{w}\|_{A_0}^2 + \sum_{t=1}^{T} \boldsymbol{g}_t^\top A_t^{-1} \boldsymbol{g}_t + (\boldsymbol{w}_t - \boldsymbol{w})^\top (A_t - A_{t-1} - \sigma_t \boldsymbol{g}_t \boldsymbol{g}_t^\top)(\boldsymbol{w}_t - \boldsymbol{w}) \\
&= \alpha \|\boldsymbol{w}\|_2^2 + \sum_{t=1}^{T} \boldsymbol{g}_t^\top A_t^{-1} \boldsymbol{g}_t + \eta_t (\boldsymbol{w}_t - \boldsymbol{w})^\top \boldsymbol{g}_t \boldsymbol{g}_t^\top (\boldsymbol{w}_t - \boldsymbol{w}) \\
&\leq \alpha \|\boldsymbol{w}\|_2^2 + \sum_{t=1}^{T} \boldsymbol{g}_t^\top A_t^{-1} \boldsymbol{g}_t + 4(CL)^2 \eta_t
\end{aligned}
$$

where the last equality uses the Lipschitz property in Assumption 1 and the boundedness of $\boldsymbol{w}_t^\top \boldsymbol{x}_t$ and $\boldsymbol{w}^\top \boldsymbol{x}_t$.

Define $\widehat{A}_t = \frac{\alpha}{\sigma + \eta_T} \boldsymbol{I}_d + \sum_{s=1}^{t} \boldsymbol{g}_s \boldsymbol{g}_s^\top$. Since $\sigma_t \geq \sigma$ and η_t is non-increasing, we have $\widehat{A}_t \preceq \frac{1}{\sigma + \eta_T} A_t$. We now proceed as follows:

$$
\begin{aligned}
\sum_{t=1}^{T} \boldsymbol{g}_t^\top A_t^{-1} \boldsymbol{g}_t &\leq \frac{1}{\sigma + \eta_T} \sum_{t=1}^{T} \boldsymbol{g}_t^\top \widehat{A}_t^{-1} \boldsymbol{g}_t \\
&= \frac{1}{\sigma + \eta_T} \sum_{t=1}^{T} \left\langle \widehat{A}_t - \widehat{A}_{t-1},\ \widehat{A}_t^{-1} \right\rangle \\
&\leq \frac{1}{\sigma + \eta_T} \sum_{t=1}^{T} \ln \frac{|\widehat{A}_t|}{|\widehat{A}_{t-1}|} \\
&= \frac{1}{\sigma + \eta_T} \ln \frac{|\widehat{A}_T|}{|\widehat{A}_0|} \\
&= \frac{1}{\sigma + \eta_T} \sum_{i=1}^{d} \ln \left(1 + \frac{(\sigma + \eta_T) \lambda_i \left(\sum_{t=1}^{T} \boldsymbol{g}_t \boldsymbol{g}_t^\top \right)}{\alpha} \right) \\
&\leq \frac{d}{\sigma + \eta_T} \ln \left(1 + \frac{(\sigma + \eta_T) \sum_{i=1}^{d} \lambda_i \left(\sum_{t=1}^{T} \boldsymbol{g}_t \boldsymbol{g}_t^\top \right)}{d\alpha} \right) \\
&= \frac{d}{\sigma + \eta_T} \ln \left(1 + \frac{(\sigma + \eta_T) \sum_{t=1}^{T} \|\boldsymbol{g}_t\|_2^2}{d\alpha} \right)
\end{aligned}
$$

where the second inequality is by the concavity of the function $\ln |X|$ (see [57, Lemma 12] for an alternative proof), and the last one is by Jensen's inequality. This concludes the proof. □

As noted earlier, the dependence on $\|\boldsymbol{w}\|_2^2$ implies that the method is not truly invariant to transformations of the data. However, this is not critical since α is fixed and small while the other part of the bound grows to eventually become the dominating term. Moreover, we can set $\alpha = 0$ and replace the inverse with the Moore-Penrose pseudoinverse, as discussed in Appendix C.2. We use $\alpha > 0$ in the remainder for simplicity.

If $\sigma = 0$ and $\eta_t = \frac{1}{CL}\sqrt{\frac{d}{t}}$, the bound simplifies to

$$R_T(\boldsymbol{w}) \leq \frac{\alpha}{2}\|\boldsymbol{w}\|_2^2 + \frac{CL}{2}\sqrt{Td}\ln\left(1 + \frac{\sum_{t=1}^T \|\boldsymbol{g}_t\|_2^2}{\alpha CL\sqrt{Td}}\right) + 4CL\sqrt{Td}\ ,$$

only losing a logarithmic factor from Proposition 2.

If $\sigma_t \geq \sigma > 0$ for all t, then with $\eta_t = 0$ the regret is

$$R_T(\boldsymbol{w}) \leq \frac{\alpha}{2}\|\boldsymbol{w}\|_2^2 + \frac{d}{2\sigma}\ln\left(1 + \frac{\sigma\sum_{t=1}^T \|\boldsymbol{g}_t\|_2^2}{d\alpha}\right)\ . \tag{4.5}$$

extending the $O(d\ln T)$ results of Hazan et al. [57] to the weaker Assumption 2 and larger comparator set.

4.4 Efficiency via Sketching

Our updates so far require $\Omega(d^2)$ time just as ONS. Here we show how to achieve regret guarantees nearly as good as the above bounds, while keeping computation within a constant factor of first-order methods.

Let $G_t \in \mathbb{R}^{t\times d}$ be a matrix such that the t-th row is $\widehat{\boldsymbol{g}}_t^\top$ where we define $\widehat{\boldsymbol{g}}_t = \sqrt{\sigma_t + \eta_t}\boldsymbol{g}_t$ to be the *to-sketch vector*. Recall our previous setting $A_t = \alpha\boldsymbol{I}_d + G_t^\top G_t$ (4.4). Sketching maintains an approximation of G_t, denoted by $S_t \in \mathbb{R}^{m\times d}$ where $m \ll d$ is the sketch size. If m is chosen so that $S_t^\top S_t$ approximates $G_t^\top G_t$ well, we can redefine A_t as $\alpha\boldsymbol{I}_d + S_t^\top S_t$ for the algorithm.

Algorithm 11 Sketched Online Newton (SON)

Input: Parameters C, α and m.

Initialize $\boldsymbol{u}_1 = \mathbf{0}_{d\times 1}$.
Initialize sketch $(S, H) \leftarrow \textbf{SketchInit}(\alpha, m)$.
for $t = 1$ **to** T **do**
    Receive example $\boldsymbol{x}_t$.
    Projection step: compute $\widehat{\boldsymbol{x}} = S\boldsymbol{x}_t$, $c = \frac{\tau_C(\boldsymbol{u}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t - \widehat{\boldsymbol{x}}^\top H \widehat{\boldsymbol{x}}}$ and set $\boldsymbol{w}_t = \boldsymbol{u}_t - c(\boldsymbol{x}_t - S^\top H\widehat{\boldsymbol{x}})$.
    Predict label $y_t = \boldsymbol{w}_t^\top \boldsymbol{x}_t$ and suffer loss $\ell_t(y_t)$.
    Compute gradient $\boldsymbol{g}_t = \ell_t'(y_t)\boldsymbol{x}_t$ and the *to-sketch vector* $\widehat{\boldsymbol{g}} = \sqrt{\sigma_t + \eta_t}\boldsymbol{g}_t$.
    $(S, H) \leftarrow \textbf{SketchUpdate}(\widehat{\boldsymbol{g}})$.
    Update weight:
$$\boldsymbol{u}_{t+1} = \boldsymbol{w}_t - \tfrac{1}{\alpha}(\boldsymbol{g}_t - S^\top H S \boldsymbol{g}_t).$$
end for

To see why this admits an efficient algorithm, notice that by Woodbury formula [109] one has

$$A_t^{-1} = \frac{1}{\alpha}\big(\boldsymbol{I}_d - S_t^\top(\alpha \boldsymbol{I}_m + S_t S_t^\top)^{-1} S_t\big)\ .$$

With the notation $H_t = (\alpha \boldsymbol{I}_m + S_t S_t^\top)^{-1} \in \mathbb{R}^{m\times m}$ and $c_t = \tau_C(\boldsymbol{u}_{t+1}^\top \boldsymbol{x}_{t+1})/(\boldsymbol{x}_{t+1}^\top \boldsymbol{x}_{t+1} - \boldsymbol{x}_{t+1}^\top S_t^\top H_t S_t \boldsymbol{x}_{t+1})$, update (4.2) becomes:

$$\begin{aligned}
\boldsymbol{u}_{t+1} &= \boldsymbol{w}_t - \frac{1}{\alpha}\big(\boldsymbol{g}_t - S_t^\top H_t S_t \boldsymbol{g}_t\big) \\
\boldsymbol{w}_{t+1} &= \boldsymbol{u}_{t+1} - c_t\big(\boldsymbol{x}_{t+1} - S_t^\top H_t S_t \boldsymbol{x}_{t+1}\big)\ .
\end{aligned}$$

The operations involving $S_t\boldsymbol{x}_t$ or $S_t\boldsymbol{g}_t$ require only $O(md)$ time, while matrix vector products with H_t require only $O(m^2)$. Altogether, updates are at most m times more expensive than first-order ones when H_t can be maintained efficiently. The Sketched Online Newton (SON) algorithm is presented in Algorithm 11.

We now discuss three sketching techniques to maintain the matrices S_t and H_t efficiently, each requiring $O(md)$ storage and time linear in d.

Algorithm 12 Random Projection Sketch for RP-SON

Internal States: S and H.

SketchInit(α, m)

1: Set $S = \mathbf{0}_{m\times d}$ and $H = \frac{1}{\alpha}\boldsymbol{I}_m$.
2: Return (S, H).

SketchUpdate$(\widehat{\boldsymbol{g}})$

1: Draw $\boldsymbol{r} \sim \mathcal{N}(0, \frac{1}{\sqrt{m}})$ and update $S \leftarrow S + \boldsymbol{r}\widehat{\boldsymbol{g}}^\top$.
2: Compute $\boldsymbol{q} = S\widehat{\boldsymbol{g}} - \frac{\widehat{\boldsymbol{g}}^\top\widehat{\boldsymbol{g}}}{2}\boldsymbol{r}$, update $H \leftarrow H - \frac{H\boldsymbol{q}\boldsymbol{r}^\top H}{1+\boldsymbol{r}^\top H\boldsymbol{q}}$ and $H \leftarrow H - \frac{H\boldsymbol{r}\boldsymbol{q}^\top H}{1+\boldsymbol{q}^\top H\boldsymbol{r}}$.
3: Return (S, H).

4.4.1 Random Projection

Random projections are classical methods for sketching [61, 3, 64]. Here we consider Gaussian Random Projection (RP) sketch: $S_t = S_{t-1} + \boldsymbol{r}_t\widehat{\boldsymbol{g}}_t^\top$, where each entry of $\boldsymbol{r}_t \in \mathbb{R}^m$ is an independent random Gaussian variable drawn from $\mathcal{N}(0, 1/\sqrt{m})$. Since the update of H_t^{-1} can be realized by two rank-one updates:

$$H_t^{-1} = \alpha\boldsymbol{I}_d + S_tS_t^\top = \alpha\boldsymbol{I}_d + (S_{t-1} + \boldsymbol{r}_t\widehat{\boldsymbol{g}}_t^\top)(S_{t-1} + \boldsymbol{r}_t\widehat{\boldsymbol{g}}_t^\top)^\top = H_{t-1}^{-1} + \boldsymbol{q}_t\boldsymbol{r}_t^\top + \boldsymbol{r}_t\boldsymbol{q}_t^\top,$$

where $\boldsymbol{q}_t = S_t\widehat{\boldsymbol{g}}_t - \frac{\|\widehat{\boldsymbol{g}}_t\|_2^2}{2}\boldsymbol{r}_t$, with Woodbury formula this results in $O(md)$ update of S and H (see Algorithm 12), and the overall time complexity for each round is also $O(md)$. Moreover when $\alpha = 0$ the algorithm is still invariant to linear transformation for each fixed realization of the randomness. If r is the rank of G_T, then Proposition 1 implies the following bound (recall the definition of R_D and R_G in Eq. (4.3)), which further implies that the regret is the same as the one in Theorem 16 up to constants.

Theorem 17. *Under Assumptions 1 and 2, if SON is run with the Random Projection sketch of size* $m = \Omega\big((r + \ln(T/\delta))\epsilon^{-2}\big)$*, where* $r = \mathit{rank}(G_T)$*, then*

(1) $\mathbb{E}[R_D] \le 4(CL)^2\sum_{t=1}^T \eta_t$*, and*

(2) $R_G \le \frac{1}{1-\epsilon}\sum_{t=1}^T \boldsymbol{g}_t^\top(\alpha\boldsymbol{I}_d + G_t^\top G_t)^{-1}\boldsymbol{g}_t$ *with probability at least* $1-\delta$.

Proof. We apply the property of the random projection method (see for example [110, Theorem 2.3]): as long as $m = \Omega\big((r + \ln(T/\delta))\epsilon^{-2}\big)$, with probability at least $1 - \delta$,

$$(1-\epsilon)G_t^\top G_t \preceq S_t^\top S_t \preceq (1+\epsilon)G_t^\top G_t \qquad \text{for all } t = 1, \dots, T$$

which implies $A_t^{-1} \preceq \frac{1}{1-\epsilon}(\alpha \boldsymbol{I}_d + G_t^\top G_t)^{-1}$ and thus $R_G \leq \frac{1}{1-\epsilon}\sum_{t=1}^T \boldsymbol{g}_t^\top(\alpha \boldsymbol{I}_d + G_t^\top G_t)^{-1}\boldsymbol{g}_t$. For R_D, first fix all the randomness before drawing $\boldsymbol{r}_t$ and let $\mathbb{E}_t$ be the corresponding conditional expectation, then we have

$$\mathbb{E}_t[A_t - A_{t-1}] = \mathbb{E}_t\Big[S_{t-1}^\top \boldsymbol{r}_t \widehat{\boldsymbol{g}}_t^\top + \widehat{\boldsymbol{g}}_t \boldsymbol{r}_t^\top S_{t-1} + \|\boldsymbol{r}_t\|_2^2 \widehat{\boldsymbol{g}}_t \widehat{\boldsymbol{g}}_t^\top\Big] = (\sigma_t + \eta_t)\boldsymbol{g}_t \boldsymbol{g}_t^\top \ .$$

Since $\boldsymbol{w}_t, \boldsymbol{w}$ and $\boldsymbol{g}_t$ are fixed, we continue with

$$\mathbb{E}_t\Big[(\boldsymbol{w}_t - \boldsymbol{w})^\top (A_t - A_{t-1} - \sigma_t \boldsymbol{g}_t \boldsymbol{g}_t^\top)(\boldsymbol{w}_t - \boldsymbol{w})\Big] = \eta_t (\boldsymbol{w}_t - \boldsymbol{w})^\top \boldsymbol{g}_t \boldsymbol{g}_t^\top (\boldsymbol{w}_t - \boldsymbol{w}) \leq 4(CL)^2 \eta_t \ .$$

Therefore, taking the overall expectation gives $\mathbb{E}[R_D] \leq 4(CL)^2 \sum_{t=1}^T \eta_t$. □

We call this combination of SON with RP-sketch RP-SON. Comparing Theorems 16 and 17, the expected regret is the same up to constants when ϵ is a constant. Consequently, RP-SON is a near invariant algorithm that gives substantial computational gains when r is significantly smaller than d with minimal regret overhead.

4.4.2 Frequent Directions

When the gradient matrix is near full-rank, RP-SON may not perform well. Frequent Directions (FD) sketch [69, 46] is a deterministic sketching method which maintains the invariant that the last row of S_t is always $\mathbf{0}$. On each round, the vector $\widehat{\boldsymbol{g}}_t^\top$ is inserted into the last row of S_{t-1}, then the covariance of the resulting matrix is eigendecomposed into $V_t^\top \Sigma_t V_t$ and S_t is set to $(\Sigma_t - \rho_t \boldsymbol{I}_m)^{\frac{1}{2}} V_t$ where ρ_t is the smallest eigenvalue. Since the rows of S_t are orthogonal to each other, $S_t S_t^\top$ is a diagonal

Algorithm 13 Frequent Direction Sketch for FD-SON

Internal States: S and H.

SketchInit(α, m)

1: Set $S = \mathbf{0}_{m\times d}$ and $H = \frac{1}{\alpha}\boldsymbol{I}_m$.
2: Return (S, H).

SketchUpdate$(\widehat{\boldsymbol{g}})$

1: Insert $\widehat{\boldsymbol{g}}$ into the last row of S.
2: Compute eigendecomposition: $V^\top \Sigma V = S^\top S$ and set $S = (\Sigma - \Sigma_{m,m}\boldsymbol{I}_m)^{\frac{1}{2}} V$.
3: Set $H = \mathrm{diag}\left\{\frac{1}{\alpha+\Sigma_{1,1}-\Sigma_{m,m}}, \cdots, \frac{1}{\alpha+\Sigma_{m,m}-\Sigma_{m,m}}\right\}$.
4: Return (S, H).

matrix and so is H_t (see Algorithm 13). We call the combination of SON with FD-sketch FD-SON.

The sketch update works in $O(md)$ time (see [46]) so the total running time is thus $O(md)$ per round. The following regret bound uses the notation $\Omega_k = \sum_{i=k+1}^{d} \lambda_i(G_T^\top G_T)$ for any $k = 0, \ldots, m-1$.

Theorem 18. *Assuming $\sigma_t \geq \sigma \geq 0$ for all t, Assumptions 1 and 2 hold, and η_t is non-increasing, If SON uses the Frequent Directions sketch, then $R_T(\boldsymbol{w})$ is at most*

$$\frac{\alpha}{2}\left\|\boldsymbol{w}\right\|_2^2 + \frac{m}{2(\sigma+\eta_T)}\ln\left(1 + \frac{\mathrm{TR}(S_T^\top S_T)}{m\alpha}\right) + \frac{m\Omega_k}{2(m-k)(\sigma+\eta_T)\alpha} + 2(CL)^2\sum_{t=1}^{T}\eta_t$$

for all $k \in \{0, \ldots, m-1\}$ and $\boldsymbol{w} \in \mathcal{K}$.

Proof. By the construction of the sketch, we have $A_t - A_{t-1} = S_t^\top S_t - S_{t-1}^\top S_{t-1} = \widehat{\boldsymbol{g}}_t\widehat{\boldsymbol{g}}_t^\top - \rho_t V_t^\top V_t \preceq \widehat{\boldsymbol{g}}_t\widehat{\boldsymbol{g}}_t^\top$. It follows immediately that R_D in Eq. (4.3) is again at most $4(CL)^2\sum_{t=1}^{T}\eta_t$. For the term R_G, we will apply the following guarantee of Frequent Directions (see the proof of Theorem 1.1 of [46]): $\sum_{t=1}^{T}\rho_t \leq \frac{\Omega_k}{m-k}$. Specifically, since

$\text{TR}(V_t A_t^{-1} V_t^\top) \leq \frac{1}{\alpha}\text{TR}(V_t V_t^\top) = \frac{m}{\alpha}$ we have

$$\begin{aligned}
R_G &= \sum_{t=1}^{T} \frac{1}{\sigma_t + \eta_t} \left\langle A_t^{-1}, A_t - A_{t-1} + \rho_t V_t^\top V_t \right\rangle \\
&\leq \frac{1}{\sigma + \eta_T} \sum_{t=1}^{T} \left(\left\langle A_t^{-1}, A_t - A_{t-1} + \rho_t V_t^\top V_t \right\rangle \right) \\
&= \frac{1}{\sigma + \eta_T} \sum_{t=1}^{T} \left(\left\langle A_t^{-1}, A_t - A_{t-1} \right\rangle + \rho_t \text{TR}(V_t A_t^{-1} V_t^\top) \right) \\
&\leq \frac{1}{(\sigma + \eta_T)} \sum_{t=1}^{T} \left\langle A_t^{-1}, A_t - A_{t-1} \right\rangle + \frac{m\Omega_k}{(m-k)(\sigma + \eta_T)\alpha} \ .
\end{aligned}$$

Finally for the term $\sum_{t=1}^{T} \left\langle A_t^{-1}, A_t - A_{t-1} \right\rangle$, we proceed similarly to the proof of Theorem 16:

$$\begin{aligned}
\sum_{t=1}^{T} \left\langle A_t^{-1}, A_t - A_{t-1} \right\rangle &\leq \sum_{t=1}^{T} \ln \frac{|A_t|}{|A_{t-1}|} = \ln \frac{|A_T|}{|A_0|} = \sum_{i=1}^{d} \ln \left(1 + \frac{\lambda_i(S_T^\top S_T)}{\alpha} \right) \\
&= \sum_{i=1}^{m} \ln \left(1 + \frac{\lambda_i(S_T^\top S_T)}{\alpha} \right) \leq m \ln \left(1 + \frac{\text{TR}(S_T^\top S_T)}{m\alpha} \right)
\end{aligned}$$

where the first inequality is by the concavity of the function $\ln |X|$, the second one is by Jensen's inequality, and the last equality is by the fact that $S_T^\top S_T$ is of rank m and thus $\lambda_i(S_T^\top S_T) = 0$ for any $i > m$. This concludes the proof. □

If $\sigma_t \geq \sigma > 0$, setting $\eta_t = 0$ gives the bound

$$\alpha \left\| w \right\|_2^2 + \frac{m}{\sigma} \ln \left(1 + \frac{\text{TR}(S_T^\top S_T)}{m\alpha} \right) + \frac{m\Omega_k}{(m-k)\sigma\alpha} \ .$$

Instead of the rank, the bound depends on the spectral decay Ω_k. With α tuned well, we pay logarithmic regret for the top m eigenvectors, just as Theorem 16 (4.5), but a square root regret for remaining directions not controlled by our sketch. This is expected for deterministic sketching which focuses on the dominant part of the spectrum. When α is not tuned we still get sublinear regret as long as Ω_k is sublinear.

Algorithm 14 Oja's Sketch for Oja-SON

Internal States: t, Λ, V and H.

SketchInit(α, m)

1: Set $t = 0, \Lambda = \mathbf{0}_{m\times m}, H = \frac{1}{\alpha}\boldsymbol{I}_m$ and let V be any $m \times d$ matrix whose rows are orthonormal.
2: Return $(\mathbf{0}_{m\times d},\, H)$.

SketchUpdate$(\widehat{\boldsymbol{g}})$

1: Update $t \leftarrow t+1$, $\Lambda \leftarrow (\boldsymbol{I}_m - \Gamma_t)\Lambda + \Gamma_t \operatorname{diag}\{V\widehat{\boldsymbol{g}}\}^2$ and $V \leftarrow V + \Gamma_t V\widehat{\boldsymbol{g}}\widehat{\boldsymbol{g}}^\top$ where Γ_t is a diagonal matrix with learning rates on the diagonal.
2: Orthonormalize the rows of V, set $S = (t\Lambda)^{\frac{1}{2}}V$ and $H = \operatorname{diag}\Big\{\frac{1}{\alpha+t\Lambda_{1,1}}, \cdots, \frac{1}{\alpha+t\Lambda_{m,m}}\Big\}$.
3: Return (S, H).

If the rank of G_T is smaller than m, then $\Omega_{m-1} = 0$ and thus the sketch does not bring any extra regret.

4.4.3 Oja's Algorithm

Oja's algorithm [85, 86] is not usually considered a sketching algorithm but seems natural here. Oja's algorithm uses online gradient descent to find the eigenvectors and eigenvalues of data in a streaming fashion, with the to-sketch vector $\widehat{\boldsymbol{g}}_t$'s as the input. Specifically, let $V_t \in \mathbb{R}^{m\times d}$ denote the estimated eigenvectors and the diagonal matrix $\Lambda_t \in \mathbb{R}^{m\times m}$ contain the estimated eigenvalues at the end of round t. Oja's algorithm updates as:

$$\Lambda_t = (\boldsymbol{I}_m - \Gamma_t)\Lambda_{t-1} + \Gamma_t \operatorname{diag}\{V_{t-1}\widehat{\boldsymbol{g}}_t\}^2$$
$$V_t \xleftarrow{\text{orth}} V_{t-1} + \Gamma_t V_{t-1}\widehat{\boldsymbol{g}}_t\widehat{\boldsymbol{g}}_t^\top$$

where $\Gamma_t \in \mathbb{R}^{m\times m}$ is a diagonal matrix with (possibly different) learning rates of order $\Theta(1/t)$ on the diagonal, and the "$\xleftarrow{\text{orth}}$" operator represents an orthonormalizing step.[13] The sketch is then $S_t = (t\Lambda_t)^{\frac{1}{2}}V_t$. The rows of S_t are orthogonal and thus

[13]For simplicity, we assume that $V_{t-1} + \Gamma_t V_{t-1}\widehat{\boldsymbol{g}}_t\widehat{\boldsymbol{g}}_t^\top$ is always of full rank so that the orthonormalizing step does not reduce the dimension of V_t.

H_t is an efficiently maintainable diagonal matrix (see Algorithm 14). We call the combination of SON with Oja's sketch Oja-SON.

The time complexity of Oja's is $O(m^2 d)$ per round due to the orthonormalizing step. To improve the running time to $O(md)$, one can only update the sketch every m rounds (similar to the block power method [50, 68]). The regret guarantee of this algorithm is unclear since existing analysis for Oja's algorithm is only for the stochastic setting (see e.g. [8, 68]). However, Oja's provides good performance experimentally.

4.5 Sparse Implementation

In many text applications data vectors (and hence gradients) are sparse in the sense that $\|\boldsymbol{x}_t\|_0 \leq s$ for all t and some small constant $s \ll d$. Most online first-order methods enjoy a per-example running time depending on s in such settings. Achieving the same for second order methods is more difficult since $A_t^{-1} g_t$ (or sketched versions) are typically dense even if g_t is sparse. Below, we show how to implement updates for each of the three algorithms we propose in a sparsity-dependent time. Note that mathematically these updates are equivalent to the non-sparse counterparts so regret guarantees are unchanged.

4.5.1 Sparse RP-SON

We recall the updates of RP sketch matrices. Since $\widehat{\boldsymbol{g}}_t$ is sparse, $S_t = S_{t-1} + r\widehat{\boldsymbol{g}}_t^\top$ is easily updated in $O(ms)$ time. H_t can also be updated in $O(m^2 + ms)$ time clearly. However, since the sketch S_t is getting denser and denser, direct update of the weight vector is a dense operation too. The solution is to represent and store $\boldsymbol{w}_t$ in the form of $\bar{\boldsymbol{w}}_t + S_{t-1}^\top \boldsymbol{b}_t$ for some $\bar{\boldsymbol{w}}_t \in \mathbb{R}^d$ and $\boldsymbol{b}_t \in \mathbb{R}^m$. Note that now computing the

Algorithm 15 Sparse Sketched Online Newton with Random Projection

Input: Parameters C, α and m.

1: Initialize $\bar{\boldsymbol{u}} = \mathbf{0}_{d\times 1}$, $\boldsymbol{b} = \mathbf{0}_{m\times 1}$ and $(S, H) \leftarrow$ **SketchInit**(α, m) (Algorithm 12).
2: **for** $t = 1$ **to** T **do**
3: Receive example $\boldsymbol{x}_t$.
4: Projection step: compute $\widehat{\boldsymbol{x}} = S\boldsymbol{x}_t$, $c = \frac{\tau_C(\bar{\boldsymbol{u}}^\top \boldsymbol{x}_t + \boldsymbol{b}^\top \widehat{\boldsymbol{x}})}{\boldsymbol{x}_t^\top \boldsymbol{x}_t - \widehat{\boldsymbol{x}}^\top H \widehat{\boldsymbol{x}}}$, $\bar{\boldsymbol{w}} = \bar{\boldsymbol{u}} - c\boldsymbol{x}_t$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + cH\widehat{\boldsymbol{x}}$.
5: Predict label $y_t = \bar{\boldsymbol{w}}^\top \boldsymbol{x}_t + \boldsymbol{b}^\top \widehat{\boldsymbol{x}}$ and suffer loss $\ell_t(y_t)$.
6: Compute gradient $\boldsymbol{g}_t = \ell_t'(y_t)\boldsymbol{x}_t$ and the to-sketch vector $\widehat{\boldsymbol{g}} = \sqrt{\sigma_t + \eta_t}\boldsymbol{g}_t$.
7: $(S, H) \leftarrow$ **SketchUpdate**$(\widehat{\boldsymbol{g}})$ (Algorithm 12).
8: Update $\bar{\boldsymbol{u}} = \bar{\boldsymbol{w}} - (\boldsymbol{r}^\top \boldsymbol{b})\widehat{\boldsymbol{g}} - \frac{1}{\alpha}\boldsymbol{g}_t$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + \frac{1}{\alpha}HS\boldsymbol{g}_t$.
9: **end for**

prediction $\boldsymbol{w}_t^\top \boldsymbol{x}_t$ needs $O(ms)$ time. Rewriting the update rules we have

$$\begin{aligned}\boldsymbol{u}_{t+1} &= \boldsymbol{w}_t - \frac{1}{\alpha}\boldsymbol{g}_t + \frac{1}{\alpha}S_t^\top H_t S_t \boldsymbol{g}_t = \bar{\boldsymbol{w}}_t + S_{t-1}^\top \boldsymbol{b}_t - \frac{1}{\alpha}\boldsymbol{g}_t + \frac{1}{\alpha}S_t^\top H_t S_t \boldsymbol{g}_t \\ &= \underbrace{\bar{\boldsymbol{w}}_t - \widehat{\boldsymbol{g}}_t \boldsymbol{r}_t^\top \boldsymbol{b}_t - \frac{1}{\alpha}\boldsymbol{g}_t}_{\stackrel{\text{def}}{=}\bar{\boldsymbol{u}}_{t+1}} + S_t^\top \underbrace{(\boldsymbol{b}_t + \frac{1}{\alpha}H_t S_t \boldsymbol{g}_t)}_{\stackrel{\text{def}}{=}\boldsymbol{b}'_{t+1}} .\end{aligned}$$

Since $\boldsymbol{g}_t$ and $\widehat{\boldsymbol{g}}_t$ are sparse, computing $\bar{\boldsymbol{u}}_{t+1}$ and $\boldsymbol{b}'_{t+1}$ needs $O(m^2 + ms)$ time. Finally, for the projection step, c_t can clearly be computed in $O(m^2 + ms)$ time, and the update rule of $\bar{\boldsymbol{w}}_{t+1}$ and $\boldsymbol{b}_{t+1}$ is thus derived as follows:

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{u}_{t+1} - c_t(\boldsymbol{x}_{t+1} - S_t^\top H_t S_t \boldsymbol{x}_{t+1}) \\ &= \bar{\boldsymbol{u}}_{t+1} + S_t^\top \boldsymbol{b}'_{t+1} - c_t(\boldsymbol{x}_{t+1} - S_t^\top H_t S_t \boldsymbol{x}_{t+1}) \\ &= \underbrace{\bar{\boldsymbol{u}}_{t+1} - c_t \boldsymbol{x}_{t+1}}_{\stackrel{\text{def}}{=}\bar{\boldsymbol{w}}_{t+1}} + S_t^\top \underbrace{(\boldsymbol{b}'_{t+1} + c_t H_t S_t \boldsymbol{x}_{t+1})}_{\stackrel{\text{def}}{=}\boldsymbol{b}_{t+1}}\end{aligned}$$

which again takes $O(m^2 + ms)$ time. Taken together, the total time complexity per round is $O(m^2 + ms)$. The pseudocode for this version of the algorithm is presented in Algorithm 15.

4.5.2 Sparse FD-SON

The sparse version of our algorithm with the Frequent Directions option is much more involved. We begin by taking a detour and introducing a fast and epoch-based variant of the Frequent Directions algorithm proposed in [46]. The idea is the following: instead of doing an eigendecomposition immediately after inserting a new $\widehat{\boldsymbol{g}}$ every round, we double the size of the sketch (to $2m$), keep up to m recent $\widehat{\boldsymbol{g}}$'s, do the decomposition only at the end of every m rounds and finally keep the top m eigenvectors with shrunk eigenvalues. The advantage of this variant is that it can be implemented straightforwardly in $O(md)$ time on average without doing a complicated rank-one SVD update, while still ensuring the exact same guarantee with the only price of doubling the sketch size.

Algorithm 16 shows the details of this variant and how we maintain H. The sketch S is always represented by two parts: the top part (DV) comes from the last eigendecomposition, and the bottom part (G) collects the recent to-sketch vector $\widehat{\boldsymbol{g}}$'s. Note that within each epoch, the update of H^{-1} is a rank-two update and thus H can be updated similarly to the case of random projection (Lines 3 and 4 of Algorithm 16).

Although we can use any available algorithm that runs in $O(m^2 d)$ time to do the eigendecomposition (Line 7 in Algorithm 16), we explicitly write down the procedure of reducing this problem to eigendecomposing a small square matrix in Algorithm 17, which will be important for deriving the sparse version of the algorithm. Lemma 12 proves that Algorithm 17 works correctly for finding the top m eigenvector and eigenvalues.

Lemma 12. *The outputs of Algorithm 17 are such that the i-th row of V' and the i-th entry of the diagonal of Σ are the i-th eigenvector and eigenvalue of $S^\top S$ respectively.*

Proof. Let $W^\top \in \mathbb{R}^{d\times(d-m-r)}$ be an orthonormal basis of the null space of $\begin{pmatrix} V \\ Q \end{pmatrix}$. By Line 2, we know that $GW^\top = \mathbf{0}$ and $E = (V^\top\ Q^\top\ W^\top)$ forms an orthonormal

Algorithm 16 Frequent Direction Sketch (epoch version)

Internal States: τ, D, V, G and H.

SketchInit(α, m)

1: Set $\tau = 1, D = \mathbf{0}_{m\times m}, G = \mathbf{0}_{m\times d}, H = \frac{1}{\alpha}\boldsymbol{I}_{2m}$ and let V be any $m \times d$ matrix whose rows are orthonormal.
2: Return $(\mathbf{0}_{2m\times d}, H)$.

SketchUpdate$(\widehat{\boldsymbol{g}})$

1: Insert $\widehat{\boldsymbol{g}}$ into the τ-th row of G.
2: **if** $\tau < m$ **then**
3: Let $\boldsymbol{e}$ be the $2m\times 1$ basic vector whose $(m+\tau)$-th entry is 1 and $\boldsymbol{q} = S\widehat{\boldsymbol{g}} - \frac{\widehat{\boldsymbol{g}}^\top\widehat{\boldsymbol{g}}}{2}\boldsymbol{e}$.
4: Update $H \leftarrow H - \frac{H\boldsymbol{q}\boldsymbol{e}^\top H}{1+\boldsymbol{e}^\top H\boldsymbol{q}}$ and $H \leftarrow H - \frac{H\boldsymbol{e}\boldsymbol{q}^\top H}{1+\boldsymbol{q}^\top H\boldsymbol{e}}$.
5: Update $\tau \leftarrow \tau + 1$.
6: **else**
7: $(V, \Sigma) \leftarrow$ **ComputeEigenSystem** $\left(\begin{pmatrix} DV \\ G \end{pmatrix}\right)$ (Algorithm 17).
8: Set D to be a diagonal matrix with $D_{i,i} = \sqrt{\Sigma_{i,i} - \Sigma_{m,m}},\ \forall i \in [m]$.
9: Set $H \leftarrow \mathrm{diag}\left\{\frac{1}{\alpha+D_{1,1}^2}, \cdots, \frac{1}{\alpha+D_{m,m}^2}, \frac{1}{\alpha}, \ldots, \frac{1}{\alpha}\right\}$.
10: Set $G = \mathbf{0}_{m\times d}$.
11: Set $\tau = 1$.
12: **end if**
13: Return $\left(\begin{pmatrix} DV \\ G \end{pmatrix}, H\right)$.

basis of $\mathbb{R}^d$. Therefore, we have

$$
\begin{aligned}
S^\top S &= V^\top D^2 V + G^\top G \\
&= E \begin{pmatrix} D^2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} E^\top + EE^\top G^\top GEE^\top \\
&= E \left(\begin{pmatrix} D^2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} + \begin{pmatrix} VG^\top \\ QG^\top \\ WG^\top \end{pmatrix} \begin{pmatrix} GV^\top & GQ^\top & GW^\top \end{pmatrix} \right) E^\top \\
&= (V^\top\ Q^\top) \underbrace{\left(\begin{pmatrix} D^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} + \begin{pmatrix} M^\top \\ L^\top \end{pmatrix} \begin{pmatrix} M & L \end{pmatrix} \right)}_{=C} \begin{pmatrix} V \\ Q \end{pmatrix}
\end{aligned}
$$

Algorithm 17 ComputeEigenSystem(S)

Input: $S = \begin{pmatrix} DV \\ G \end{pmatrix}$.

Output: Output $V' \in \mathbb{R}^{m \times d}$ and diagonal matrix $\Sigma \in \mathbb{R}^{m \times m}$ such that the i-th row of V' and the i-th entry of the diagonal of Σ are the i-th eigenvector and eigenvalue of $S^\top S$ respectively.

1: Compute $M = GV^\top$.

2: Decompose $G - MV$ into the form LQ where $L \in \mathbb{R}^{m \times r}$, Q is a $r \times d$ matrix whose rows are orthonormal and r is the rank of $G - MV$ (e.g. by a Gram-Schmidt process).

3: Compute the top m eigenvectors ($U \in \mathbb{R}^{m \times (m+r)}$) and eigenvalues ($\Sigma \in \mathbb{R}^{m \times m}$) of the matrix $\begin{pmatrix} D^2 & \mathbf{0}_{m \times r} \\ \mathbf{0}_{r \times m} & \mathbf{0}_{r \times r} \end{pmatrix} + \begin{pmatrix} M^\top \\ L^\top \end{pmatrix} \begin{pmatrix} M & L \end{pmatrix}$.

4: Return (V', Σ) where $V' = U \begin{pmatrix} V \\ Q \end{pmatrix}$.

where in the last step we use the fact $GQ^\top = (MV + LQ)Q^\top = L$. Now it is clear that the eigenvalue of C will be the eigenvalue of $S^\top S$ and the eigenvector of C will be the eigenvector of $S^\top S$ after left multiplied by matrix $(V^\top \ Q^\top)$, completing the proof. □

We are now ready to present the sparse version of SON with Frequent Direction sketch (Algorithm 18). The key point is that we represent V_t as $F_t Z_t$ for some $F_t \in \mathbb{R}^{m \times m}$ and $Z_t \in \mathbb{R}^{m \times d}$, and the weight vector $\boldsymbol{w}_t$ as $\bar{\boldsymbol{w}}_t + Z_{t-1}^\top \boldsymbol{b}_t$ and ensure that the update of Z_t and $\bar{\boldsymbol{w}}_t$ will always be sparse. To see this, denote the sketch S_t by $\begin{pmatrix} D_t F_t Z_t \\ G_t \end{pmatrix}$ and let $H_{t,1}$ and $H_{t,2}$ be the top and bottom half of H_t. Now the update

rule of $\boldsymbol{u}_{t+1}$ can be rewritten as

$$\begin{aligned}
\boldsymbol{u}_{t+1} &= \boldsymbol{w}_t - \left(\boldsymbol{I}_d - S_t^\top H_t S_t\right)\tfrac{\boldsymbol{g}_t}{\alpha} \\
&= \bar{\boldsymbol{w}}_t + Z_{t-1}^\top \boldsymbol{b}_t - \frac{1}{\alpha}\boldsymbol{g}_t + \frac{1}{\alpha}(Z_t^\top F_t^\top D_t, G_t^\top) \begin{pmatrix} H_{t,1} S_t \boldsymbol{g}_t \\ H_{t,2} S_t \boldsymbol{g}_t \end{pmatrix} \\
&= \underbrace{\bar{\boldsymbol{w}}_t + \frac{1}{\alpha}(G_t^\top H_{t,2} S_t \boldsymbol{g}_t - \boldsymbol{g}_t) - (Z_t - Z_{t-1})^\top \boldsymbol{b}_t}_{\bar{\boldsymbol{u}}_{t+1}} + Z_t^\top \underbrace{(\boldsymbol{b}_t + \frac{1}{\alpha} F_t^\top D_t H_{t,1} S_t \boldsymbol{g}_t)}_{\boldsymbol{b}'_{t+1}}
\end{aligned}$$

We will show that $Z_t - Z_{t-1} = \Delta_t G_t$ for some $\Delta_t \in \mathbb{R}^{m \times m}$ shortly, and thus the above update is efficient due to the fact that the rows of G_t are collections of previous sparse vectors $\widehat{\boldsymbol{g}}$.

Similarly, the update of $\boldsymbol{w}_{t+1}$ can be written as

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{u}_{t+1} - c_t(\boldsymbol{x}_{t+1} - S_t^\top H_t S_t \boldsymbol{x}_{t+1}) \\
&= \bar{\boldsymbol{u}}_{t+1} + Z_t^\top \boldsymbol{b}'_{t+1} - c_t \boldsymbol{x}_{t+1} + c_t (Z_t^\top F_t^\top D_t, G_t^\top) \begin{pmatrix} H_{t,1} S_t \boldsymbol{x}_{t+1} \\ H_{t,2} S_t \boldsymbol{x}_{t+1} \end{pmatrix} \\
&= \underbrace{\bar{\boldsymbol{u}}_{t+1} + c_t(G_t^\top H_{t,2} S_t \boldsymbol{x}_{t+1} - \boldsymbol{x}_{t+1})}_{\bar{\boldsymbol{w}}_{t+1}} + Z_t^\top \underbrace{(\boldsymbol{b}'_{t+1} + c_t F_t^\top D_t H_{t,1} S_t \boldsymbol{x}_{t+1})}_{\boldsymbol{b}_{t+1}}.
\end{aligned}$$

It is clear that c_t can be computed efficiently, and thus the update of $\boldsymbol{w}_{t+1}$ is also efficient. These updates correspond to Line 6 and 10 of Algorithm 18.

It remains to perform the sketch update efficiently. Algorithm 19 is the sparse version of Algorithm 16. The challenging part is to compute eigenvectors and eigenvalues efficiently. Fortunately, in light of Algorithm 17, using the new representation $V = FZ$ one can directly translate the process to Algorithm 20, and find that the eigenvectors can be expressed in the form $N_1 Z + N_2 G$. The only tricky part is the decomposing step (Line 2 of Algorithm 20). Essentially, this step requires finding L and Q such that $LQR = PR$ and the rows of QR are orthonormal, where in this case

Algorithm 18 Sparse Sketched Online Newton with Frequent Directions

Input: Parameters C, α and m.

1: Initialize $\bar{\boldsymbol{u}} = \mathbf{0}_{d\times 1}$, $\boldsymbol{b} = \mathbf{0}_{m\times 1}$ and $(D, F, Z, G, H) \leftarrow$ **SketchInit**(α, m) (Algorithm 19).

2: Let S denote the matrix $\begin{pmatrix} DFZ \\ G \end{pmatrix}$ throughout the algorithm (without actually computing it).

3: Let H_1 and H_2 denote the upper and lower half of H, i.e. $H = \begin{pmatrix} H_1 \\ H_2 \end{pmatrix}$.

4: **for** $t = 1$ **to** T **do**

5: Receive example $\boldsymbol{x}_t$.

6: Projection step: compute $\widehat{\boldsymbol{x}} = S\boldsymbol{x}_t$ and $c = \frac{\tau_C(\bar{\boldsymbol{u}}^\top \boldsymbol{x}_t + \boldsymbol{b}^\top Z\boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t - \widehat{\boldsymbol{x}}^\top H\widehat{\boldsymbol{x}}}$, update $\bar{\boldsymbol{w}} = \bar{\boldsymbol{u}} + c(G^\top H_2\widehat{\boldsymbol{x}} - \boldsymbol{x}_t)$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + cF^\top DH_1\widehat{\boldsymbol{x}}$.

7: Predict label $y_t = \bar{\boldsymbol{w}}^\top \boldsymbol{x}_t + \boldsymbol{b}^\top Z\boldsymbol{x}_t$ and suffer loss $\ell_t(y_t)$.

8: Compute gradient $\boldsymbol{g}_t = \ell_t'(y_t)\boldsymbol{x}_t$ and the to-sketch vector $\widehat{\boldsymbol{g}} = \sqrt{\sigma_t + \eta_t}\boldsymbol{g}_t$.

9: $(D, F, Z, G, H, \Delta) \leftarrow$ **SketchUpdate**$(\widehat{\boldsymbol{g}})$ (Algorithm 19).

10: Update $\bar{\boldsymbol{u}} = \bar{\boldsymbol{w}} + \frac{1}{\alpha}(G^\top H_2 S\boldsymbol{g} - \boldsymbol{g}) - G^\top \Delta^\top \boldsymbol{b}$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + \frac{1}{\alpha}F^\top DH_1 S\boldsymbol{g}$.

11: **end for**

$P = \begin{pmatrix} -MF & \boldsymbol{I}_m \end{pmatrix}$ and R is $\begin{pmatrix} Z \\ G \end{pmatrix}$. This can in fact be achieved by performing the Gram-Schmidt process to P in a Banach space where inner product is defined as $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^\top K\boldsymbol{b}$ with $K = RR^\top$ being the Gram matrix of R. Since we can efficiently maintain the Gram matrix of Z (see Line 10 of Algorithm 19) and thus R, this decomposing step can be done efficiently too. The modified Gram-Schmidt algorithm is presented in Algorithm 21 (which will also be used in sparse Oja's sketch), where Line 4 is the key difference compared to standard Gram-Schmidt.

Lastly, having the eigenvectors in the form $N_1Z + N_2G$, we can simply update F as N_1 and Z as $Z + N_1^{-1}N_2G$ so that the invariant $V = FZ$ still holds. The update of Z is sparse since G is sparse.

We finally summarize the results of this section in the following theorem.

Theorem 19. *The average running time of Algorithm 18 is $O(m^2 + ms)$ per round, and the regret bound is exactly the same as the one stated in Theorem 18.*

Algorithm 19 Sparse Frequent Direction Sketch

Internal States: τ, D, F, Z, G, H and K.

SketchInit(α, m)

1: Set $\tau = 1, D = \mathbf{0}_{m\times m}, F = K = \boldsymbol{I}_m, H = \frac{1}{\alpha}\boldsymbol{I}_{2m}, G = \mathbf{0}_{m\times d}$, and let Z be any $m \times d$ matrix whose rows are orthonormal.
2: Return (D, F, Z, G, H).

SketchUpdate$(\widehat{\boldsymbol{g}})$

1: Insert $\widehat{\boldsymbol{g}}$ into the τ-th row of G.
2: **if** $\tau < m$ **then**
3: Let $\boldsymbol{e}$ be the $2m \times 1$ basic vector whose $(m+\tau)$-th entry is 1 and compute $\boldsymbol{q} = S\widehat{\boldsymbol{g}} - \frac{\widehat{\boldsymbol{g}}^\top\widehat{\boldsymbol{g}}}{2}\boldsymbol{e}$.
4: Update $H \leftarrow H - \frac{H\boldsymbol{q}\boldsymbol{e}^\top H}{1+\boldsymbol{e}^\top H\boldsymbol{q}}$ and $H \leftarrow H - \frac{H\boldsymbol{e}\boldsymbol{q}^\top H}{1+\boldsymbol{q}^\top H\boldsymbol{e}}$.
5: Set $\Delta = \mathbf{0}_{m\times m}$.
6: Set $\tau \leftarrow \tau + 1$.
7: **else**
8: $(N_1, N_2, \Sigma) \leftarrow$ **ComputeSparseEigenSystem** $\left(\begin{pmatrix} DFZ \\ G \end{pmatrix}, K\right)$ (Algorithm 20).
9: Compute $\Delta = N_1^{-1}N_2$.
10: Update Gram matrix $K \leftarrow K + \Delta GZ^\top + ZG^\top\Delta^\top + \Delta GG^\top\Delta^\top$.
11: Update $F = N_1, Z \leftarrow Z + \Delta G$, and let D be such that $D_{i,i} = \sqrt{\Sigma_{i,i} - \Sigma_{m,m}},\ \forall i \in [m]$.
12: Set $H \leftarrow \mathrm{diag}\left\{\frac{1}{\alpha+D_{1,1}^2}, \cdots, \frac{1}{\alpha+D_{m,m}^2}, \frac{1}{\alpha}, \ldots, \frac{1}{\alpha}\right\}$.
13: Set $G = \mathbf{0}_{m\times d}$.
14: Set $\tau = 1$.
15: **end if**
16: Return (D, F, Z, G, H, Δ).

4.5.3 Sparse Oja-SON

We finally provide the details for the sparse version of Oja-SON, which runs in $O(m^3 + ms)$ time. There are two ingredients to doing this:

1. The eigenvectors V_t are represented as $V_t = F_tZ_t$, where $Z_t \in \mathbb{R}^{m\times d}$ is a sparsely updatable direction and $F_t \in \mathbb{R}^{m\times m}$ is a matrix such that F_tZ_t is orthonormal.

2. The weights $\boldsymbol{w}_t$ are again split as $\bar{\boldsymbol{w}}_t + Z_{t-1}^\top\boldsymbol{b}_t$ where $\boldsymbol{b}_t \in \mathbb{R}^m$ maintains the weights on the subspace captured by V_{t-1} (which is same as Z_{t-1}). $\bar{\boldsymbol{w}}_t$ captures

Algorithm 20 ComputeSparseEigenSystem(S, K)

Input: $S = \begin{pmatrix} DFZ \\ G \end{pmatrix}$ and Gram matrix $K = ZZ^\top$.

Output: Output $N_1, N_2 \in \mathbb{R}^{m\times m}$ and diagonal matrix $\Sigma \in \mathbb{R}^{m\times m}$ such that the i-th row of $N_1 Z + N_2 G$ and the i-th entry of the diagonal of Σ are the i-th eigenvector and eigenvalue of the matrix $S^\top S$.

1: Compute $M = GZ^\top F^\top$.
2: $(L, Q) \leftarrow$ Decompose $\left(\begin{pmatrix} -MF & \boldsymbol{I}_m \end{pmatrix}, \begin{pmatrix} K & ZG^\top \\ GZ^\top & GG^\top \end{pmatrix} \right)$ (Algorithm 21).
3: Let r be the number of columns of L. Compute the top m eigenvectors ($U \in \mathbb{R}^{m\times(m+r)}$) and eigenvalues ($\Sigma \in \mathbb{R}^{m\times m}$) of the matrix $\begin{pmatrix} D^2 & \mathbf{0}_{m\times r} \\ \mathbf{0}_{r\times m} & \mathbf{0}_{r\times r} \end{pmatrix} + \begin{pmatrix} M^\top \\ L^\top \end{pmatrix} \begin{pmatrix} M & L \end{pmatrix}$.
4: Set $N_1 = U_1 F + U_2 Q_1$ and $N_2 = U_2 Q_2$ where U_1 and U_2 are the first m and last r columns of U respectively, and Q_1 and Q_2 are the left and right half of Q respectively.
5: Return (N_1, N_2, Σ).

Algorithm 21 Decompose(P, K)

Input: $P \in \mathbb{R}^{m\times n}$, $K \in \mathbb{R}^{m\times m}$ such that K is the Gram matrix $K = RR^\top$ for some matrix $R \in \mathbb{R}^{n\times d}$ where $n \geq m, d \geq m$,

Output: Output $L \in \mathbb{R}^{m\times r}$ and $Q \in \mathbb{R}^{r\times n}$ such that $LQR = PR$ where r is the rank of PR and the rows of QR are orthonormal.

1: Initialize $L = \mathbf{0}_{m\times m}$ and $Q = \mathbf{0}_{m\times n}$.
2: **for** $i = 1$ **to** m **do**
3: Let $\boldsymbol{p}^\top$ be the i-th row of P.
4: Compute $\boldsymbol{\alpha} = QK\boldsymbol{p}, \boldsymbol{\beta} = \boldsymbol{p} - Q^\top \boldsymbol{\alpha}$ and $c = \sqrt{\boldsymbol{\beta}^\top K \boldsymbol{\beta}}$.
5: **if** $c \neq 0$ **then**
6: Insert $\frac{1}{c}\boldsymbol{\beta}^\top$ to the i-th row of Q.
7: **end if**
8: Set the i-th entry of $\boldsymbol{\alpha}$ to be c and insert $\boldsymbol{\alpha}$ to the i-th row of L.
9: **end for**
10: Delete the all-zero columns of L and all-zero rows of Q.
11: Return (L, Q).

the residual weights on the complementary subspace which are again updated sparsely.

Note that the Oja's updates can now be written in terms of F and Z as

$$\begin{aligned}\Lambda_t &= (\boldsymbol{I}_m - \Gamma_t)\Lambda_{t-1} + \Gamma_t \operatorname{diag}\{F_{t-1}Z_{t-1}\widehat{\boldsymbol{g}}_t\}^2 \\ F_tZ_t &\xleftarrow{\text{orth}} F_{t-1}Z_{t-1} + \Gamma_tF_{t-1}Z_{t-1}\widehat{\boldsymbol{g}}_t\widehat{\boldsymbol{g}}_t^\top = F_{t-1}(Z_{t-1} + F_{t-1}^{-1}\Gamma_tF_{t-1}Z_{t-1}\widehat{\boldsymbol{g}}_t\widehat{\boldsymbol{g}}_t^\top)\ .\end{aligned} \tag{4.6}$$

Here, the update for the eigenvalues is straightforward. For the update of eigenvectors, first we let $Z_t = Z_{t-1} + \boldsymbol{\delta}_t\widehat{\boldsymbol{g}}_t^\top$ where $\boldsymbol{\delta}_t = F_{t-1}^{-1}\Gamma_tF_{t-1}Z_{t-1}\widehat{\boldsymbol{g}}_t$ (note that under the assumption of Footnote 13, F_t is always invertible). Finally it remains to update F_t so that F_tZ_t is the same as orthonormalizing $F_{t-1}Z_t$, which can in fact be achieved by applying the Gram-Schmidt algorithm to F_{t-1} in a Banach space where inner product is defined as $\langle \boldsymbol{a}, \boldsymbol{b}\rangle = \boldsymbol{a}^\top K_t\boldsymbol{b}$ where K_t is the Gram matrix $Z_tZ_t^\top$ (see Algorithm 21). Since we can maintain K_t efficiently based on the update of Z_t:

$$K_t = K_{t-1} + \boldsymbol{\delta}_t\widehat{\boldsymbol{g}}_t^\top Z_{t-1}^\top + Z_{t-1}\widehat{\boldsymbol{g}}_t\boldsymbol{\delta}_t^\top + (\widehat{\boldsymbol{g}}_t^\top\widehat{\boldsymbol{g}}_t)\boldsymbol{\delta}_t\boldsymbol{\delta}_t^\top,$$

the update of F_t can therefore be implemented in $O(m^3)$ time.

Next we describe the sparse updates for $\bar{\boldsymbol{w}}_t$ and $\boldsymbol{b}_t$. Since $S_t = (t\Lambda_t)^{\frac{1}{2}}V_t$ we have

$$\begin{aligned}\boldsymbol{u}_{t+1} &= \boldsymbol{w}_t - \left(\boldsymbol{I}_d - S_t^\top H_tS_t\right)\tfrac{\boldsymbol{g}_t}{\alpha} \\ &= \underbrace{\bar{\boldsymbol{w}}_t - \tfrac{\boldsymbol{g}_t}{\alpha} - (Z_t - Z_{t-1})^\top\boldsymbol{b}_t}_{\stackrel{\text{def}}{=}\bar{\boldsymbol{u}}_{t+1}} + Z_t^\top(\underbrace{\boldsymbol{b}_t + \tfrac{1}{\alpha}F_t^\top(t\Lambda_tH_t)F_tZ_t\boldsymbol{g}_t}_{\stackrel{\text{def}}{=}\boldsymbol{b}'_{t+1}})\ .\end{aligned}$$

$Z_t - Z_{t-1}$ is sparse by construction, as is $\boldsymbol{g}_t$, implying the update scales with s, not d. Using the update for $\boldsymbol{w}_{t+1}$ in terms of $\boldsymbol{u}_{t+1}$, we have

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{u}_{t+1} - c_t(\boldsymbol{x}_{t+1} - S_t^\top H_tS_t\boldsymbol{x}_{t+1}) \\ &= \underbrace{\bar{\boldsymbol{u}}_{t+1} - c_t\boldsymbol{x}_{t+1}}_{\stackrel{\text{def}}{=}\bar{\boldsymbol{w}}_{t+1}} + Z_t^\top(\underbrace{\boldsymbol{b}'_{t+1} + c_tF_t^\top(t\Lambda_tH_t)F_tZ_t\boldsymbol{x}_{t+1}}_{\stackrel{\text{def}}{=}\boldsymbol{b}_{t+1}})\ .\end{aligned}$$

Algorithm 22 Sparse Sketched Online Newton

Input: Parameters C, α and m.

Initialize $\bar{\boldsymbol{u}} = \mathbf{0}_{d\times 1}$ and $\boldsymbol{b} = \mathbf{0}_{m\times 1}$.
$(\Lambda, F, Z, H) \leftarrow$ **SketchInit**(α, m) (Algorithm 23).
for $t = 1$ **to** T **do**
  Receive example $\boldsymbol{x}_t$.
  Projection step: compute $\widehat{\boldsymbol{x}} = FZ\boldsymbol{x}_t$ and $c = \frac{\tau_C(\bar{\boldsymbol{u}}^\top \boldsymbol{x}_t + \boldsymbol{b}^\top Z\boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t - (t-1)\widehat{\boldsymbol{x}}^\top \Lambda H \widehat{\boldsymbol{x}}}$. Obtain $\bar{\boldsymbol{w}} = \bar{\boldsymbol{u}} - c\boldsymbol{x}_t$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + c(t-1)F^\top \Lambda H \widehat{\boldsymbol{x}}$.
  Predict label $y_t = \bar{\boldsymbol{w}}^\top \boldsymbol{x}_t + \boldsymbol{b}^\top Z\boldsymbol{x}_t$ and suffer loss $\ell_t(y_t)$.
  Compute gradient $\boldsymbol{g}_t = \ell_t'(y_t)\boldsymbol{x}_t$ and the *to-sketch vector* $\widehat{\boldsymbol{g}} = \sqrt{\sigma_t + \eta_t}\boldsymbol{g}_t$.
  $(\Lambda,\ F,\ Z,\ H,\ \boldsymbol{\delta}) \leftarrow$ **SketchUpdate**$(\widehat{\boldsymbol{g}})$ (Algorithm 23).
  Update weight: $\bar{\boldsymbol{u}} = \bar{\boldsymbol{w}} - \frac{1}{\alpha}\boldsymbol{g}_t - (\boldsymbol{\delta}^\top \boldsymbol{b})\widehat{\boldsymbol{g}}$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + \frac{1}{\alpha}tF^\top \Lambda H F Z \boldsymbol{g}_t$.
end for

Algorithm 23 Sparse Oja's Sketch

Internal States: t, Λ, F, Z, H and K.

SketchInit(α, m)

Set $t = 0, \Lambda = \mathbf{0}_{m\times m}, F = K = \boldsymbol{I}_m, H = \frac{1}{\alpha}\boldsymbol{I}_m$ and let Z be any $m \times d$ matrix whose rows are orthonormal.
Return $(\Lambda,\ F,\ Z,\ H)$.

SketchUpdate$(\widehat{\boldsymbol{g}})$

Update $t \leftarrow t + 1$ and pick a diagonal learning rate matrix Γ_t.
Update $\Lambda \leftarrow (\boldsymbol{I} - \Gamma_t)\Lambda + \Gamma_t \operatorname{diag}\{FZ\widehat{\boldsymbol{g}}\}^2$.
Compute $\boldsymbol{\delta} = A^{-1}\Gamma_t FZ\widehat{\boldsymbol{g}}$.
Update $K \leftarrow K + \boldsymbol{\delta}\widehat{\boldsymbol{g}}^\top Z^\top + Z\widehat{\boldsymbol{g}}\boldsymbol{\delta}^\top + (\widehat{\boldsymbol{g}}^\top \widehat{\boldsymbol{g}})\boldsymbol{\delta}\boldsymbol{\delta}^\top$.
Update $Z \leftarrow Z + \boldsymbol{\delta}\widehat{\boldsymbol{g}}^\top$.
$(L, Q) \leftarrow$ Decompose(F, K) (Algorithm 21), so that $LQZ = FZ$ and QZ is orthogonal. Set $F = Q$.
Set $H \leftarrow \operatorname{diag}\left\{\frac{1}{\alpha + t\Lambda_{1,1}}, \cdots, \frac{1}{\alpha + t\Lambda_{m,m}}\right\}$.
Return $(\Lambda,\ F,\ Z,\ H,\ \boldsymbol{\delta})$.

c_t and $Z_t\boldsymbol{x}_{t+1}$ can both be computed in time $O(m^2 + ms)$. All the other computations scale with m and not d, so both $\bar{\boldsymbol{w}}_{t+1}$ and $\boldsymbol{b}_{t+1}$ require only $O(m^2 + ms)$ time to maintain. Furthermore, the prediction $\boldsymbol{w}_t^\top \boldsymbol{x}_t = \bar{\boldsymbol{w}}_t^\top \boldsymbol{x}_t + \boldsymbol{b}_t^\top Z_{t-1}\boldsymbol{x}_t$ can also be computed in $O(ms)$ time since $\boldsymbol{x}_t$ is sparse.

The pseudocode is presented in Algorithms 22 and 23. This is the first sparse implementation of online eigenvector computation to the best of our knowledge.

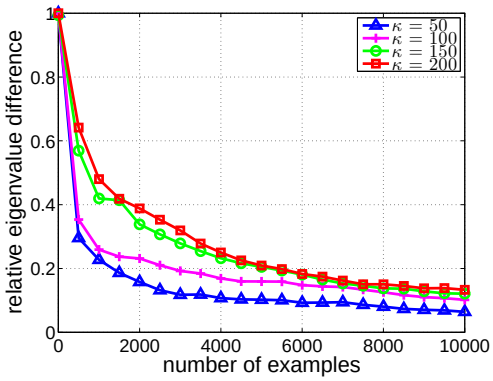


Figure 4.1: Oja's algorithm's eigenvalue recovery error

4.6 Experiments

Preliminary experiments revealed that out of our three sketching options, Oja's sketch has the best empirical performance. For more thorough evaluation, we implemented the sparse version of Oja-SON in the open source toolkit Vowpal Wabbit. We compare it with AdaGrad [33, 80] on both synthetic and real-world datasets. Each algorithm takes a stepsize parameter: $\frac{1}{\alpha}$ serves as a stepsize for Oja-SON and a scaling constant on the gradient matrix for AdaGrad. We try both the methods with the parameters set to 2^j for $j = -3, -2, \ldots, 6$ and report the best results. We keep the learning rate matrix for eigenvalues in Oja-SON fixed as $\Gamma_t = \frac{1}{t}\boldsymbol{I}_m$ throughout. All methods make one online pass over data.

4.6.1 Synthetic Datasets

To investigate Oja-SON's performance in the setting it is really designed for, we generated a range of synthetic ill-conditioned datasets as follows. We picked a random Gaussian matrix $Z \sim \mathbb{R}^{T\times d}$ (T = 10,000 and $d = 100$) and a random orthonormal basis $V \in \mathbb{R}^{d\times d}$. We chose a specific spectrum $\boldsymbol{\lambda} \in \mathbb{R}^d$ where the first $d - 10$ coordinates are 1 and the rest increase linearly to some fixed *condition number parameter* κ. We let $X = Z\text{diag}\{\boldsymbol{\lambda}\}^{\frac{1}{2}} V^\top$ be our example matrix, and created a binary classification problem with labels $y = \text{sign}(\boldsymbol{\theta}^\top \boldsymbol{x})$, where $\boldsymbol{\theta} \in \mathbb{R}^d$ is a random vector. We

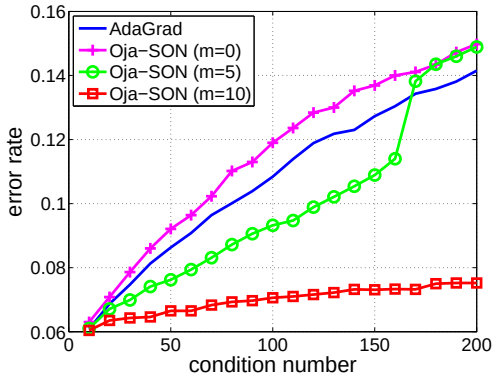
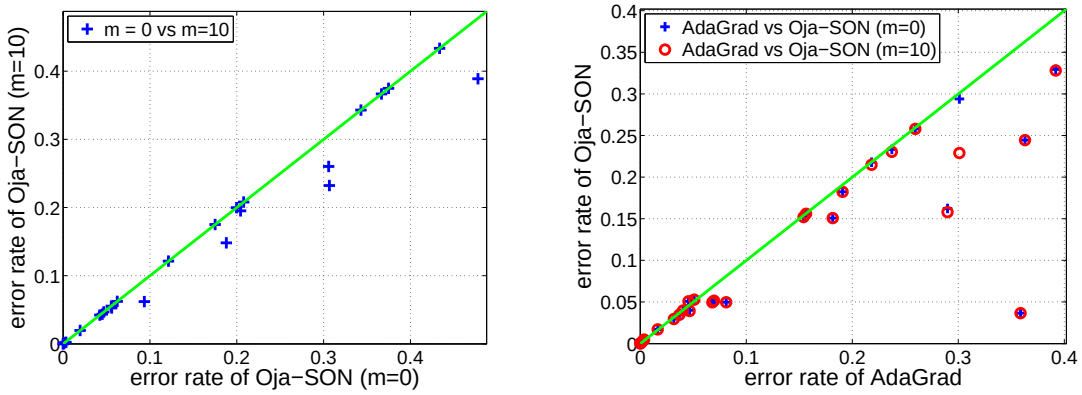


Figure 4.2: Error rate of Oja-SON compared with AdaGrad on a synthetic ill-conditioned problem

generated 20 such datasets with the same Z, V and labels y but different values of $\kappa \in \{10, 20, \ldots, 200\}$. Note that if the algorithm is truly invariant, it would have the same behavior on these 20 datasets.

Fig. 4.2 shows the final progressive error rate for AdaGrad and Oja-SON (with sketch size $m = 0, 5, 10$) as the condition number increases. As expected, the plot confirms the performance of first-order methods such as AdaGrad degrades when the data is ill-conditioned. The plot also shows that as we increase the sketch size, Oja-SON becomes more accurate: when $m = 0$ (no sketch at all), Oja-SON is vanilla gradient descent and is worse than AdaGrad as expected; when $m = 5$, the accuracy greatly improves; and finally when $m = 10$, the accuracy of Oja-SON is substantially better and hardly worsens with κ.

To further explain the effectiveness of Oja's algorithm in identifying top eigenvalues and eigenvectors, the plot in Fig. 4.1 shows the largest relative difference between the true and estimated top 10 eigenvalues as Oja's algorithm sees more data. This gap drops quickly after seeing just 500 examples.



(a) Comparison of two sketch sizes (b) Comparison against AdaGrad

Figure 4.3: Comparisons of Oja-SON with different sketch sizes and AdaGrad on real data.

4.6.2 Real-world Datasets

Next we evaluated Oja-SON on 23 benchmark datasets from the UCI and LIBSVM repository (see Appendix D for description of these datasets). In Fig. 4.3a, we show the effect of using sketched second order information, by comparing sketch size $m = 0$ and $m = 10$ for Oja-SON (concrete error rates are in Appendix D). We observe significant improvements in 5 datasets (*acoustic, census, heart, ionosphere, letter*), demonstrating the advantage of using second order information.

However, we found that Oja-SON was outperformed by AdaGrad on most datasets, mostly because the diagonal adaptation of AdaGrad greatly reduces the condition number on these datasets. Moreover, one disadvantage of SON is that for the directions not in the sketch, it is essentially doing vanilla gradient descent. We expect better results using diagonal adaptation as in AdaGrad in off-sketch directions.

To incorporate this high level idea, we performed a simple modification to Oja-SON: upon seeing example $\boldsymbol{x}_t$, we feed $D_t^{-\frac{1}{2}}\boldsymbol{x}_t$ to our algorithm instead of $\boldsymbol{x}_t$, where $D_t \in \mathbb{R}^{d\times d}$ is the diagonal part of the matrix $\sum_{\tau=1}^{t-1} \boldsymbol{g}_\tau \boldsymbol{g}_\tau^\top$.[14] With this modification,

[14] D_1 is defined as $0.1 \times \boldsymbol{I}_d$ to avoid division by zero.

Table 4.1: Largest relative error between true and estimated eigenvalues

Dataset	Relative eigenvalue difference
a9a	0.896533
australian	0.846063
breast-cancer	5.383578
diabetes	5.128097
heart	4.360492
ijcnn1	0.569727
magic04	11.479917
mushrooms	0.910814
splice	8.226752
w8a	0.954262

Oja-SON outperforms AdaGrad on most of the datasets even for $m = 0$, as shown in Fig. 4.3b (concrete error rates are again in Appendix D). The improvement on AdaGrad at $m = 0$ is surprising but not impossible as the updates are not identical– the latter is scale invariant like Ross et al. [94]. However, the diagonal adaptation already greatly reduces the condition number on all datasets except *splice*,[15] so little improvement is seen for sketch size $m = 10$ over $m = 0$. For several of the datasets, we verified the accuracy of Oja's method in computing the top-10 eigenvalues (see Table 4.1), suggesting that the lack of difference between sketch sizes is solely due to the lack of second order information after the diagonal correction.

The average running time of our algorithm when $m = 10$ is about 11 times slower than AdaGrad, matching expectations. Overall, SON can significantly outperform baselines on ill-conditioned data, while maintaining a practical computational complexity.

In summary, we have studied both the theoretical and empirical properties of sketched second order updates for online learning. In theory, RP-SON has very attractive properties of invariant updates and strong regret bounds, although it is

[15]For this dataset, the condition number before and after the diagonal adaptation are 682 and 668 respectively, and a large improvement is seen with $m = 10$.

found empirically lacking possibly due to the dependence on the rank of the gradient matrix. Oja-SON has promising empirical performance, and it would be interesting to investigate if other online PCA methods can get us the best on both the fronts in future work. It would be also desirable to automatically adapt the sketch size to further reduce the complexity overhead for well-conditioned data, while incorporating second-order information when that is helpful.

4.7 Conclusion and Future Directions

In this chapter, we studied both the theoretical and empirical properties of sketched second order updates for online learning. Starting from a novel invariant setup, we showed what the optimal but expensive algorithm is for this setting. Then by applying different sketching methods, we improved the efficiency of this algorithm and obtained linear time algorithms with provable regret guarantees. Moreover, we further improved the running time to be linear in the sparsity of the data, which is very important for applications with high dimensional sparse data. We note that our contribution of sparse sketch update might be of independent interest even before combining with the learning algorithm. Empirically, we found that Oja-SON has very promising performance compared to previous work.

For future directions, it would be desirable to incorporate more adaptivity to our algorithms. For example, can we allow the algorithm to automatically adapt the sketch size based on the data instead of fixing it in advance? Can we also allow the algorithm to automatically turn the parameters so that it achieves optimal regret bounds for both convex and strongly convex losses simultaneously?

Appendix A

Omitted Details of Chapter 2

A.1 Proof of Lemma 2

Proof. Let $F(s) = \exp\left(\frac{[s-1]_-^2}{dt}\right) + \exp\left(\frac{[s+1]_-^2}{dt}\right) - 2\exp\left(\frac{[s]_-^2}{d(t-1)}\right)$. It suffices to show

$$F(s) \leq 2(b_t - b_{t-1}) = \exp\left(\frac{4}{dt}\right) - 1,$$

which is clearly true for the following 3 cases:

$$F(s) = \begin{cases} 0 & \text{if } s > 1; \\ \exp\left(\frac{(s-1)^2}{dt}\right) - 1 < \exp\left(\frac{1}{dt}\right) - 1 & \text{if } 0 < s \leq 1; \\ \exp\left(\frac{(s-1)^2}{dt}\right) + 1 - 2\exp\left(\frac{s^2}{d(t-1)}\right) < \exp\left(\frac{4}{dt}\right) - 1 & \text{if } -1 < s \leq 0. \end{cases}$$

For the last case $s \leq -1$, if we can show that $F(s)$ is increasing in this region, then the lemma follows. Below, we show this by proving $F'(s)$ is nonnegative when $s \leq -1$.

Let $h(s,c) = \frac{\partial \exp\left(s^2/c\right)}{\partial s} = \frac{2s}{c}\exp\left(\frac{s^2}{c}\right)$. $F'(s)$ can now be written as

$$F'(s) = h(s-1,c) + h(s+1,c) - 2h(s,c) + 2(h(s,c) - h(s,c')),$$

where $c = dt$ and $c' = d(t-1)$. Next we apply (one-dimensional) Taylor expansion to $h(s-1,c)$ and $h(s+1,c)$ around s, and $h(s,c')$ around c, leading to

$$\begin{aligned} F'(s) &= \sum_{k=1}^{\infty} \frac{(-1)^k}{k!} \frac{\partial^k h(s,c)}{\partial s^k} + \sum_{k=1}^{\infty} \frac{1}{k!} \frac{\partial^k h(s,c)}{\partial s^k} - 2 \sum_{k=1}^{\infty} \frac{(c'-c)^k}{k!} \frac{\partial^k h(s,c)}{\partial c^k} \\ &= 2 \sum_{k=1}^{\infty} \left(\frac{1}{(2k)!} \frac{\partial^{2k} h(s,c)}{\partial s^{2k}} - \frac{(-d)^k}{k!} \frac{\partial^k h(s,c)}{\partial c^k} \right). \end{aligned}$$

Direct computation (see Lemma 13 below) shows that $\frac{\partial^k h(s,c)}{\partial c^k}$ and $\frac{\partial^{2k} h(s,c)}{\partial s^{2k}}$ share exact same forms only with different constants:

$$\begin{aligned} \frac{\partial^k h(s,c)}{\partial c^k} &= \exp\left(\frac{s^2}{c}\right) \sum_{j=0}^{k} (-1)^k \alpha_{k,j} \cdot \frac{s^{2j+1}}{c^{k+j+1}}, \\ \frac{\partial^{2k} h(s,c)}{\partial s^{2k}} &= \exp\left(\frac{s^2}{c}\right) \sum_{j=0}^{k} \beta_{k,j} \cdot \frac{s^{2j+1}}{c^{k+j+1}}, \end{aligned} \tag{A.1}$$

where $\alpha_{k,j}$ and $\beta_{k,j}$ are recursively defined as:

$$\begin{aligned} \alpha_{k+1,j} &= \alpha_{k,j-1} + (k+j+1)\alpha_{k,j}, \\ \beta_{k+1,j} &= 4\beta_{k,j-1} + (8j+6)\beta_{k,j} + (2j+3)(2j+2)\beta_{k,j+1}, \end{aligned} \tag{A.2}$$

with initial values $\alpha_{0,0} = \beta_{0,0} = 2$ (when $j \notin \{0,\ldots,k\}$, $\alpha_{k,j}$ and $\beta_{k,j}$ are all defined to be 0). Therefore, $F'(s)$ can be further simplified as

$$F'(s) = 2\exp\left(\frac{s^2}{c}\right) \sum_{k=1}^{\infty} \sum_{j=0}^{k} \frac{s^{2j+1}}{c^{k+j+1}} \left(\frac{\beta_{k,j}}{(2k)!} - \frac{d^k \alpha_{k,j}}{k!} \right).$$

Since s is negative, it suffices to show that $\frac{\beta_{k,j}}{(2k)!} \leq \frac{d^k \alpha_{k,j}}{k!}$ holds for all k and j, which turns out to be true as long as $d \geq 3$, as shown by induction in the technical lemma 14 below. To sum up, $\Phi_t(s-1) + \Phi_t(s+1) \leq 2\Phi_{t-1}(s)$ for all $s \in \mathbb{R}$ and $t = 2,\ldots,T$.

Finally, we need to show that Eq. (2) still holds. The inequality we just proved above implies $\sum_i \Phi_t(s_{t,i}) \leq \sum_i \Phi_{t-1}(s_{t-1,i})$ for $t = 2,\ldots,T$, as shown in Theorem

2. Thus the only thing we need to show here is the case when $t = 1$. Note that $\Phi_1(s-1) + \Phi_1(s+1) \leq 2\Phi_0(s)$ does not hold for all s apparently. However, in order to prove $\sum_i \Phi_1(s_{1,i}) \leq \sum_i \Phi_0(s_{0,i})$, we in fact only need a much weaker statement: $\Phi_1(-1) + \Phi_1(1) \leq 2\Phi_0(0)$ since $s_{0,i} \equiv 0$. This is equivalent to $\exp(1/d) - 1 \leq \exp(4/d) - 1$, which is true trivially. □

Lemma 13. *Let* $h(s,c) = \frac{2s}{c}\exp\left(\frac{s^2}{c}\right)$. *The partial derivatives of* $h(s,c)$ *satisfy Eq.* (A.1) *and* (A.2).

Proof. The base case holds trivially. Assume Eq. (A.1) holds for a fixed k. Then we have

$$\begin{aligned}
\frac{\partial^{k+1}h(s,c)}{\partial c^{k+1}} &= \exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k}(-1)^k\alpha_{k,j}\cdot\left(-\frac{s^2}{c^2}\frac{s^{2j+1}}{c^{k+j+1}} - (k+j+1)\frac{s^{2j+1}}{c^{k+j+2}}\right) \\
&= \exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k}(-1)^{k+1}\alpha_{k,j}\cdot\left(\frac{s^{2(j+1)+1}}{c^{(k+1)+(j+1)+1}} + (k+j+1)\frac{s^{2j+1}}{c^{(k+1)+j+1}}\right) \\
&= \exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k+1}(-1)^{k+1}\left(\alpha_{k,j-1} + (k+j+1)\alpha_{k,j}\right)\cdot\frac{s^{2j+1}}{c^{(k+1)+j+1}} \\
&= \exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k+1}(-1)^{k+1}\alpha_{k+1,j}\cdot\frac{s^{2j+1}}{c^{(k+1)+j+1}},
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial^{2(k+1)}h(s,c)}{\partial s^{2(k+1)}} &= \partial\left[\exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k}\beta_{k,j}\cdot\left(\frac{2s^{2j+2}}{c^{k+j+2}} + (2j+1)\frac{s^{2j}}{c^{k+j+1}}\right)\right]\Big/\partial s \\
&= \exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k}\beta_{k,j}\cdot\left(\frac{4s^{2j+3}}{c^{k+j+3}} + (8j+6)\frac{s^{2j+1}}{c^{k+j+2}} + (2j+1)2j\frac{s^{2j-1}}{c^{k+j+1}}\right) \\
&= \exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k+1}\left(4\beta_{k,j-1} + (8j+6)\beta_{k,j} + (2j+3)(2j+2)\beta_{k,j+1}\right)\cdot\frac{s^{2j+1}}{c^{k+j+2}} \\
&= \exp\left(\frac{s^2}{c}\right)\sum_{j=0}^{k+1}\beta_{k+1,j}\cdot\frac{s^{2j+1}}{c^{k+j+2}},
\end{aligned}$$

concluding the proof. □

Lemma 14. *Let $\alpha_{k,j}$ and $\beta_{k,j}$ be defined as in Eq.* (A.2)*. Then $\frac{\beta_{k,j}}{(2k)!} \leq \frac{d^k \alpha_{k,j}}{k!}$ holds for all $k \geq 0$ and $j \in \{0, \ldots, k\}$ when $d \geq 3$.*

Proof. We prove the lemma by induction on k. The base case $k = 0$ is trivial. Assume $\frac{\beta_{k,j}}{(2k)!} \leq \frac{d^k \alpha_{k,j}}{k!}$ holds for a fixed k and all $j \in \{0, \ldots, k\}$, then we have $\forall j$,

$$\begin{aligned}\frac{\beta_{k+1,j}}{(2k+2)!} &= \frac{4\beta_{k,j-1} + (8j+6)\beta_{k,j} + (2j+3)(2j+2)\beta_{k,j+1}}{(2k+2)!} \\ &\leq \frac{d^k \left(4\alpha_{k,j-1} + (8j+6)\alpha_{k,j} + (2j+3)(2j+2)\alpha_{k,j+1}\right)}{(2k+2)(2k+1)k!}.\end{aligned}$$

We need to show that the above expression is at most $d^{k+1}\alpha_{k+1,j}/(k+1)!$, which, after arrangements, is equivalent to $2\alpha_{k,j-1} + (4j+3)\alpha_{k,j} + (2j+3)(j+1)\alpha_{k,j+1} \leq d(2k+1)\alpha_{k+1,j}$. We will prove this by another induction on k. Then the lemma follows.

The base case ($k = 0$) is simplified to $6 \leq 2d$, which is true by our assumption $d \geq 3$. Assume the inequality holds for a fixed k, then by the definition of $\alpha_{k,j}$, one has

$$\begin{aligned}&2\alpha_{k+1,j-1} + (4j+3)\alpha_{k+1,j} + (2j+3)(j+1)\alpha_{k+1,j+1} \\ =\ &(2\alpha_{k,j-2} + (4j+3)\alpha_{k,j-1} + (2j+3)(j+1)\alpha_{k,j}) + \\ &(2(k+j)\alpha_{k,j-1} + (4j+3)(k+j+1)\alpha_{k,j} + (2j+3)(j+1)(k+j+2)\alpha_{k,j+1}) \\ =\ &(2\alpha_{k,j-2} + (4j-1)\alpha_{k,j-1} + (2j+1)j\alpha_{k,j}) + \\ &(k+j+2)\left(2\alpha_{k,j-1} + (4j+3)\alpha_{k,j} + (2j+3)(j+1)\alpha_{k,j+1}\right) \\ \leq\ &d(2k+1)(\alpha_{k+1,j-1} + (k+j+2)\alpha_{k+1,j}) \qquad \text{(by induction)} \\ =\ &d(2k+1)\alpha_{k+2,j} \\ \leq\ &d(2k+3)\alpha_{k+2,j},\end{aligned}$$

completing the induction. □

DGv2

Given: a loss function $L(s) = \mathbf{1}\{s \leq -R\}$.
For $t = 1, \ldots, T$:

1. The player chooses a distribution $\boldsymbol{p}_t$ over N chips.
2. The adversary randomly decides the movement of each chip $z_{t,i} \geq -1$ subject to $\mathbb{E}_t[z_{t,i}] \leq 1, \mathbb{E}_t[z_{t,i}^2] \leq 1/p_{t,i}$ and $\mathbb{E}_t[\boldsymbol{p}_t \cdot \boldsymbol{z}_t] \geq 0$.

The player suffers loss $\sum_{i=1}^{N} L(\sum_{t=1}^{T} z_{t,i})$.

Figure A.1: Protocol for a drifting game variant DGv2

Algorithm 24 A General MAB Algorithm

Input: A convex, nonincreasing, nonnegative function $\Phi_T(s) \in \mathbf{C}^2$, with nonincreasing second derivative.

for $t = T$ **to** 1 **do**
    Find a convex function $\Phi_{t-1}(s)$ s.t. the conditions of Theorem 20 hold.
end for
Set: $\boldsymbol{s}_0 = \mathbf{0}$.
for $t = 1$ **to** T **do**
    Set: $p_{t,i} \propto \Phi_t(s_{t-1,i} - 1) - \Phi_t(s_{t-1,i} + 1)$.
    Draw $i_t \sim \boldsymbol{p}_t$ and receive loss ℓ_{t,i_t}.
    Set: $z_{t,i} = \mathbf{1}\{i = i_t\} \cdot \ell_{t,i_t}/p_{t,i_t} - \ell_{t,i_t}$, $\forall i$.
    Set: $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + \boldsymbol{z}_t$.
end for

A.2 A General MAB Algorithm and Regret Bounds

This section includes the detailed protocol of DGv2 (Figure A.1), which we use to model the MAB problem and obtain a general recipe Algorithm 24 and its analysis.

Theorem 20. *Suppose $\Phi_t(s)$ is convex, twice continuously differentiable (i.e. $\Phi_t(s) \in \mathbf{C}^2$), have nonincreasing second derivative, and satisfies:*

$$\left(\tfrac{1}{2} + N\alpha_t\right)\Phi_t(s-1) + \left(\tfrac{1}{2} - N\alpha_t\right)\Phi_t(s+1) \leq \Phi_{t-1}(s), \forall s \in \mathbb{R} \tag{A.3}$$

where $\alpha_t = \frac{1}{2}\max_s \frac{\Phi_t'(s-1)}{\Phi_t(s-1)-\Phi_t(s+1)}$. *If the player's strategy is such that* $p_{t,i} \propto \Phi_t(s_{t-1,i}-1) - \Phi_t(s_{t-1,i}+1)$, *then Eq.* (2.2) *holds in expectation.*

Proof. As discussed before, the main difficulty here is the unboundedness of $z_{t,i}$. However, the expectation of $z_{t,i}$ is still in $[-1,1]$ as in DGv1. To exploit this fact, we apply Taylor's theorem to $\Phi_t(s_{t-1,i}+z_{t,i})$ to the second order term:

$$\begin{aligned}\Phi_t(s_{t,i}) &= \Phi_t(s_{t-1,i}+z_{t,i}) \\ &= \Phi_t(s_{t-1,i}) + \Phi_t'(s_{t-1,i})z_{t,i} + \tfrac{1}{2}\Phi_t''(\xi_{t,i})z_{t,i}^2 \\ &\le \Phi_t(s_{t-1,i}) + \Phi_t'(s_{t-1,i})z_{t,i} + \tfrac{1}{2}\Phi_t''(s_{t-1,i}-1)z_{t,i}^2,\end{aligned}$$

where $\xi_{t,i}$ is between $s_{t-1,i}+z_{t,i}$ and $s_{t-1,i}$, and the inequality holds because $\Phi_t''(s)$ is nonincreasing and $z_{t,i} \ge -1$ by assumption. Now taking expectation on both sides with respect to the randomness of $z_{t,i}$, using the convexity of $\Phi_t(s)$, and plugging the assumption $\mathbb{E}_t[z_{t,i}^2] \le 1/p_{t,i}$ give:

$$\begin{aligned}\mathbb{E}_t[\Phi_t(s_{t,i})] &\le \Phi_t(s_{t-1,i}) + \Phi_t'(s_{t-1,i})\mathbb{E}_t[z_{t,i}] + \tfrac{1}{2}\Phi_t''(s_{t-1,i}-1)\mathbb{E}_t[z_{t,i}^2] \\ &\le \Phi_t\left(s_{t-1,i}+\mathbb{E}_t[z_{t,i}]\right) + \tfrac{1}{2}\Phi_t''(s_{t-1,i}-1)/p_{t,i}.\end{aligned}$$

Let $w_{t,i} = \frac{1}{2}\left(\Phi_t(s_{t-1,i}-1) - \Phi_t(s_{t-1,i}+1)\right)$. Further plugging $\boldsymbol{p}_{t,i} \propto w_{t,i}$ and summing over all i, we arrive at

$$\begin{aligned}\sum_{i=1}^N \mathbb{E}_t[\Phi_t(s_{t,i})] &\le \sum_{i=1}^N \left(\Phi_t\left(s_{t-1,i}+\mathbb{E}_t[z_{t,i}]\right) + \frac{\Phi_t''(s_{t-1,i}-1)}{2w_{t,i}} \cdot \sum_{i=1}^N w_{t,i}\right) \\ &\le \sum_{i=1}^N \left(\Phi_t\left(s_{t-1,i}+\mathbb{E}_t[z_{t,i}]\right) + 2\alpha_t \sum_{i=1}^N w_{t,i}\right) \quad \text{(by the defintion of } \alpha_t\text{)} \\ &= \sum_{i=1}^N \left(\Phi_t\left(s_{t-1,i}+\mathbb{E}_t[z_{t,i}]\right) + 2N\alpha_t w_{t,i}\right).\end{aligned}$$

Since $\mathbb{E}_t[\boldsymbol{p}_t \cdot \boldsymbol{z}_t] \geq 0$ implies $\sum_{i=1}^N w_{t,i}\mathbb{E}_t[z_{t,i}] \geq 0$, we thus have

$$\begin{aligned}
\sum_{i=1}^N \mathbb{E}_t[\Phi_t(s_{t,i})] &\leq \sum_{i=1}^N \left(\Phi_t\left(s_{t-1,i} + \mathbb{E}_t[z_{t,i}]\right) + w_{t,i}\mathbb{E}_t[z_{t,i}] + 2N\alpha_t w_{t,i}\right) \\
&\leq \sum_{i=1}^N \left(\max_{z\in[-1,+1]} \left(\Phi_t\left(s_{t-1,i} + z\right) + w_{t,i}z\right) + 2N\alpha_t w_{t,i}\right) \\
&= \sum_{i=1}^N \left(\max_{z\in\{-1,+1\}} \left(\Phi_t\left(s_{t-1,i} + z\right) + w_{t,i}z\right) + 2N\alpha_t w_{t,i}\right) \\
&\qquad\qquad \text{(by the convexity of } \Phi_t(s)\text{)} \\
&= \sum_{i=1}^N \left(\left(\tfrac{1}{2} + N\alpha_t\right)\Phi_t(s_{t-1,i} - 1) + \left(\tfrac{1}{2} - N\alpha_t\right)\Phi_t(s_{t-1,i} + 1)\right) \\
&\leq \sum_{i=1}^N \Phi_{t-1}(s_{t-1,i}). \qquad\qquad \text{(by assumption)}
\end{aligned}$$

The theorem follows by taking expectation on both sides with respect to the past (i.e. the randomness of $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{t-1}$). □

Theorem 21. *For Algorithm 24, if R and ϵ are such that $\Phi_0(0) < \epsilon$ and $\Phi_T(s) \geq \mathbf{1}\{s \leq -R\}$ for all $s \in \mathbb{R}$, then $\mathbb{E}[\sum_{t=1}^T \ell_{t,i_t} - \sum_{t=1}^T \ell_{t,i_\epsilon}] < R$ for any non-oblivious adversary. Moreover, using $\Phi_T(s) = \exp(-\eta(s+R))$ (and let Eq. (A.3) hold with equality) gives exactly the EXP3 algorithm with regret $O(\sqrt{TN\ln(1/\epsilon)})$.*

Proof of Theorem 21. We first show that Algorithm 24 converts the multi-armed bandit problem to a valid instance of DGv2. It suffices to prove that $z_{t,i} = \mathbf{1}\{i = i_t\} \cdot \ell_{t,i_t}/p_{t,i_t} - \ell_{t,i_t}$ satisfies all conditions defined in DGv2, as shown below ($z_{t,i} \geq -1$ is trivial):

$$\mathbb{E}_t[z_{t,i}] = \ell_{t,i} - \boldsymbol{p}_t \cdot \boldsymbol{\ell}_t \leq 1,$$

$$\mathbb{E}_t[z_{t,i}^2] = p_{t,i}\left(\frac{\ell_{t,i}}{p_{t,i}} - \ell_{t,i}\right)^2 + \sum_{j\neq i} p_{t,j}\ell_{t,j}^2 \leq p_{t,i}\left(\frac{1}{p_{t,i}} - 1\right)^2 + \sum_{j\neq i} p_{t,j} = \frac{1}{p_{t,i}} - 1 \leq \frac{1}{p_{t,i}},$$

$$\mathbb{E}_t[\boldsymbol{p}_t \cdot \boldsymbol{z}_t] = \mathbb{E}_t\left[\ell_{t,i_t} - \sum_{j=1}^N p_{t,j}\ell_{t,i_t}\right] = 0.$$

Therefore, we can apply Theorem 20 directly, arriving at:

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[\Phi_T(s_{T,i})] \leq \cdots \leq \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[\Phi_0(s_{0,i})] = \Phi_0(0) \leq \epsilon.$$

On the other hand, by applying Jensen' inequality, we have

$$\mathbb{E}[\Phi_T(s_{T,i})] \geq \Phi_T(\mathbb{E}[s_{T,i}]) \geq \mathbf{1}\{\mathbb{E}[s_{T,i}] \leq -R\}.$$

Note that $\mathbb{E}[s_{T,i}]$ is equal to $\mathbb{E}\left[\sum_{t=1}^{T}(\ell_{t,i} - \ell_{t,i_t})\right]$. We thus know

$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\left\{\mathbb{E}\left[\sum_{t=1}^{T}(\ell_{t,i} - \ell_{t,i_t})\right] \leq -R\right\} < \epsilon,$$

which implies $\mathbb{E}\left[\sum_{t=1}^{T}\ell_{t,i_t} - \sum_{t=1}^{T}\ell_{t,i_\epsilon}\right] < R$ for any non-oblivious adversary for the exact same argument used in the proof of Theorem 2.

Finally, we show how to recover EXP3 using Algorithm 24 with input $\Phi_T(s) = \exp(-\eta(s+R))$. To compute $\Phi_t(s)$ for $t < T$, we simply use Eq. (A.3) with equality. One can verify using induction that

$$\Phi_t(s) = \exp\left(-\eta(s+R)\right)\left(\frac{e^{\eta} + e^{-\eta} + Ne^{\eta}\eta^2}{2}\right)^{T-t},$$

$$\alpha_t = \frac{1}{2}\max_s \frac{\eta^2\Phi_t(s-1)}{\Phi_t(s-1) - \Phi_t(s+1)} = \frac{e^{\eta}\eta^2}{2(e^{\eta} - e^{-\eta})},$$

$$\Phi_t'''(s) = -\eta^3\Phi_t(s) \leq 0.$$

The player's strategy is thus $\boldsymbol{p}_{t,i} \propto \exp(-\eta\sum_{\tau=1}^{t-1}\hat{\ell}_{\tau,i})$ (recall $\hat{\ell}_{t,i} = \mathbf{1}\{i = i_t\}\cdot\ell_{t,i_t}/p_{t,i_t}$ is the estimated loss), which is exactly the same as EXP3 (in fact a simplified version of the original EXP3, see for example [101]). Moreover, the regret can be computed

DGv3

Given: a compact convex set $\mathcal{K}$, a loss function $L(s) = \mathbf{1}\{s \leq -R\}$.
For $t = 1, \ldots, T$:

1. The player chooses a density function $p_t(\boldsymbol{w})$ on S.
2. The adversary decides a function $z_t(\boldsymbol{w}) : S \to [-1, 1]$ subject to $\mathbb{E}_{\boldsymbol{w} \sim p_t}[z_t(\boldsymbol{w})] \geq 0$.

The player suffers loss $\int_{\boldsymbol{w} \in \mathcal{K}} L(\sum_{t=1}^{T} z_t(\boldsymbol{w}))d\boldsymbol{w}$.

Figure A.2: Protocol for a drifting game variant DGv3

by setting $\Phi_0(0) = \epsilon$, leading to

$$\begin{aligned} R &= \frac{1}{\eta}\ln\left(\frac{1}{\epsilon}\right) + \frac{T}{\eta}\ln\left(\frac{e^{\eta} + e^{-\eta}}{2} + \frac{1}{2}Ne^{\eta}\eta^2\right) \\ &\leq \frac{1}{\eta}\ln\left(\frac{1}{\epsilon}\right) + \frac{T}{\eta}\ln\left(e^{\eta^2/2} + \frac{1}{2}Ne^{\eta}\eta^2\right) && \text{(by Hoeffding's Lemma)} \\ &\leq \frac{1}{\eta}\ln\left(\frac{1}{\epsilon}\right) + \frac{T}{\eta}\left(\frac{\eta^2}{2} + \frac{1}{2}Ne^{\eta - \frac{\eta^2}{2}}\eta^2\right) && (\ln(1+x) \leq x) \end{aligned}$$

If $\eta \leq 1$ so that $e^{\eta - \eta^2/2} \leq \sqrt{e}$, then we have $R \leq \frac{1}{\eta}\ln(\frac{1}{\epsilon}) + T\eta\left(\frac{1}{2} + \frac{N\sqrt{e}}{2}\right)$, which is $\sqrt{2T(1 + N\sqrt{e})\ln(1/\epsilon)}$ after optimally choosing η ($\eta \leq 1$ will be satisfied when T is large enough). □

A.3 A General OCO Algorithm and Regret Bounds

This section includes the detailed protocol of DGv3 (Figure A.2), which we use to model OCO and obtain a general recipe Algorithm 25 and its analysis.

Definition of ϵ-regret in the OCO setting: Let $\mathcal{K}_\epsilon \subset \mathcal{K}$ be such that the ratio of its volume and the one of $\mathcal{K}$ is ϵ and also $\sum_{t=1}^{T} f_t(\boldsymbol{w}') \leq \sum_{t=1}^{T} f_t(\boldsymbol{w})$ for all $\boldsymbol{w}' \in S_\epsilon$ and $\boldsymbol{w} \in S \backslash S_\epsilon$ (it is clear that such set exists). Then ϵ-regret is defined as $R_T^\epsilon = \sum_{t=1}^{T} f_t(\boldsymbol{w}_t) - \inf_{\boldsymbol{w} \in \mathcal{K} \backslash \mathcal{K}_\epsilon} \sum_{t=1}^{T} f_t(\boldsymbol{w})$.

Algorithm 25 A General OCO Algorithm

Input: A convex, nonincreasing, nonnegative function $\Phi_T(s)$

for $t = T$ **to** 1 **do**
    Find a convex function $\Phi_{t-1}(s)$ s.t. $\forall s$, $\Phi_t(s-1) + \Phi_t(s+1) \le 2\Phi_{t-1}(s)$.
end for
Set: $s_0(x) \equiv 0$.
for $t = 1$ **to** T **do**
    Predict $\boldsymbol{w}_t = \mathbb{E}_{\boldsymbol{w}\sim p_t}[\boldsymbol{w}]$ where p_t is such that $p_t(\boldsymbol{w}) \propto \Phi_t(s_{t-1}(\boldsymbol{w}) - 1) - \Phi_t(s_{t-1}(\boldsymbol{w}) + 1)$.
    Receive loss function f_t from the adversary.
    Set: $z_t(\boldsymbol{w}) = f_t(\boldsymbol{w}) - f_t(\boldsymbol{w}_t)$.
    Set: $s_t(\boldsymbol{w}) = s_{t-1}(\boldsymbol{w}) + z_t(\boldsymbol{w})$.
end for

Theorem 22. *For Algorithm 25, if R is such that $\Phi_T(s) \ge \mathbf{1}\{s \le -R\}$ and $\Phi_0(0) < \epsilon$, then we have $R_T^\epsilon < R$ and $R_T < R + T\epsilon^{1/d}$. Specifically, if $R = O(\sqrt{T\ln(1/\epsilon)})$, then setting $\epsilon = T^{-d}$ gives $R_T = O(\sqrt{dT\ln T})$.*

Proof. Let $u_t(\boldsymbol{w}) = \frac{1}{2}\left(\Phi_t(s_{t-1}(\boldsymbol{w}) - 1) - \Phi_t(s_{t-1}(\boldsymbol{w}) + 1)\right)$. Similarly to the expert setting, the "sum" of potentials never increases:

$$\begin{aligned}\int_{\boldsymbol{w}\in\mathcal{K}} \Phi_t(s_t(\boldsymbol{w}))d\boldsymbol{w} &\le \int_{\boldsymbol{w}\in\mathcal{K}} \left(\Phi_t(s_{t-1}(\boldsymbol{w}) + z_t(\boldsymbol{w})) + u_t(\boldsymbol{w})z_t(\boldsymbol{w})\right) d\boldsymbol{w} \\ &\le \int_{\boldsymbol{w}\in\mathcal{K}} \Phi_{t-1}(s_{t-1}(\boldsymbol{w}))d\boldsymbol{w}.\end{aligned}$$

Here, the first inequality is due to $\mathbb{E}_{\boldsymbol{w}\sim p_t}[z_t(\boldsymbol{w})] \ge 0$, and the second inequality holds for the exact same reason as in the case for the expert problem. Therefore, we have

$$\int_{\boldsymbol{w}\in\mathcal{K}} \mathbf{1}\{s_T(\boldsymbol{w}) \le -R\}d\boldsymbol{w} \le \int_{\boldsymbol{w}\in\mathcal{K}} \Phi_T(s_T(\boldsymbol{w}))d\boldsymbol{w} \le \cdots \le \int_{\boldsymbol{w}\in\mathcal{K}} \Phi_0(0)d\boldsymbol{w} < \epsilon V,$$

where V is the volume of $\mathcal{K}$. Recall the construction of $\mathcal{K}_\epsilon$. There must exist a point $\boldsymbol{w}' \in \mathcal{K}_\epsilon$ such that $s_T(\boldsymbol{w}') > -R$, otherwise $\int_{\boldsymbol{w}} \mathbf{1}\{s_T(\boldsymbol{w}) \le -R\}d\boldsymbol{w}$ would be at least ϵV. Unfolding $s_T(\boldsymbol{w}')$, we arrive at $\sum_t f_t(\boldsymbol{w}_t) - \sum_t f_t(\boldsymbol{w}') < R$. Using the fact $\sum_t f_t(\boldsymbol{w}') \le \inf_{\boldsymbol{w}\in\mathcal{K}\setminus\mathcal{K}_\epsilon} \sum_t f_t(\boldsymbol{w})$ gives the bound for ϵ-regret.

Next consider a shrunk version of $\mathcal{K}$: $\mathcal{K}'_\epsilon = \{(1-\epsilon^{\frac{1}{d}})\boldsymbol{w}^* + \epsilon^{\frac{1}{d}}\boldsymbol{w} : \boldsymbol{w} \in \mathcal{K}\}$ where $\boldsymbol{w}^* \in \arg\min_{\boldsymbol{w}} \sum_t f_t(\boldsymbol{w})$. Then $\int_{\boldsymbol{w}\in\mathcal{K}} \mathbf{1}\{s_T(\boldsymbol{w}) \le -R\} d\boldsymbol{w}$ is at least

$$\int_{\boldsymbol{w}\in\mathcal{K}'_\epsilon} \mathbf{1}\{s_T(\boldsymbol{w}) \le -R\} d\boldsymbol{w} = \epsilon \int_{\boldsymbol{w}\in\mathcal{K}} \mathbf{1}\left\{s_T\left((1-\epsilon^{\frac{1}{d}})\boldsymbol{w}^* + \epsilon^{\frac{1}{d}}\boldsymbol{w}\right) \le -R\right\} d\boldsymbol{w},$$

which, by the convexity and the boundedness of $f_t(\boldsymbol{w})$, is at least

$$\begin{aligned}
&\epsilon \int_{\boldsymbol{w}\in\mathcal{K}} \mathbf{1}\left\{\sum_{t=1}^T \left((1-\epsilon^{\frac{1}{d}})f_t(\boldsymbol{w}^*) + \epsilon^{\frac{1}{d}} f_t(\boldsymbol{w}) - f_t(\boldsymbol{w}_t)\right) \le -R\right\} d\boldsymbol{w} \\
\ge\ &\epsilon \int_{\boldsymbol{w}\in\mathcal{K}} \mathbf{1}\left\{\sum_{t=1}^T \left(f_t(\boldsymbol{w}^*) - f_t(\boldsymbol{w}_t)\right) \le -R - T\epsilon^{\frac{1}{d}}\right\} d\boldsymbol{w} \\
=\ &\epsilon V \cdot \mathbf{1}\left\{\sum_{t=1}^T \left(f_t(\boldsymbol{w}^*) - f_t(\boldsymbol{w}_t)\right) \le -R - T\epsilon^{\frac{1}{d}}\right\}.
\end{aligned}$$

Following the previous discussion, the expression in the last line above is strictly less than $\epsilon V \cdot$, which means that the value of the indicator function has to be 0, namely, $R_T(\boldsymbol{w}_{1:T}, f_{1:T}) < R + T\epsilon^{1/d}$. □

A.4 Proof of Lemma 5

Proof. We prove the lemma by induction on T. The base case $T = 1$ is trivial. Now assume the lemma is true for any number of v's smaller than T. Suppose $a_j, U_j = [s_j, t_j]$ $(j = 1, \ldots, M)$ is an optimal construction. Let $t^* \in \arg\min_t v_t$. Without loss of generality assume t^* belongs and only belongs to $U_1, \ldots, U_k$ for some k. By the definition of h, we must have

$$h(\{v_1, \ldots, v_T\}) = v_{t^*} + h(\{v_1 - w_1, \ldots, v_{t^*-1} - w_{t^*-1}\}) + h(\{v_{t^*+1} - w_{t^*+1}, \ldots, v_T - w_T\}),$$

where we define $w_t = \sum_{j=1}^k a_j \mathbf{1}\{t \in U_j\}$ and $h(\emptyset) = 0$. (Note that we also use the fact that $v_{t^*} = \min_t v_t$ so that $v_t - w_t \ge v_t - v_{t^*}$ is always nonnegative as required.) Now

the key idea is to show that "extending" each U_j ($j \in [k]$) to the entire time interval $[1, T]$ does not increase the objective value in some sense. First consider extending U_1. By the inductive assumption, we have

$$\begin{aligned}
&h(\{v_1 - w_1 - a_1, \ldots, v_{s_1-1} - w_{s_1-1} - a_1, v_{s_1} - w_{s_1}, \ldots, v_{t^*-1} - w_{t^*-1}\} \\
&= (v_1 - w_1 - a_1) + \left(\sum_{t=2}^{s_1-1} [(v_t - w_t - a_1) - (v_{t-1} - w_{t-1} - a_1)]_+ \right) \\
&\quad + [(v_{s_1} - w_{s_1}) - (v_{s_1-1} - w_{s_1-1} - a_1)]_+ + \left(\sum_{t=s_1+1}^{t^*-1} [(v_t - w_t) - (v_{t-1} - w_{t-1})]_+ \right) \\
&= h(\{v_1 - w_1, \ldots, v_{t^*-1} - w_{t^*-1}\}) + [(v_{s_1} - w_{s_1}) - (v_{s_1-1} - w_{s_1-1} - a_1)]_+ \\
&\quad - ([(v_{s_1} - w_{s_1}) - (v_{s_1-1} - w_{s_1-1})]_+ + a_1) \\
&\leq h(\{v_1 - w_1, \ldots, v_{t^*-1} - w_{t^*-1}\}),
\end{aligned}$$

where the inequality follows from the fact $[b + c]_+ \leq [b]_+ + c$. Similarly, we also have

$$\begin{aligned}
&h(\{v_{t^*+1} - w_{t^*+1}, \ldots, v_{t_1} - w_{t_1}, v_{t_1+1} - w_{t_1+1} - a_1, \ldots, v_T - w_T - a_1\} \\
&\leq h(\{v_{t^*+1} - w_{t^*+1}, \ldots, v_T - w_T\}).
\end{aligned}$$

By extending $U_2, \ldots, U_k$ one by one in a similar way, we arrive at

$$h(\{v_1, \ldots, v_T\}) \geq v_{t^*} + h(\{v_1 - v_{t^*}, \ldots, v_{t^*-1} - v_{t^*}\}) + h(\{v_{t^*+1} - v_{t^*}, \ldots, v_T - v_{t^*}\}).$$

Notice that the right hand side of the above inequality admits a valid construction for $\{v_1, \ldots, v_T\}$: an interval $U = [1, T]$ with weight $a = v_{t^*}$, together with the optimal constructions for $\{v_1 - v_{t^*}, \ldots, v_{t^*-1} - v_{t^*}\}$ and $\{v_{t^*+1} - v_{t^*}, \ldots, v_T - v_{t^*}\}$. Therefore, by the optimality of h, the above inequality must be an equality. By using the inductive

assumption again, we thus have

$$
\begin{aligned}
& h(\{v_1, \ldots, v_T\}) \\
&= v_{t^*} + h(\{v_1 - v_{t^*}, \ldots, v_{t^*-1} - v_{t^*}\}) + h(\{v_{t^*+1} - v_{t^*}, \ldots, v_T - v_{t^*}\}) \\
&= v_{t^*} + \left(v_1 - v_{t^*} + \sum_{t=2}^{t^*-1} [v_t - v_{t-1}]_+ \right) + \left(v_{t^*+1} - v_{t^*} + \sum_{t=t^*+2}^{T} [v_t - v_{t-1}]_+ \right) \\
&= \left(\sum_{t=1}^{t^*-1} [v_t - v_{t-1}]_+ \right) + \left(\sum_{t=t^*+1}^{T} [v_t - v_{t-1}]_+ \right) \\
&= \sum_{t=1}^{T} [v_t - v_{t-1}]_+,
\end{aligned}
$$

where the last step follows from $[v_{t^*} - v_{t^*-1}]_+ = 0$ by the definition of t^*. This completes the proof. $\square$

Appendix B

Omitted Details of Chapter 3

B.1 Proof of Lemma 6

Proof. Fix a weak learner, say WL^i. Let

$$U = \{t : (\boldsymbol{x}_t, y_t) \text{ passed to } \mathrm{WL}^i\}.$$

Since inequality (3.1) holds even for *adaptive* adversaries, with high probability we have

$$\sum_{t=1}^{T} \mathbf{1}\{\mathrm{WL}^i(\boldsymbol{x}_t) \neq y_t\}\mathbf{1}\{t \in U\} \leq (\tfrac{1}{2} - \gamma)|U| + S. \tag{B.1}$$

Now fix the internal randomness of WL^i. Note that $\mathbb{E}_t[\mathbf{1}\{t \in U\}] = p_t^i = \frac{w_t^i}{\|\boldsymbol{w}^i\|_\infty}$, where $\mathbb{E}_t[\cdot]$ is the expectation conditioned on all the randomness of the booster until (and not including) round t. Define $\sigma = \sum_{t=1}^{T} p_t^i$.

We now show using martingale concentration bounds that with high probability,

$$\sum_{t=1}^{T} \mathbf{1}\{\mathrm{WL}^i(\boldsymbol{x}_t) \neq y_t\}p_t^i \leq \sum_{t=1}^{T} \mathbf{1}\{\mathrm{WL}^i(\boldsymbol{x}_t) \neq y_t\}\mathbf{1}\{t \in U\} + \tilde{O}\left(\sqrt{\sigma}\right) \tag{B.2}$$

and

$$|U| \leq \sigma + \tilde{O}\left(\sqrt{\sigma}\right). \tag{B.3}$$

Here, the $\tilde{O}(\cdot)$ notation suppresses dependence on $\log\log(T)$.

To prove inequality (B.2), consider the martingale difference sequence

$$X_t = \mathbf{1}\{\mathrm{WL}^i(\boldsymbol{x}_t) \neq y_t\}\mathbf{1}\{t \in U\} - \mathbf{1}\{\mathrm{WL}^i(\boldsymbol{x}_t) \neq y_t\}p_t^i.$$

Note that $|X_t| \leq 1$, and the conditional variance satisfies

$$\mathrm{Var}_t[X_t|X_1, X_2, \ldots, X_{t-1}] \leq p_t^i.$$

Then, by Lemma 2 of Bartlett et al. [11], for any $\delta < 1/e$ and assuming $T \geq 4$, with probability at least $1 - \log_2(T)\delta$, we have

$$\sum_{t=1}^{T} X_t \leq 2\max\left\{2\sqrt{\sigma}, \sqrt{\ln(\tfrac{1}{\delta})}\right\}\sqrt{\ln(\tfrac{1}{\delta})} = \tilde{O}(\sqrt{\sigma}),$$

by choosing $\delta \ll \frac{1}{\log_2(T)}$. This implies inequality (B.2). Inequality (B.3) is proved similarly. Note that these high probability bounds are conditioned on the internal randomness of WL^i. By taking an expectation of this conditional probability over the internal randomness of WL^i, we conclude that inequalities (B.2) and (B.3) hold with high probability unconditionally.

Via a union bound, inequalities (B.1), (B.2) and (B.3) all hold simultaneously with high probability, which implies that

$$\sum_{t=1}^{T} \mathbf{1}\{\mathrm{WL}^i(\boldsymbol{x}_t) \neq y_t\}p_t^i \leq (\tfrac{1}{2} - \gamma)\sigma + S + \tilde{O}\left(\sqrt{\sigma}\right). \tag{B.4}$$

Using the facts that $p_t^i = \frac{w_t^i}{\|\boldsymbol{w}^i\|_\infty}$ and $\mathbf{1}\{\mathrm{WL}^i(\boldsymbol{x}_t) \neq y_t\} = \frac{1-z_t^i}{2}$ and simplifying, we get

$$\begin{aligned}
\boldsymbol{w}^i \cdot \boldsymbol{z}^i &\geq 2\gamma\|\boldsymbol{w}^i\|_1 - 2S\|\boldsymbol{w}^i\|_\infty - \tilde{O}(\sqrt{\|\boldsymbol{w}^i\|_1\|\boldsymbol{w}^i\|_\infty}) \\
&\geq 2\gamma\|\boldsymbol{w}^i\|_1 - 2S\|\boldsymbol{w}^i\|_\infty - \gamma\|\boldsymbol{w}^i\|_1 - \tilde{O}(\tfrac{\|\boldsymbol{w}^i\|_\infty}{\gamma}) \\
&= \gamma\|\boldsymbol{w}^i\|_1 - 2S\|\boldsymbol{w}^i\|_\infty - \tilde{O}(\tfrac{\|\boldsymbol{w}^i\|_\infty}{\gamma}).
\end{aligned}$$

The second inequality above follows from the arithmetic mean-geometric mean inequality. This gives us the desired bound. The high probability bound for all weak learners follows by taking a union bound. □

B.2 Variants and Generalizations of Online Gradient Boosting

We provide the omitted details of two variants of our boosting algorithms: (a) a variant that work with a different kind of base learner which does greedy fitting, and (b) a variant that incorporates a scaling of the base functions to improves performance. We also show how our algorithmic technique can be used to improve the convergence speed for batch boosting.

B.2.1 Fitting to actual loss functions

The choice of an online *linear* learning algorithm over the base function class in our algorithms was made to ease the analysis. In practice, it is more common to have an online algorithm which produce predictions with comparable accuracy to the best function in hindsight for the *actual* sequence of loss functions. In particular, a common heuristic in boosting algorithms such as the original gradient boosting algorithm by Friedman [41] or the matching pursuit algorithm of Mallat and Zhang [76] is to build a linear combination of base functions by iteratively augmenting the current linear

combination by greedily choosing a base function and a step size for it that minimizes the loss with respect to the residual label. Indeed, the boosting algorithm of Zhang and Yu [111] also uses this kind of greedy fitting algorithm as the base learner.

In the online setting, we can model greedy fitting as follows. We first fix a step size $\alpha \geq 0$ in advance. Then, in each round t, the base learner $\mathcal{A}$ receives not only the example $\boldsymbol{x}_t$, but also an *offset* $\boldsymbol{y}'_t \in \mathbb{R}^d$ for the prediction, and produces a prediction $\mathcal{A}(\boldsymbol{x}_t) \in \mathbb{R}^d$, after which it receives the loss function ℓ_t and suffers loss $\ell_t(\boldsymbol{y}'_t + \alpha\mathcal{A}(\boldsymbol{x}_t))$. The predictions of $\mathcal{A}$ satisfy

$$\sum_{t=1}^{T} \ell_t(\boldsymbol{y}'_t + \alpha\mathcal{A}(\boldsymbol{x}_t)) \;\leq\; \inf_{f\in\mathcal{F}} \sum_{t=1}^{T} \ell_t(\boldsymbol{y}'_t + \alpha f(\boldsymbol{x}_t)) + R(T),$$

where R is the regret. We now describe how our algorithms can be made to work with this kind of base learner as well.

Assume that for some known parameter $B > 0$, we have $\|\boldsymbol{y}'_t\| \leq B$, for all t. Let $B' = B + \alpha D$, and assume that the loss functions ℓ_t are $L_{B'}$ Lipschitz and $\beta_{B'}$ smooth on $\mathbb{B}^d(B')$. Then using the convexity and smoothness of the loss functions, we have $\ell_t(\boldsymbol{y}'_t + \alpha\mathcal{A}(\boldsymbol{x}_t)) \geq \ell_t(\boldsymbol{y}'_t) + \alpha\nabla\ell_t(\boldsymbol{y}'_t)\cdot\mathcal{A}(\boldsymbol{x}_t)$ and $\ell_t(\boldsymbol{y}'_t + \alpha f(\boldsymbol{x}_t)) \leq \ell_t(\boldsymbol{y}'_t) + \alpha\nabla\ell_t(\boldsymbol{y}'_t)\cdot f(\boldsymbol{x}_t) + \frac{\beta_{B'}\alpha^2}{2}\|f(\boldsymbol{x}_t)\|^2$. Plugging these bounds into the above regret bound we get, for any $f \in \mathcal{F}$,

$$\sum_{t=1}^{T} \nabla\ell_t(\boldsymbol{y}'_t)\cdot\mathcal{A}(\boldsymbol{x}_t) \;\leq\; \sum_{t=1}^{T} \left(\nabla\ell_t(\boldsymbol{y}'_t)\cdot f(\boldsymbol{x}_t) + \frac{\beta_{B'}}{2}\alpha\|f(\boldsymbol{x}_t)\|^2\right) + \frac{1}{\alpha}R(T).$$

Since $\|f(\boldsymbol{x}_t)\| \leq D$, setting $\alpha = \sqrt{\frac{2R(T)}{\beta_{B'}D^2T}}$, we conclude that

$$\sum_{t=1}^{T} \nabla\ell_t(\boldsymbol{y}'_t)\cdot\mathcal{A}(\boldsymbol{x}_t) \;\leq\; \sum_{t=1}^{T} \nabla\ell_t(\boldsymbol{y}'_t)\cdot f(\boldsymbol{x}_t) + \sqrt{2\beta_{B'}D^2TR(T)}. \tag{B.5}$$

This regret bound is sublinear in T if $R(T)$ is sublinear. We can obtain a better regret bound if we assume that $R(T)$ scales linearly with α: this is a natural assumption since the functions $\ell_t(\boldsymbol{y}'_t + \alpha\boldsymbol{y})$ are $\alpha L_{B'}$ Lipschitz in the prediction $\boldsymbol{y}$. In this case, the regret bound $R(T) = \alpha R'(T)$ for some fixed $R' : \mathbb{N} \to \mathbb{R}_+$ indepedent of α, and we can choose $\alpha = \frac{2R'(T)}{\beta_{B'} D^2 T}$ so that

$$\sum_{t=1}^{T} \nabla\ell_t(\boldsymbol{y}'_t) \cdot \mathcal{A}(\boldsymbol{x}_t) \;\leq\; \sum_{t=1}^{T} \nabla\ell_t(\boldsymbol{y}'_t) \cdot f(\boldsymbol{x}_t) + 2R'(T). \tag{B.6}$$

Either the bound (B.5) or (B.6) suffices for the analysis of our boosting algorithms to go through: to use this kind of base learner $\mathcal{A}$, we again keep N copies $\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^N$ with a suitably chosen step size α, and simply change line 11 of Algorithm 10 and line 13 of Algorithm 9 to pass the offset $\boldsymbol{y}'_t = \boldsymbol{y}_t^{i-1}$ to $\mathcal{A}^i$.

B.2.2 Improving the regret bound via scaling

Given an online linear learning algorithm $\mathcal{A}$ over the function class $\mathcal{F}$ with regret R, then for any scaling parameter $\lambda > 0$, we trivially obtain an online linear learning algorithm, denoted $\lambda\mathcal{A}$, over a λ-scaling of $\mathcal{F}$, viz. $\lambda\mathcal{F} := \{\lambda f \mid f \in \mathcal{F}\}$, simply by multiplying the predictions of $\mathcal{A}$ by λ. The corresponding regret scales by λ as well, i.e. it becomes λR.

The performance of Algorithm 9 can be improved by using such an online linear learning algorithm over $\lambda\mathcal{F}$ for a suitably chosen scaling $\lambda \geq 1$ of the function class $\mathcal{F}$. Let $\|f\|'_1 = \max\{1, \frac{\|f\|_1}{\lambda}\}$ be the 1-norm of f measured with respect to $\lambda\mathcal{F}$, and $B' = \min\{\eta N \lambda D,\ \inf\{b \geq \lambda D :\ \eta\beta_b b^2 \geq \epsilon_b \lambda D\}\}$. Then we immediately get the following corollary of Theorem 13:

Corollary 4. *For any $f \in$ span$(\mathcal{F})$, let $\Delta_0 = \sum_{t=1}^{T} \ell_t(0) - \ell_t(f(\boldsymbol{x}_t))$. Algorithm 9, using $\lambda\mathcal{A}$ as the online linear algorithm over $\lambda\mathcal{F}$, is an online learning algorithm for span$(\mathcal{F})$ for losses in $\mathcal{C}$ with the following regret bound for any $f \in$ span$(\mathcal{F})$:*

$$R'_f(T) \leq \left(1 - \frac{\eta}{\|f\|'_1}\right)^N \Delta_0 + 3\eta\beta_{B'}B'^2\|f\|'_1 T + L_{B'}\|f\|'_1\lambda R(T) + 2L_{B'}B'\|f\|'_1\sqrt{T}.$$

Choosing large values of λ implies that $\|f\|'_1$ can be significantly smaller than $\|f\|_1$. But B' becomes bigger than B, and correspondingly, the parameters $\beta_{B'}$ and $L_{B'}$ become bigger than β_B and L_B respectively. Also, the (lower order) dependence on the regret term $R(T)$ increases by a factor of λ.

However, it turns out (see the end of section 3.2.2) that in several common applications of the algorithm, B' can be set to be equal to B or the increase from B is a very slow growing function of λ, such as $\log(\lambda)$. In such situations choosing larger values of λ leads to improvement in the higher order terms of the regret bound, while making the lower order term (i.e. $L_{B'}\|f\|'_1\lambda R(T)$) worse; overall the regret bound can be improved by choosing a suitably large scaling factor λ to balance between the two.

B.2.3 Improvements for batch boosting

Our algorithmic technique can be used to improve convergence speed for batch boosting as well, in the setup considered by Zhang and Yu [111]. Since the focus of this paper is on online boosting, we give a high level comparison of the bounds here, making some simplifying assumptions to ease the technical details, using our notation as much as possible.

In the setup of Zhang and Yu [111], we have a base set of real valued functions $\mathcal{F}$, which we assume is symmetric and contains the zero function, $\mathbf{0}$. Then $\text{span}(\mathcal{F})$ is a linear function space, and let $\|\cdot\|$ be some norm defined on $\text{span}(\mathcal{F})$. For clarity of presentation, we assume that for any $f \in \mathcal{F}$, we have $\|f\| \leq 1$. This implies that for any $f \in \text{span}(\mathcal{F})$, we have $\|f\| \leq \|f\|_1$.

The goal is to minimize a given convex functional $\ell : \text{span}(\mathcal{F}) \to \mathbb{R}$ over its domain, $\text{span}(\mathcal{F})$. We assume, for simplicity, that ℓ is β-smooth over $\text{span}(\mathcal{F})$ under the norm $\|\cdot\|$, i.e. for any $f, f' \in \text{span}(\mathcal{F})$, we have

$$\ell(f') \;\leq\; \ell(f) + \nabla\ell(f) \cdot (f' - f) + \frac{\beta}{2}\|f - f'\|^2.$$

Zhang and Yu [111] assume[16] that we have access to a base learning algorithm $\mathcal{A}$ that, given any $f \in \text{span}(\mathcal{F})$ and a step size $\eta \geq 0$ can find a function $g \in \mathcal{F}$ that minimizes $\ell(f + \eta g)$. We denote the output of $\mathcal{A}$ by $\mathcal{A}(f, \eta)$.

Given such a base learning algorithm, and a sequence of step sizes $\eta_1, \eta_2, \ldots$, the boosting algorithm of Zhang and Yu [111] computes a sequence of functions $f_0, f_1, f_2, \ldots \in \text{span}(\mathcal{F})$ via greedy fitting as follows: f_0 is set to $\mathbf{0}$, and for any $i \geq 1$,

$$f_i \;:=\; f_{i-1} + \eta_i \mathcal{A}(f_{i-1}, \eta_i).$$

Define $s_0 = 1$ and $s_i = s_{i-1} + \eta_i$ for any $i \geq 1$.

For any $f \in \text{span}(f)$, for $i = 1, 2, \ldots$, let $\Delta_i = \ell(f_i) - \ell(f)$ denote the optimization errors of the function f_i. Zhang and Yu [111] prove that for any $N \in \mathbb{N}$, we have

$$\Delta_N \;\leq\; \frac{s_0 + \|f\|_1}{s_N + \|f\|_1}\Delta_0 + \sum_{i=1}^{N} \frac{s_i + \|f\|_1}{s_N + \|f\|_1} \cdot \frac{\beta}{2}\eta_i^2. \tag{B.7}$$

Using the techniques in this paper, we can define a different boosting algorithm which works as follows. Given the same sequence of step sizes $\eta_1, \eta_2, \ldots$ as above, we set $f_0 = \mathbf{0}$, and for any $i \geq 1$,

$$f_i \;:=\; (1 - \sigma_i\eta_i) f_{i-1} + \eta_i \mathcal{A}(f_{i-1}, \eta_i),$$

[16] This is a slight simplification of the base learning algorithm considered in [111], which also performs a search over the step size η. Also, the analysis in [111] allows some optimization error for the base learning algorithm; to simplify the comparison we assume this error is 0.

where

$$\sigma_i \;:=\; \begin{cases} 1 & \text{if } \nabla\ell(f_{i-1}) \cdot f_{i-1} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

We can analyze this algorithm along the lines of the proof of Theorem 13. First, let $g_i = \mathcal{A}(f_{i-1}, \eta_i)$. Then for $g \in \mathcal{F}$, we have $\ell(f_{i-1} + \eta_i g_i) \leq \ell(f_{i-1} + \eta_i g)$, and by the convexity and β-smoothness of ℓ, we conclude that

$$\nabla\ell(f_{i-1}) \cdot g_i \;\leq\; \nabla\ell(f_{i-1}) \cdot g + \frac{\beta}{2}\eta_i.$$

Using this fact and following the proof of Theorem 13, we get the following bound on the optimization error $\Delta_i = \ell(f_i) - \ell(f)$ of the function f_i:

$$\Delta_N \;\leq\; \exp\left(-\frac{s_N - s_0}{\|f\|_1}\right)\Delta_0 + \sum_{i=1}^{N} \exp\left(-\frac{s_N - s_i}{\|f\|_1}\right) \cdot \frac{\beta}{2}\eta_i^2(s_i^2 + 1). \tag{B.8}$$

We can compare our bound (B.8) to the bound (B.7) of Zhang and Yu [111], by comparing corresponding terms in the bound. For each term, we can calculate how large s_N needs to be for the term to be reduced to less than some given bound ϵ. To reduce the first term to less than ϵ our algorithm needs $s_N \geq \|f\|_1 \log(\frac{\Delta_0}{\epsilon}) + s_0$, whereas the algorithm of Zhang and Yu [111] needs $s_N \geq (\frac{\Delta_0}{\epsilon})(s_0 + \|f\|_1) - \|f\|_1$. As for the second term, to reduce the i-th term in the sum to less than ϵ, our algorithm needs $s_N \geq \|f\|_1 \log(\frac{\beta\eta_i^2(s_i^2+1)}{2\epsilon}) + s_i$, whereas the algorithm of Zhang and Yu [111] needs $s_N \geq (\frac{\beta\eta_i^2}{2\epsilon})(s_i + \|f\|_1) - \|f\|_1$. Since in either case, the dependence on ϵ is $\log(\frac{1}{\epsilon})$ for our algorithm, whereas it is $\frac{1}{\epsilon}$ for the algorithm of Zhang and Yu [111], we conclude that our algorithm converges exponentially faster.

Appendix C

Omitted Details of Chapter 4

C.1 Projection

We prove a more general version of Lemma 11 which does not require invertibility of the matrix A here.

Lemma 15. *For any $\boldsymbol{x} \neq \boldsymbol{0}, \boldsymbol{u} \in \mathbb{R}^{d\times 1}$ and positive semidefinite matrix $A \in \mathbb{R}^{d\times d}$, we have*

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}:|\boldsymbol{w}^\top \boldsymbol{x}|\leq C} \|\boldsymbol{w}-\boldsymbol{u}\|_A = \begin{cases} \boldsymbol{u} - \frac{\tau_C(\boldsymbol{u}^\top \boldsymbol{x})}{\boldsymbol{x}^\top A^\dagger \boldsymbol{x}} A^\dagger \boldsymbol{x} & \text{if } \boldsymbol{x} \in \text{range}(A) \\ \boldsymbol{u} - \frac{\tau_C(\boldsymbol{u}^\top \boldsymbol{x})}{\boldsymbol{x}^\top (\boldsymbol{I}-A^\dagger A)\boldsymbol{x}} (\boldsymbol{I}-A^\dagger A)\boldsymbol{x} & \text{if } \boldsymbol{x} \notin \text{range}(A) \end{cases}$$

where $\tau_C(y) = sign(y)\max\{|y|-C, 0\}$ and $A^\dagger$ is the Moore-Penrose pseudoinverse of A. (Note that when A is rank deficient, this is one of the many possible solutions.)

Proof. First consider the case when $\boldsymbol{x} \in \text{range}(A)$. If $|\boldsymbol{u}^\top \boldsymbol{x}| \leq C$, then it is trivial that $\boldsymbol{w}^* = \boldsymbol{u}$. We thus assume $\boldsymbol{u}^\top \boldsymbol{x} \geq C$ below (the last case $\boldsymbol{u}^\top \boldsymbol{x} \leq -C$ is similar). The Lagrangian of the problem is

$$L(\boldsymbol{w}, \lambda_1, \lambda_2) = \frac{1}{2}(\boldsymbol{w}-\boldsymbol{u})^\top A(\boldsymbol{w}-\boldsymbol{u}) + \lambda_1(\boldsymbol{w}^\top \boldsymbol{x} - C) + \lambda_2(\boldsymbol{w}^\top \boldsymbol{x} + C)$$

where $\lambda_1 \geq 0$ and $\lambda_2 \leq 0$ are Lagrangian multipliers. Since $\boldsymbol{w}^\top \boldsymbol{x}$ cannot be C and $-C$ at the same time, The complementary slackness condition implies that either $\lambda_1 = 0$ or $\lambda_2 = 0$. Suppose the latter case is true, then setting the derivative with respect to $\boldsymbol{w}$ to 0, we get $\boldsymbol{w}^* = \boldsymbol{u} - \lambda_1 A^\dagger \boldsymbol{x} + (\boldsymbol{I} - A^\dagger A)\boldsymbol{z}$ where $\boldsymbol{z} \in R^{d \times 1}$ can be arbitrary. However, since $A(\boldsymbol{I} - A^\dagger A) = 0$, this part does not affect the objective value at all and we can simply pick $z = 0$ so that w^* has a consistent form regardless of whether A is full rank or not. Now plugging $\boldsymbol{w}^*$ back, we have

$$L(\boldsymbol{w}^*, \lambda_1, 0) = -\frac{{\lambda_1}^2}{2} \boldsymbol{x}^\top A^\dagger \boldsymbol{x} + \lambda_1 (\boldsymbol{u}^\top \boldsymbol{x} - C)$$

which is maximized when $\lambda_1 = \frac{\boldsymbol{u}^\top \boldsymbol{x} - C}{\boldsymbol{x}^\top A^\dagger \boldsymbol{x}} \geq 0$. Plugging this optimal λ_1 into $\boldsymbol{w}^*$ gives the stated solution. On the other hand, if $\lambda_1 = 0$ instead, we can proceed similarly and verify that it gives a smaller dual value (0 in fact), proving the previous solution is indeed optimal.

We now move on to the case when $\boldsymbol{x} \notin \text{range}(A)$. First of all the stated solution is well defined since $\boldsymbol{x}^\top (\boldsymbol{I} - A^\dagger A)\boldsymbol{x}$ is nonzero in this case. Moreover, direct calculation shows that $\boldsymbol{w}^*$ is in the valid space: $|{\boldsymbol{w}^*}^\top \boldsymbol{x}| = |\boldsymbol{u}^\top \boldsymbol{x} - \tau_C(\boldsymbol{u}^\top \boldsymbol{x})| \leq C$, and also it gives the minimal possible distance value $\|\boldsymbol{w}^* - \boldsymbol{u}\|_A = 0$, proving the lemma. □

C.2 A Truly Invariant Algorithm

In this section we discuss how to make our adaptive online Newton algorithm truly invariant to invertible linear transformations. To achieve this, we set $\alpha = 0$ and replace A_t^{-1} with the Moore-Penrose pseudoinverse $A_t^\dagger$: [17]

$$\begin{aligned} \boldsymbol{u}_{t+1} &= \boldsymbol{w}_t - A_t^\dagger \boldsymbol{g}_t, \\ \boldsymbol{w}_{t+1} &= \operatorname*{argmin}_{\boldsymbol{w} \in \mathcal{K}_{t+1}} \|\boldsymbol{w} - \boldsymbol{u}_{t+1}\|_{A_t} \ . \end{aligned} \tag{C.1}$$

[17] See Appendix C.1 for the closed form of the projection step.

When written in this form, it is not immediately clear that the algorithm has the invariant property. However, one can rewrite the algorithm in a mirror descent form:

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \operatorname*{argmin}_{\boldsymbol{w}\in\mathcal{K}_{t+1}} \left\|\boldsymbol{w} - \boldsymbol{w}_t + A_t^\dagger \boldsymbol{g}_t\right\|_{A_t}^2 \\ &= \operatorname*{argmin}_{\boldsymbol{w}\in\mathcal{K}_{t+1}} \|\boldsymbol{w} - \boldsymbol{w}_t\|_{A_t}^2 + 2(\boldsymbol{w} - \boldsymbol{w}_t)^\top A_t A_t^\dagger \boldsymbol{g}_t \\ &= \operatorname*{argmin}_{\boldsymbol{w}\in\mathcal{K}_{t+1}} \|\boldsymbol{w} - \boldsymbol{w}_t\|_{A_t}^2 + 2\boldsymbol{w}^\top \boldsymbol{g}_t\end{aligned}$$

where we use the fact that $\boldsymbol{g}_t$ is in the range of A_t in the last step. Now suppose all the data $\boldsymbol{x}_t$ are transformed to $M\boldsymbol{x}_t$ for some unknown and invertible matrix M, then one can verify that all the weights will be transformed to $M^{-T}\boldsymbol{w}_t$ accordingly, ensuring the prediction to remain the same.

Moreover, the regret bound of this algorithm can be bounded as below. First notice that even when A_t is rank deficient, the projection step still ensures the following: $\|\boldsymbol{w}_{t+1} - \boldsymbol{w}\|_{A_t}^2 \le \|\boldsymbol{u}_{t+1} - \boldsymbol{w}\|_{A_t}^2$, which is proven in [57, Lemma 8]. Therefore, the entire proof of Theorem 16 still holds after replacing A_t^{-1} with $A_t^\dagger$, giving the regret bound:

$$\frac{1}{2}\sum_{t=1}^{T} \boldsymbol{g}_t^\top A_t^\dagger \ \boldsymbol{g}_t + 2(CL)^2\eta_t \ . \tag{C.2}$$

Similar to the discussion in Section 4.3.2, the key now is to bound the term $\sum_{t=1}^T \boldsymbol{g}_t^\top \widehat{A}_t^\dagger \ \boldsymbol{g}_t$ where we define $\widehat{A}_t = \sum_{s=1}^t \boldsymbol{g}_s\boldsymbol{g}_s^\top$. In order to do this, we proceed similarly to the proof of [24, Theorem 4.2] to show that this term is of order $O(d^2 \ln T)$ in the worst case.

Theorem 23. *Let λ^* be the minimum among the smallest nonzero eigenvalues of $\widehat{A}_t$ $(t = 1, \ldots, T)$ and r be the rank of $\widehat{A}_T$. We have*

$$\sum_{t=1}^{T} \boldsymbol{g}_t^\top \widehat{A}_t^\dagger \ \boldsymbol{g}_t \le r + \frac{(1+r)r}{2} \ln\left(1 + \frac{2\sum_{t=1}^T \|\boldsymbol{g}_t\|_2^2}{(1+r)r\lambda^*}\right) \ .$$

Proof. First by Cesa-Bianchi et al. [24, Lemma D.1], we have

$$\boldsymbol{g}_t^\top \widehat{A}_t^\dagger \, \boldsymbol{g}_t = \begin{cases} 1 & \text{if } \boldsymbol{g}_t \notin \text{range}(\widehat{A}_{t-1}) \\ 1 - \frac{\det_+(\widehat{A}_{t-1})}{\det_+(\widehat{A}_t)} < 1 & \text{if } \boldsymbol{g}_t \in \text{range}(\widehat{A}_{t-1}) \end{cases}$$

where $\det_+(M)$ denotes the product of the nonzero eigenvalues of matrix M. We thus separate the steps t such that $\boldsymbol{g}_t \in \text{range}(\widehat{A}_{t-1})$ from those where $\boldsymbol{g}_t \notin \text{range}(\widehat{A}_{t-1})$. For each $k = 1, \ldots, r$ let T_k be the first time step t in which the rank of A_t is k (so that $T_1 = 1$). Also let $T_{r+1} = T + 1$ for convenience. With this notation, we have

$$\begin{aligned} \sum_{t=1}^{T} \boldsymbol{g}_t^\top \widehat{A}_t^\dagger \, \boldsymbol{g}_t &= \sum_{k=1}^{r} \left(\boldsymbol{g}_{T_k}^\top \widehat{A}_{T_k}^\dagger \boldsymbol{g}_{T_k} + \sum_{t=T_k+1}^{T_{k+1}-1} \boldsymbol{g}_t^\top \widehat{A}_t^\dagger \, \boldsymbol{g}_t \right) \\ &= \sum_{k=1}^{r} \left(1 + \sum_{t=T_k+1}^{T_{k+1}-1} \left(1 - \frac{\det_+(\widehat{A}_{t-1})}{\det_+(\widehat{A}_t)} \right) \right) \\ &= r + \sum_{k=1}^{r} \sum_{t=T_k+1}^{T_{k+1}-1} \left(1 - \frac{\det_+(\widehat{A}_{t-1})}{\det_+(\widehat{A}_t)} \right) \\ &\le r + \sum_{k=1}^{r} \sum_{t=T_k+1}^{T_{k+1}-1} \ln \frac{\det_+(\widehat{A}_t)}{\det_+(\widehat{A}_{t-1})} \\ &= r + \sum_{k=1}^{r} \ln \frac{\det_+(\widehat{A}_{T_{k+1}-1})}{\det_+(\widehat{A}_{T_k})} \, . \end{aligned}$$

Fix any k and let $\lambda_{k,1}, \ldots, \lambda_{k,k}$ be the nonzero eigenvalues of $\widehat{A}_{T_k}$ and $\lambda_{k,1} + \mu_{k,1}, \ldots, \lambda_{k,k} + \mu_{k,k}$ be the nonzero eigenvalues of $\widehat{A}_{T_{k+1}-1}$. Then

$$\ln \frac{\det_+(\widehat{A}_{T_{k+1}-1})}{\det_+(\widehat{A}_{T_k})} = \ln \prod_{i=1}^{k} \frac{\lambda_{k,i} + \mu_{k,i}}{\lambda_{k,i}} = \sum_{i=1}^{k} \ln \left(1 + \frac{\mu_{k,i}}{\lambda_{k,i}} \right) \, .$$

Hence, we arrive at

$$\sum_{t=1}^{T} \boldsymbol{g}_t^\top \widehat{A}_t^+ \boldsymbol{g}_t \le r + \sum_{k=1}^{r} \sum_{i=1}^{k} \ln \left(1 + \frac{\mu_{k,i}}{\lambda_{k,i}} \right) \, .$$

To further bound the latter quantity, we use $\lambda^* \leq \lambda_{k,i}$ and Jensen's inequality :

$$\begin{aligned}\sum_{k=1}^{r}\sum_{i=1}^{k}\ln\left(1+\frac{\mu_{k,i}}{\lambda_{k,i}}\right) &\leq \sum_{k=1}^{r}\sum_{i=1}^{k}\ln\left(1+\frac{\mu_{k,i}}{\lambda^*}\right) \\ &= \frac{(1+r)r}{2}\ln\left(1+\frac{2\sum_{k=1}^{r}\sum_{i=1}^{k}\mu_{k,i}}{(1+r)r\lambda^*}\right) .\end{aligned}$$

Finally noticing that

$$\sum_{i=1}^{k}\mu_{k,i} = \textsc{tr}(\widehat{A}_{T_{k+1}-1}) - \textsc{tr}(\widehat{A}_{T_k}) = \sum_{t=T_k+1}^{T_{k+1}-1}\textsc{tr}(\boldsymbol{g}_t\boldsymbol{g}_t^\top) = \sum_{t=T_k+1}^{T_{k+1}-1}\|\boldsymbol{g}_t\|_2^2$$

completes the proof. □

Taken together, Eq. (C.2) and Theorem 23 lead to the following regret bounds (recall the definitions of λ^* and r from Theorem 23).

Corollary 1. *If $\sigma_t = 0$ for all t and η_t is set to be $\frac{1}{CL}\sqrt{\frac{d}{t}}$, then the regret of the algorithm defined by Eq.* (C.1) *is at most*

$$\frac{CL}{2}\sqrt{\frac{T}{d}}\left(r+\frac{(1+r)r}{2}\ln\left(1+\frac{2\sum_{t=1}^{T}\|\boldsymbol{g}_t\|_2^2}{(1+r)r\lambda^*}\right)\right)+4CL\sqrt{Td}.$$

On the other hand, if $\sigma_t \geq \sigma > 0$ for all t and η_t is set to be 0, then the regret is at most

$$\frac{1}{2\sigma}\left(r+\frac{(1+r)r}{2}\ln\left(1+\frac{2\sum_{t=1}^{T}\|\boldsymbol{g}_t\|_2^2}{(1+r)r\lambda^*}\right)\right) .$$

Appendix D

Experiment Details

D.1 Description of Datasets

The datasets we use in this thesis come from the UCI repository[18], LIBSVM datasets[19], and the HCRC Map Task Corpus[20]. Table D.1 shows the number of examples, number of features and average number of nonzero features of all the datasets we use.

Experiments in Section 3.1.5 use the following datasets: *20news, a9a, activity, adult, bio, census, covtype, maptaskcoref, nomao, poker, rcv1, vehv2binary.*

Experiments in Section 3.2.5 use the following datasets: *a9a, abalone, activity, adult, bank, cal_housing, casp, census, covtype, kddcup04 (phy), letter, slice, year.*

Experiments in Section 4.6 use the following datasets: *20news, a9a, acoustic, adult, australian, breast-cancer, census, cod-rna, covtype, diabetes, gisette, heart, ijcnn1, ionoshpere, letter, magic04, mnist, mushrooms, rcv1, real-sim, splice, w1a, w8a.*

[18] `http://archive.ics.uci.edu/ml/`
[19] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`
[20] `http://groups.inf.ed.ac.uk/maptask/`

Table D.1: Attributes of datasets used in experiments

Dataset	#examples	avg. sparsity	#features
20news	18845	93.9	101631
a9a	48841	13.9	123
acoustic	78823	50.0	50
adult	48842	12.0	105
australian	690	11.2	14
breast-cancer	683	10.0	10
census	299284	32.0	401
cod-rna	271617	8.0	8
covtype	581011	11.9	54
diabetes	768	7.0	8
gisette	1000	4971.0	5000
heart	270	9.8	13
ijcnn1	91701	13.0	22
ionosphere	351	30.1	34
letter	20000	15.6	16
magic04	19020	10.0	10
mnist	11791	142.4	780
mushrooms	8124	21.0	112
rcv1	781265	75.7	43001
real-sim	72309	51.3	20958
splice	1000	60.0	60
w1a	2477	11.5	300
w8a	49749	11.7	300

D.2 Detailed Experiment Results

The detailed error rates for Section 3.2.5 and 4.6 are summarized in Table D.2 and D.3 respectively.

Table D.2: Average loss for Online Gradient Boosting with different weak learners

	VW			Regression stumps			Neural Networks		
Dataset	Baseline	Alg 9	Alg 10	Baseline	Alg 9	Alg 10	Baseline	Alg 9	Alg 10
kddcup04/phy	0.7475	0.7466	0.7470	0.9201	0.7733	0.7924	0.7441	0.7480	0.7446
cal_housing	0.0094	0.0094	0.0104	0.0151	0.0138	0.0124	0.0096	0.0096	0.0107
casp	0.0632	0.0631	0.0630	0.0741	0.0741	0.0742	0.0639	0.0632	0.0631
a9a	0.4261	0.4283	0.4249	0.5749	0.5074	0.5758	0.4256	0.4266	0.4246
abalone	3.7263	3.7482	3.7154	6.7791	3.8273	4.2270	3.7380	3.7255	3.7212
activity	0.0334	0.0337	0.0316	0.4492	0.1454	0.3141	0.0192	0.0143	0.0186
adult	0.1055	0.1057	0.1056	0.1388	0.1261	0.1250	0.1081	0.1062	0.1081
bank	0.2971	0.2968	0.2973	0.3774	0.3240	0.3257	0.2962	0.2969	0.2969
census	0.1544	0.1545	0.1553	0.2073	0.1884	0.1789	0.1531	0.1531	0.1523
covtype	0.7256	0.7270	0.7286	0.7910	0.7986	0.7911	0.6807	0.6465	0.6757
letter	0.6441	0.5698	0.6108	0.7420	0.7087	0.7168	0.6542	0.5729	0.6108
shuttle	0.1616	0.1547	0.1577	0.8551	0.3678	0.4354	0.0760	0.0694	0.0802
slice	0.0076	0.0067	0.0065	0.0559	0.0362	0.0410	0.0054	0.0022	0.0044
year	0.0116	0.0119	0.0115	0.0152	0.0140	0.0141	0.0116	0.0119	0.0122

Table D.3: Error rates for Oja-SON and AdaGrad

Dataset	Oja-SON				AdaGrad
	Without Diagonal Adaptation		With Diagonal Adaptation		
	$m = 0$	$m = 10$	$m = 0$	$m = 10$	
20news	0.121338	0.121338	0.049590	0.049590	0.068020
a9a	0.204447	0.195203	0.155953	0.155953	0.156414
acoustic	0.305824	0.260241	0.257894	0.257894	0.259493
adult	0.199763	0.199803	0.150830	0.150830	0.181582
australian	0.366667	0.366667	0.162319	0.157971	0.289855
breast-cancer	0.374817	0.374817	0.036603	0.036603	0.358712
census	0.093610	0.062038	0.051479	0.051439	0.069629
cod-rna	0.175107	0.175107	0.049710	0.049643	0.081066
covtype	0.042304	0.042312	0.050827	0.050818	0.045507
diabetes	0.433594	0.433594	0.329427	0.328125	0.391927
gisette	0.208000	0.208000	0.152000	0.152000	0.154000
heart	0.477778	0.388889	0.244444	0.244444	0.362963
ijcnn1	0.046826	0.046826	0.034536	0.034645	0.036913
ionosphere	0.188034	0.148148	0.182336	0.182336	0.190883
letter	0.306650	0.232300	0.233250	0.230450	0.237350
magic04	0.000263	0.000263	0.000158	0.000158	0.000210
mnist	0.062336	0.062336	0.040031	0.039182	0.046561
mushrooms	0.003323	0.002339	0.002462	0.002462	0.001969
rcv1	0.055976	0.052694	0.052764	0.052766	0.050938
real-sim	0.045140	0.043577	0.029498	0.029498	0.031670
splice	0.343000	0.343000	0.294000	0.229000	0.301000
w1a	0.001615	0.001615	0.004845	0.004845	0.003633
w8a	0.000101	0.000101	0.000422	0.000422	0.000221

Bibliography

[1] Jacob Abernethy and Manfred K. Warmuth. Repeated games against budgeted adversaries. In *Advances in Neural Information Processing Systems 23*, 2010.

[2] Jacob Abernethy, Peter L. Bartlett, Alexander Rakhlin, and Ambuj Tewari. Optimal strategies and minimax lower bounds for online convex games. In *Proceedings of the 21st Annual Conference on Learning Theory*, 2008.

[3] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[4] Dmitry Adamskiy, Wouter M Koolen, Alexey Chernov, and Vladimir Vovk. A closer look at adaptive regret. In *Algorithmic Learning Theory*, pages 290–304, 2012.

[5] Jean-Yves Audibert and Sébastien Bubeck. Regret bounds and minimax policies under partial monitoring. *The Journal of Machine Learning Research*, 11:2785–2836, 2010.

[6] Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi. Regret in online combinatorial optimization. *Mathematics of Operations Research*, 39(1):31–45, 2014.

[7] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32 (1):48–77, 2002.

[8] Akshay Balsubramani, Sanjoy Dasgupta, and Yoav Freund. The fast convergence of incremental pca. In *Advances in Neural Information Processing Systems 26*, 2013.

[9] Boaz Barak, Moritz Hardt, and Satyen Kale. The uniform hardcore lemma via approximate bregman projections. In *The twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1193–1200, 2009.

[10] Peter L. Bartlett and Mikhail Traskin. AdaBoost is consistent. *Journal of Machine Learning Research*, 8:2347–2368, 2007.

[11] Peter L. Bartlett, Varsha Dani, Thomas Hayes, Sham Kakade, Alexander Rakhlin, and Ambuj Tewari. High-probability regret bounds for bandit online linear optimization. In *Proceedings of the 21st Annual Conference on Learning Theory*, pages 335–342, 2008.

[12] Shai Ben-David, Dávid Pál, and Shai Shalev-Shwartz. Agnostic online learning. In *Proceedings of the 22st Annual Conference on Learning Theory*, 2009.

[13] Alina Beygelzimer, Elad Hazan, Satyen Kale, and Haipeng Luo. Online gradient boosting. In *Advances in Neural Information Processing Systems 28*, 2015.

[14] Alina Beygelzimer, Satyen Kale, and Haipeng Luo. Optimal and adaptive algorithms for online boosting. In *Proceedings of the 32st International Conference on Machine Learning*, 2015.

[15] Avrim Blum. Empirical support for Winnow and Weighted-Majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 26(1): 5–23, 1997.

[16] Avrim Blum and Yishay Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8:1307–1324, 2007.

[17] Avrim Blum, Adam Kalai, and John Langford. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 203–208, 1999.

[18] Olivier Bousquet and Manfred K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363–396, 2003.

[19] Joseph K. Bradley and Robert E. Schapire. FilterBoost: Regression and classification on large datasets. In *Advances in Neural Information Processing Systems 20*, 2008.

[20] Nader H Bshouty and Dmitry Gavinsky. On boosting with polynomially bounded distributions. *The Journal of Machine Learning Research*, 3:483–506, 2003.

[21] Richard H Byrd, SL Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *arXiv preprint arXiv:1401.7020*, 2014.

[22] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[23] Nicolò Cesa-Bianchi, Yoav Freund, David Haussler, David P. Helmbold, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, May 1997.

[24] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. A second-order perceptron algorithm. *SIAM Journal on Computing*, 34(3):640–668, 2005.

[25] Nicolò Cesa-Bianchi, Pierre Gaillard, Gábor Lugosi, and Gilles Stoltz. Mirror descent meets fixed share (and feels no regret). In *Advances in Neural Information Processing Systems 25*, 2012.

[26] Kamalika Chaudhuri, Yoav Freund, and Daniel Hsu. A parameter-free hedging algorithm. In *Advances in Neural Information Processing Systems 22*, 2009.

[27] Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. An Online Boosting Algorithm with Theoretical Justifications. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[28] Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. Boosting with Online Binary Learners for the Multiclass Bandit Problem. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[29] Alexey Chernov and Vladimir Vovk. Prediction with advice of unknown number of experts. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, 2010.

[30] Michael Collins, Robert E. Schapire, and Yoram Singer. Logistic regression, AdaBoost and Bregman distances. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, 2000.

[31] Thomas M. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, January 1991.

[32] Steven de Rooij, Tim van Erven, Peter D. Grünwald, and Wouter M. Koolen. Follow the leader if you can, hedge if you must. *Journal of Machine Learning Research*, 15:1281–1316, 2014.

[33] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[34] Nigel Duffy and David Helmbold. Boosting methods for regression. *Machine Learning*, 47(2/3):153–200, 2002.

[35] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[36] Yoav Freund. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 391–398, July 1992.

[37] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[38] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[39] Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[40] Yoav Freund, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. Using and combining predictors that specialize. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 334–343, 1997.

[41] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), October 2001.

[42] Pierre Gaillard, Gilles Stoltz, and Tim Van Erven. A second-order bound with excess losses. In *Proceedings of the 27th Annual Conference on Learning Theory*, 2014.

[43] Dan Garber and Elad Hazan. A linearly convergent conditional gradient algorithm with applications to online and stochastic optimization. *arXiv preprint arXiv:1301.4666*, 2013.

[44] Dan Garber, Elad Hazan, and Tengyu Ma. Online learning of eigenvectors. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 560–568, 2015.

[45] Claudio Gentile and Francesco Orabona. On multilabel classification and ranking with partial feedback. In *Advances in Neural Information Processing Systems 25*, pages 1151–1159. Curran Associates, Inc., 2012.

[46] Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *arXiv preprint arXiv:1501.01711*, 2015.

[47] Alon Gonen and Shai Shalev-Shwartz. Faster sgd using sketched conditioning. *arXiv preprint arXiv:1506.02649*, 2015.

[48] Helmut Grabner and Horst Bischof. On-line boosting and vision. In *CVPR*, volume 1, pages 260–267, 2006.

[49] Helmut Grabner, Christian Leistner, and Horst Bischof. Semi-supervised on-line boosting for robust tracking. In *ECCV*, pages 234–247, 2008.

[50] Moritz Hardt and Eric Price. The noisy power method: A meta algorithm with applications. In *Advances in Neural Information Processing Systems 27*, pages 2861–2869, 2014.

[51] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Verlag, 2001.

[52] Trevor J Hastie and Robert J Tibshirani. *Generalized additive models*, volume 43. CRC Press, 1990.

[53] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2016.

[54] Elad Hazan and Satyen Kale. Projection-free online learning. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[55] Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization. *JMLR*, 15(1): 2489–2512, 2014.

[56] Elad Hazan and C. Seshadhri. Adaptive algorithms for online decision problems. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 14, 2007.

[57] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[58] David P. Helmbold and Robert E. Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(1):51–68, April 1997.

[59] Mark Herbster and Manfred Warmuth. Tracking the best expert. *Machine Learning*, 32(2):151–178, 1998.

[60] Mark Herbster and Manfred K Warmuth. Tracking the best linear predictor. *The Journal of Machine Learning Research*, 1:281–309, 2001.

[61] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[62] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning*, pages 427–435, 2013.

[63] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[64] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):4, 2014.

[65] Robert Kleinberg. Anytime algorithms for multi-armed bandit problems. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 928–936. ACM, 2006.

[66] Robert David Kleinberg. *Online decision problems with large strategy sets*. PhD thesis, MIT, 2005.

[67] Wouter M Koolen, Dmitry Adamskiy, and Manfred K Warmuth. Putting bayes to sleep. In *Advances in Neural Information Processing Systems 25*, pages 135–143, 2012.

[68] Chun-Liang Li, Hsuan-Tien Lin, and Chi-Jen Lu. Rivalry of two families of algorithms for memory-restricted streaming pca. *arXiv preprint arXiv:1506.01490*, 2015.

[69] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[70] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[71] Xiaoming Liu and Ting Yu. Gradient feature selection for online boosting. In *ICCV*, pages 1–8, 2007.

[72] Haipeng Luo and Robert E. Schapire. Towards Minimax Online Learning with Unknown Time Horizon. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[73] Haipeng Luo and Robert E. Schapire. A Drifting-Games Analysis for Online Learning and Applications to Boosting. In *Advances in Neural Information Processing Systems 27*, 2014.

[74] Haipeng Luo and Robert E. Schapire. Achieving All with No Parameters: AdaNormalHedge. In *Proceedings of the 28th Annual Conference on Learning Theory*, 2015.

[75] Haipeng Luo, Alekh Agarwal, Nicolò Cesa-Bianchi, and John Langford. Efficient second order online learning via sketching. *arXiv preprint arXiv:1602.02202*, 2016.

[76] Stéphane G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, December 1993.

[77] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems 12*, 2000.

[78] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers*. MIT Press, 2000.

[79] H Brendan McMahan and Francesco Orabona. Unconstrained online linear learning in hilbert spaces: Minimax algorithms and normal approximations. In *Proceedings of the 27th Annual Conference on Learning Theory*, 2014.

[80] H. Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. In *Proceedings of the 23rd Annual Conference on Learning Theory*, 2010.

[81] Aryan Mokhtari and Alejandro Ribeiro. Global convergence of online limited memory bfgs. *arXiv preprint arXiv:1409.2045*, 2014.

[82] Philipp Moritz, Robert Nishihara, and Michael I Jordan. A linearly-convergent stochastic l-bfgs algorithm. *arXiv preprint arXiv:1508.02087*, 2015.

[83] Indraneel Mukherjee and Robert E. Schapire. Learning with continuous experts using drifting games. *Theoretical Computer Science*, 411(29):2670–2683, 2010.

[84] Hariharan Narayanan and Alexander Rakhlin. Random walk approach to regret minimization. In *Advances in Neural Information Processing Systems 23*, 2010.

[85] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[86] Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106(1):69–84, 1985.

[87] Francesco Orabona. Simultaneous model selection and optimization through parameter-free stochastic learning. In *Advances in Neural Information Processing Systems 27*, 2014.

[88] Francesco Orabona and Dávid Pál. Scale-free algorithms for online linear optimization. In *The 26th International Conference on Algorithmic Learning Theory (ALT)*, 2015.

[89] Francesco Orabona, Koby Crammer, and Nicolo Cesa-Bianchi. A generalized online mirror descent with applications to classification and regression. *Machine Learning*, 99(3):411–435, 2015.

[90] Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *Eighth International Workshop on Artificial Intelligence and Statistics*, pages 105–112, 2001.

[91] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Relax and localize: From value to algorithms. In *Advances in Neural Information Processing Systems 25*, 2012. Full version available in arXiv:1204.0870.

[92] Lev Reyzin and Robert E. Schapire. How boosting the margin can also boost classifier complexity. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[93] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin American Mathematical Society*, 55:527–535, 1952.

[94] Stéphane Ross, Paul Mineiro, and John Langford. Normalized online learning. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligenc (UAI)*, 2013.

[95] Amir Sani, Gergely Neu, and Alessandro Lazaric. Exploiting easy data in online optimization. In *Advances in Neural Information Processing Systems 27*, 2014.

[96] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2): 197–227, 1990.

[97] Robert E. Schapire. Drifting games. *Machine Learning*, 43(3):265–291, June 2001.

[98] Robert E. Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. MIT Press, 2012.

[99] Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-newton method for online convex optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 436–443, 2007.

[100] Rocco A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.

[101] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011.

[102] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[103] Matus Telgarsky. Boosting with the logistic loss is consistent. In *Proceedings of the 26th Annual Conference on Learning Theory*, 2013.

[104] Tim Van Erven, Wojciech Kotlowski, and Manfred K Warmuth. Follow the leader with dropout perturbations. In *Proceedings of the 27th Annual Conference on Learning Theory*, 2014.

[105] Vowpal Wabbit. `https://github.com/JohnLangford/vowpal_wabbit/`.

[106] Manfred K. Warmuth and Wouter M. Koolen. Open problem: Shifting experts on easy data. In *Proceedings of the 27th Annual Conference on Learning Theory*, 2014.

[107] Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. Context tree weighting: A sequential universal source coding procedure for FSMX sources. In *Proceedings 1993 IEEE International Symposium on Information Theory*, page 59, 1993.

[108] Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.

[109] Max A Woodbury. Inverting modified matrices. *Memorandum report*, 42:106, 1950.

[110] David P Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Machine Learning*, 10(1-2):1–157, 2014. ISSN 1551-305X. doi: 10.1561/0400000060.

[111] Tong Zhang and Bin Yu. Boosting with early stopping: Convergence and consistency. *Annals of Statistics*, 33(4):1538–1579, 2005.

[112] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.