Algorithmic Improvements to Boosting and Neural Networks

Alexander Mathiasen

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Abstract

This thesis is based on three publications [1, 2, 3].

The first publication [1] concerns a type of machine learning called "boosting." Informally, boosting algorithms construct an accurate "boosted" classifier by combining several less accurate classifiers. Experimentally, the generalization error of boosted classifiers improves during training, even when training *after* achieving zero training error. *Margin theory* [8] attributes this desirable generalization phenomenon to large "margins." This led to a line of research with a common goal [10, 12, 24, 43, 44]: maximize the smallest margin of a boosted classifier using as few "less accurate classifiers" as possible. This line of research culminated with AdaBoostV [44], which was later conjectured to yield an optimal trade-off between the number of "less accurate classifiers" and the smallest margin [40]. Our main contribution is an algorithm, SparsiBoost, which provides a better trade-off and thus refutes the conjecture. Finally, we present a lower bound which implies that SparsiBoost is optimal.

The second publication [2] concerns generalization error upper bounds from *margin theory* [8]. Despite numerous margin-based generalization *upper bounds* [8, 12, 21, 50], nothing is known about the tightness of these bounds. To this end, we present the first margin-based generalization *lower bound*. Our lower bound matches Breiman's generalization upper bound [12], which depends on the *smallest* margin, ruling out any further improvements. Furthermore, our lower bound almost matches the k'th margin bound [21], the strongest known margin-based generalization upper bound, which depends on *all* margins, ruling out any improvements larger than a multiplicative log factor.

The third publication [3] concerns neural networks. Training neural networks sometimes entail computing time-consuming operations on their weight matrices, e.g., matrix determinants [31]. Operations like matrix determinants are often faster to compute given the Singular Value Decomposition (SVD). Previous work implicitly maintains the SVD of the weight matrices in a Neural Network without explicitly computing the SVD [52]. In theory, their technique allows faster determinant computation, however, in practice, we find no speed up. We present an algorithm, FastH, which is faster in practice (around 5 times faster than [52]). FastH has the same time complexity as [52], however, for a $d \times d$ weight matrix and a batch size b, FastH reduces the number of sequential stages from $O(d)$ to $O(d/b+b)$. During the writing of this thesis, we improved FastH to use $O(d/b+\log(b))$ sequential stages.

Abstract (Danish)

Denne afhandling er baseret på tre publikationer [1, 2, 3].

Den første publikation [1] omhandler en type maskinlæring kaldet "boosting." Beskrevet uformelt, konstruerer boosting algoritmer en nøjagtig "boostet" klassifikator ved at kombinere flere mindre nøjagtige klassifikatorer. Eksperimentelt har man fundet at generaliserings fejlen af boostet klassifikatorer falder med træning, også selvom klassifikatoren har nul træningsfejl. *Margen teori* [8] tilskriver dette ønskværdige generaliserings fænomen til store "margener." Dette ledte til adskillige forskningsartikler med et fælles mål [10, 12, 24, 43, 44]: maksimer den mindste margen af en boostet klassifikator ved brug af så få "mindre nøjagtige klassifikatorer" som muligt. Tidligere forskning kulminerede med algoritmen AdaBoostV [44], hvilken [40] formodede garanterer en optimal afvejning mellem antallet af "mindre nøjagtige klassifikatorer" og den mindste margen. Vores hovedbidrag er en ny algoritme, SparsiBoost, der giver en bedre afvejning og derfor afviser [40]'s formodning. Endeligt præsenterer vi en nedre grænse der medfører at SparsiBoost garanterer en optimal afvejning.

Den anden publikation [2] omhandler grænser for generaliserings fejl fra *margen teori* [8]. På trods af talrige øvre grænser for generaliserings fejl der baserer sig på margener [8, 12, 21, 50], er intet kendt for tilsvarende nedre grænser. Vi præsenterer den første nedre grænse for generaliserings fejl baseret på margener. Breiman's [12] øvre grænse for generaliserings fejl afhænger af den *mindste margin*, for hvilken vores nedre grænse udelukker enhver forbedring. Den k'te margen [21] øvre grænse er den stærkest kendte øvre grænse og afhænger af *alle margener*, for hvilken vores nedre grænse udelukker enhver forbedring større end en multiplikativ log faktor.

Den tredje publikation [3] omhandler neurale netværk. Træning af neurale netværk indebærer nogle gange tidskrævende operation på deres vægt matricer, e.g., matrix determinanter [31]. Operationer som matrix determinanter kan ofte udregnes hurtigere givet matricens Singulær Værdi Dekomposition (SVD). Tidligere forskning [52] vedligeholder SVD'en af vægt matricer i neurale netværk implicit, uden nogen eksplicit SVD udregninger. I teorien tillader deres teknik hurtigere determinant beregninger, men i praksis fandt vi ingen forbedring. Vi præsenterer en algoritme, FastH, der er hurtigere i praksis (omkring 5 gange hurtigere end [52]). FastH har samme tidskompleksitet som [52], men, for en $d \times d$ matrix og en batch størrelse b, reducerer FastH antallet af sekventielle skridt fra $O(d)$ til $O(d/b+b)$. Imens denne afhandling blev skrevet, forbedrede vi FastH til at bruge $O(d/b+\log(b))$ sekventielle skridt.

To my mom and dad. Thank you for your unconditional love and your infinite patience to answer my endless stream of questions. One lifetime is too short to express my gratitude.

Alexander Mathiasen,
Aarhus, November 28, 2021.

Acknowledgments

During my time as a PhD student, I have had the good fortune to work with countless exceptional people. I want to express my gratitude to every single person I met during this journey. Friends, collaborators, peers, students, teachers, and family. Thank you. A special thanks go to my supervisor, Kasper Green Larsen. Kasper has given me the ultimate freedom to pursue my own ideas. At the same time, Kasper has supported me while I chased my occasionally crazy ideas, always ready to provide excellent feedback whenever I needed it. For this, I am hugely indebted to Kasper. Finally, I'd like to thank the entire research community at Aarhus University. From the data-intensive systems group to the cardiovascular researchers at the department of biomedicine. Thanks for your time, collaboration and open-mindedness.

Alexander Mathiasen,
Aarhus, November 28, 2021.

Preface

This thesis is based on three publications [1, 2, 3] and one manuscript [4]. The work is grouped into two chapters by topic.

- **Chapter 2: Boosting**
  - [1] *Optimal Minimal Margin Maximization with Boosting.* **ICML 2019**.
  - [2] *Margin-Based Generalization Lower Bounds for Boosted Classifiers.* **NeurIPS 2019**.
- **Chapter 3: Neural Networks**
  - [3] *What if Neural Networks had SVDs?* **NeurIPS 2020**. Spotlight presentation, 200 out of 9500.
  - [4] *One Reflection Suffice* (Manuscript).

Purpose. In Denmark, the purpose of a PhD thesis is defined by law [47] with further regulations from my faculty at Aarhus University [5]. In short, a PhD thesis *"presents the results of the PhD project and documents the PhD student's ability to communicate theoretical and experimental skills"* [5]. To this end, the student may write a PhD thesis as a monograph or include papers and manuscripts. I chose to include papers and manuscripts which make up Chapter 4. When including articles and manuscripts, the student must write introductions. These introductions make up Chapters 2 and 3.

Contents

Chapter 1

Introduction

This thesis concerns *algorithmic improvements* to *machine learning*. Below we (1) introduce *machine learning*, (2) explain what *algorithmic improvements* are, and (3) explain why *algorithmic improvements* are desirable in *machine learning*.

(1) Machine Learning

Consider the images in Figure 1.1. Your brain will classify one image as a cat and the other as a dog. Can we make computers classify such images? Computers represent images as tables with many numbers. For a computer to classify an image, we need to write a program that takes a "number table" as input and outputs whether the "number table" represents a cat or a dog. No human has successfully written such a "classifier program" for images. One may think of machine learning as a data-driven approach for writing such classifier programs. Instead of writing the classifier program ourselves, we write a machine learning program that constructs the classifier program for us by utilizing a lot of "training" images.

(2) Algorithmic Improvement

An algorithm is a procedure to solve a mathematical problem. For example, $172 \cdot 812$ is a mathematical problem. The procedure you learned to solve multiplication problems in primary school is an algorithm. Computer scientists want algorithms that scale well. That is, algorithms for which "the amount of work" needed to solve larger and larger problem instances increases as little as possible. The amount of work it takes to multiply two n digit numbers with the primary school multiplication algorithm is "roughly" n^2. Can we do better than n^2? Karatsuba [28] proved that we can do better. In particular, he introduced an algorithm for which the amount of work increases by roughly $n^{1.59}$. Notably, Karatsuba's algorithm computes *exactly* the same as the primary school multiplication algorithm. If both algorithms are given the problem $3 \cdot 5$, then both algorithms finds the solution 15. Yet, for very large numbers, Karatsuba's algorithm requires less work. This is an algorithmic improvement.

(3) Desirable?

Machine learning has improved drastically during the past decade. For example, on a popular image classification benchmark, the accuracy of the best machine learning classifier improved from 50 to 91 percent [14]. In many cases, such improvements come at the cost of larger machine learning systems, which require much more expensive computers. It is not unheard of that research groups spend millions of dollars training a single machine learning system [13]. From this perspective, it is desirable to have algorithms that perform as little work as possible. The following two pages clarify how the publications within this thesis [1, 2, 3] relates to algorithmic improvements through a short technical overview. The subsequent Chapters 2 and 3 contain further details and a less technical introduction.

Figure 1.1: Image of a cat and dog from [29].

Publication [1]. Chapter 2 concerns a type of machine learning called "boosting." Boosting algorithms use training data $\langle(x_1,y_1),\dots,(x_n,y_n)\rangle$ to produce a "boosted classifier" $f(x)=\text{sign}(\sum_{j=1}^T \alpha_j h_j(x))$ where $x_i\in\mathbb{R}^d$, $y_i\in\{\pm1\}$, $\alpha_j\in\mathbb{R}$ and $h_j:\mathbb{R}^d\to[\pm1]$ is a "less accurate classifier" from a set of classifiers $h_j\in H$. Experimentally, the generalization error of a boosted classifier $f(x)$ improves during training, even when training *after* achieving zero training error. *Margin theory* [8] attributes this desirable generalization phenomenon to large "margins." The margin of a training example (x_i,y_i) is:

$$\text{margin}(x_i):=\frac{y_i\sum_{j=1}^T \alpha_j h_j(x_i)}{\sum_{j=1}^T|\alpha_j|}\in[-1,1]. \tag{1.1}$$

A positive margin indicates that the classifier is correct and larger values indicate higher confidence. Margin theory led to a long line of research with a common goal [10, 12, 24, 43, 44]: find a boosted classifier $f(x)$ with a large *minimal margin*[1] using few "less accurate classifiers." This line of research culminated with the AdaBoostV algorithm [44], which satisfies the following approximation guarantee. Let p^* be the best possible minimal margin *any* linear combination of classifiers from H can achieve on the training data $\langle(x_1,y_1),\dots,(x_n,y_n)\rangle$. AdaBoostV guarantees that it finds a classifier $f_T(x)$ with at most T "less accurate classifiers," such that $f_T(x)$ has minimal margin $p_T\geq p^*-v$ where:

$$v=O\left(\sqrt{\frac{\log(n)}{T}}\right). \tag{1.2}$$

It was later conjectured by [40] that AdaBoostV is optimal, that is, no algorithm can promise a better guarantee than $v=\Omega(\sqrt{\log(n)/T})$. Our main contribution is a new algorithm, SparsiBoost, which guarantees a better $v=O(\sqrt{\log(n/T)/T})$ and thus refutes the conjecture. Finally, we prove that SparsiBoost is optimal. From an algorithmic perspective, SparsiBoost presents an *algorithmic improvement* to the previous machine learning approximation algorithm AdaBoostV.

Publication [2]. Sections 2.4 and 2.5 concerns *margin theory* [8]. Margin theory is the motivation for why [12, 43, 44] want a boosted classifier with a large minimal margin. The goal of a machine learning classifier is to perform well on "unseen" examples. Formally, one assumes that the training data $S=\langle(x_1,y_1),\dots,(x_n,y_n)\rangle$ contains independently and identically distributed samples $(x_i,y_i)\sim D$ for an unknown distribution D. The error on "unseen" examples is then the generalization error $\Pr_{(x,y)\sim D}[f(x)y<0]$. It is difficult to minimize the generalization error because D is unknown. Margin theory address this difficulty by relating generalization error to margins. Loosely speaking, for a finite set of classifiers H, Breiman [12] proved that for any "boosting classifiers" $f(x)$, the following holds with high probability over sampling $S\sim D^n$:

$$\underbrace{\Pr_{(x,y)\sim D}[f(x)y<0]}_{\text{Generalization Error}}\leq\underbrace{O\left(\frac{\ln(|H|)\ln(n)}{\widehat{\theta}^2 n}\right)}_{\text{Generalization Upper Bound}} \quad\text{where}\quad \widehat{\theta}:=\min_{i=1,\dots,n}\text{margin}(x_i). \tag{1.3}$$

This allows us to minimize an upper bound on generalization error, by choosing a boosting classifier $f(x)$ with a large minimal margin on the training data. This is exactly what the minimal margin boosting algorithms does [10, 12, 24, 43, 44], including our own SparsiBoost [1].

From this perspective, it is desirable that Breiman's upper bound is tight, that is, one cannot prove a stronger minimal-margin based upper bound. If one could prove a stronger generalization bound, SparsiBoost might minimize a sub-optimal upper bound on generalization error. Our main result from [2] implies that Breiman's generalization bound is tight. From an algorithmic perspective, our result thus rules out any potential *algorithmic improvements* that rely on a stronger minimal margin generalization upper bound. Finally, our lower bound also applies to a more general case than the minimal margin. In particular, we can rule out any improvements larger than a multiplicative $\log(n)$ factor to the k'th margin generalization bound [21], which is currently the strongest margin-based generalization bound.

[1]The minimal margin is $\widehat{\theta}:=\min_{i=1}^n\text{margin}(x_i)$.

Publication [3]. Chapter 3 concerns faster training of neural networks. Neural networks sometimes use time-consuming operations on their weight matrices. For example, previous work [31] uses a loss function that depends on determinants. For a $d \times d$ weight matrix W it takes $O(d^3)$ time to compute $\det(W)$ naively. Operations like $\det(W)$ are often faster to compute given the Singular Value Decomposition (SVD) $W = U\Sigma V^T$ for diagonal Σ and orthogonal $U^TU = V^TV = I$.

$$|\det(W)| = |\det(U\Sigma V^T)| = |\det(U)\cdot\det(\Sigma)\cdot\det(V^T)| = |\det(\Sigma)| = \prod_{i=1}^{d}|\Sigma_{ii}|. \tag{1.4}$$

We can then compute $\det(W)$ in $O(d)$ time, however, computing the SVD takes $O(d^3)$ time. Previous work mitigates the $O(d^3)$ time SVD computation by representing W implicitly in its SVD [52]. In theory, their technique allows faster determinant computations, however, in practice, we find no improvements. The poor performance in practice occurs because the previous technique is sequential and thus ill-suited for GPUs. In particular, their method has $O(d)$ sequential stages because the orthogonal matrices U and V are represented by d Householder matrices. For $x, v_i \in \mathbb{R}^d$ they compute:

$$x \cdot U = \overbrace{x \cdot H(v_1)}^{\text{Stage 1}} \overbrace{\cdot H(v_2)}^{\text{Stage 2}} \overbrace{\cdot H(v_3)}^{\text{Stage 3}} \cdots \overbrace{\cdot H(v_d)}^{\text{Stage d}} \quad \text{where} \quad H(v) := I - 2\frac{vv^T}{||v||_2^2}. \tag{1.5}$$

We mitigate this issue with a new algorithm, FastH, which, for a batch size b reduces the number of sequential stages from $O(d)$ to $O(d/b + b)$. In practice FastH is faster than the sequential algorithm, fast enough to speed up determinant computations, see Figure 1.2.

From an algorithmic perspective, FastH presents an *algorithmic improvement* for determinant computations in Neural Networks. That said, the use of SVDs with FastH goes beyond matrix determinants, in particular, one can achieve *algorithmic improvements* for several matrix operations used during the training of Neural Networks, we present four such examples in Section 3.1.1. Finally, during the writing of this thesis, we improved FastH by reducing the number of sequential stages from $O(d/b + b)$ to $O(d/b + \log(b))$. We present this improvement in Section 3.4.1.

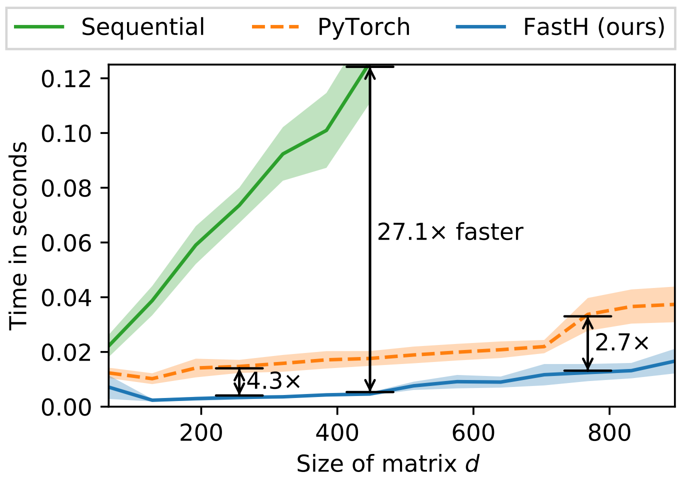


Figure 1.2: Time on a GPU when running (1) the naive sequential algorithm from [52], (2) time of $O(d^3)$ determinant computation with PyTorch [41], and (3) our FastH algorithm. FastH is much faster than the sequential algorithm, it is even fast enough to speed up determinant computations for neural networks.

Chapter 2

Boosting

Much of machine learning boils down to classifying data. As a result, there are many different types of machine learning classifiers, e.g., support vector machines [17], decision trees [20], and neural networks [22]. Instead of creating yet another type of machine learning classifier, one may ask if it is possible to combine classifiers like decision trees "boosting" their individual accuracy. This is possible with boosting algorithms, which construct a single joint "accurate classifier" by combining several "less accurate classifiers."

In machine learning, one gathers independent and identically distributed examples which are split into training and validation data. The training data is used to train a classifier which is "tested" using the validation data. During training, validation error may worsen even though training error is improving, i.e., the classifier is "overfitting" the training data. AdaBoost [19], perhaps the most famous boosting algorithm, has been found to experience a surprising phenomenon related to overfitting. As demonstrated by [8], validation error can continue to improve even *after* reaching 0% training error, see Figure 2.1.

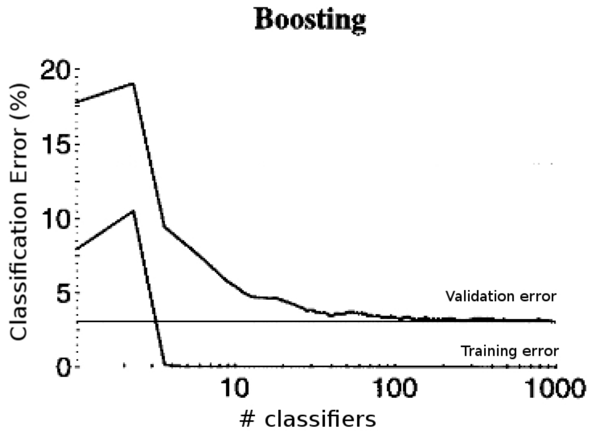


Figure 2.1: Figure from [8]. AdaBoost constructs a joint classifier in iterations: at the t'th iteration, a new classifier is added which rectifies the mistakes of the previous $t-1$ classifiers. Just 5 classifiers achieve 0% training error while subsequent training improves validation error.

This *generalization phenomenon* is desirable, however, it has proven difficult to explain. Arguably, the most prominent explanation is *margin theory* [8]. Many algorithms were developed to utilize the generalization phenomena as described by margin theory [10, 12, 24, 43, 44]. Prior to our work, the state-of-the-art was AdaBoostV [44]. As alluded to in the introduction, AdaBoostV satisfies a "margin-based" approximation guarantee. Our algorithm, SparsiBoost [1], has a provably better guarantee. Furthermore, we manage to prove a lower bound that matches SparsiBoost. In a certain sense, this means that one cannot hope for a better guarantee than that provided by SparsiBoost. We outline the development from AdaBoost and AdaBoostV to SparsiBoost in Section 2.1.

2.1 From Prior Boosting Algorithms [19, 44] to Our Results [1]

For the sake of simplicity, we consider AdaBoost for binary classification, that is, we are given n training examples $x_i \in X$ with labels $y_i \in \{\pm 1\}$. The goal of AdaBoost is to combine individual classifiers $h_t : X \rightarrow \{\pm 1\}$ into a "more accurate" classifier $C(x)$.

$$C(x) = \text{sign}(f(x)) \quad \text{where} \quad f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \quad \text{for} \quad \alpha_t \in \mathbb{R}. \tag{2.1}$$

All hypotheses $h_1, \ldots, h_T$ come from a hypothesis set H, e.g., decision trees or neural networks. AdaBoost produces the joint classifier C in iterations: at the t'th iteration, AdaBoost trains h_t on a reweighed version of the training data, and assigns h_t a carefully chosen weight α_t. Informally, [18] proved that the error of C decreases exponentially fast in T if each h_t is "slightly better than random guessing." In particular, if we let ε_t be the error of h_t on the reweighed training data, then C has error no larger than $\exp(-2\sum_{t=1}^{T}(1/2 - \varepsilon_t)^2)$ on the original training data. As a result, if all $\varepsilon_t \leq 1/2 - c$ for a fixed $c > 0$, then f has error less than $1/n$ after $O(\log n)$ iterations, i.e., it perfectly classifies the training data after $T = O(\log n)$ iterations.

This may raise concerns for overfitting. Surprisingly, continued training after reaching zero training error can improve validation error. For example, in Figure 2.1 we see that AdaBoost reaches 0% training error after only 5 iterations, however, further training continues to decrease validation error. The most prominent attempt at explaining this phenomenon was pioneered by [8] which relies on margins.

$$\text{margin}(x) := yF(x) \quad \text{where} \quad F(x) = \frac{f(x)}{\sum_{t=1}^{T} |\alpha_t|} \tag{2.2}$$

Note that F is just a scaled version of f that ensures $F(x) \in [-1, +1]$ and thus margin$(x) \in [-1, 1]$. It may be useful to think of margin(x) as a number that indicates how much F struggles with a training example x. If margin(x) is positive $C(x)$ is correct, and larger margin(x) indicates that $F(x)$ is more "confident." Experimentally, [8] found that continued training lead to better (larger) margins on the training data. For example, consider the classifier in Figure 2.1 when it had $T = 5$ and $T = 100$ hypotheses. The $T = 100$ hypotheses have better validation error but also much better (larger) margins, see Figure 2.2.

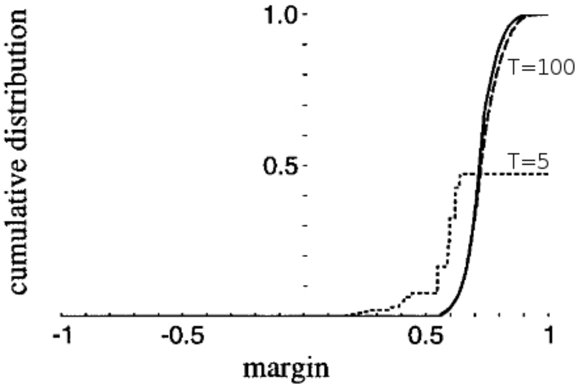


Figure 2.2: Figure from [8]. Each line depicts a classifier, we only consider the two dashed lines ($T = 5$ and $T = 100$). The plot shows the cumulative distribution of margin, e.g., for $T = 5$ it shows that roughly 10% of training data has a margin ≤ 0.5.

Inspired by their experimental findings, [8] subsequently proved a margin-based generalization bound, which, informally, states that larger margins lead to a smaller error on new unseen samples (assuming the unseen samples and the training samples are independently and identically distributed). One refers to the error on new unseen samples as "out-of-sample error" since it is the error "outside" the training samples. Further research led to several improved generalization bounds [8, 12, 21]. Of these, perhaps the simplest is due to Breimann [12], which we now explain.

A voting classifier is a linear combination $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$ if all α_t are non-negative and $\sum_t \alpha_t = 1$. For simplicity we consider hypothesis sets H where $h \in H$ implies $-h \in H$, this allows us to assume $\alpha_t \geq 0$ by exchanging h and $-h$. Furthermore, we can scale $\alpha_j = \alpha_j/(\sum_t \alpha_t)$ without changing the prediction of $C(x)$. We may thus view the output of AdaBoost as a voting classifier. Notably, Breimann's generalization bound states that with high probability over sampling training data $S = \langle (x_1, y_1), \ldots (x_n, y_n) \rangle \sim D^n$ from an unknown distribution D, it holds that every voting classifier f satisfies the following inequality:

$$\underbrace{\Pr_{(x,y)\sim D}[f(x)y < 0]}_{\text{Out-of-sample Error}} \leq \underbrace{O\left(\frac{\log|H|\log n}{\theta^2 n}\right)}_{\text{Upper Bound}} \quad \text{where} \quad \underbrace{\theta := \min_{i=1}^{n} \text{margin}(x_i).}_{\text{Minimal Margin}} \tag{2.3}$$

This variant of Breiman's bound relies on $\log|H|$ and assumes H is finite. One can attain a similar bound for infinite H by using the famous Vapnik-Chervonenkis (VC) dimensions [49]. The key insight from Breiman's bound is that the upper bound decreases with larger minimal margin θ. This suggests that a larger minimal margin leads to smaller out-of-sample error. Indeed, if we carefully inspect Figure 2.2 and Figure 2.1, we find that the classifier with the largest minimal margin had the lowest validation error (validation error attempts to approximate out-of-sample error). In particular, the classifier with $T = 5$ hypotheses attains a minimal margin around 0.2 with 7% validation error, while the classifier with $T = 100$ hypotheses has a minimal margin 0.5 with 3% validation error.

One may then suspect the generalization phenomena relates to margins. However, AdaBoost does not explicitly maximize the minimal margin. This lead Breimann to introduce an algorithm Arc-GV which explicitly maximizes the minimal margin [12] which was followed by many subsequent algorithms [10, 24, 43, 44]. Some of these algorithms have provable guarantees for how good the minimal margin of the produced classifier is. Of these, the best bound is attained by AdaBoostV [44], which we now explain. Suppose the best possible minimal margin of any voting classifier on the given training data is p^*

$$p^* = \max_{a>0} \min_i y_i \cdot \frac{\sum_{h\in H} \alpha_h h(x_i)}{\sum_{h\in H} \alpha_h}. \tag{2.4}$$

AdaBoostV promises to return a classifier $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$ with minimal margin $p = p^* - v$ with an additive "error" $v = O(\sqrt{\log(n)/T})$. To be consistent with the literature, we shall say that $f(x)$ has a "gap" v. The number of hypotheses T is linearly proportional to training and prediction time. It is thus desirable to attain a better (smaller) "gap" using the same number of hypotheses. From an algorithmic point of view, AdaBoostV is very elegant. The only change relative to AdaBoost is the computation of α_t which indirectly alters the reweighing of the training data.

After AdaBoostV there was no improvements to the $v = O(\sqrt{\log(n)/T})$ bound. This lead [40] to wonder whether it is impossible to improve AdaBoostV. To this end, [40] pointed out that a seemingly unrelated lower bound [32] has the following consequences: there exists training data $(x_1, y_1), \ldots, (x_n, y_n)$ and a hypothesis set H such that for any $T \in [\log(n); \sqrt{n}]$ the gap will be at least $v = \Omega(\sqrt{\log(n)/T})$. This means that AdaBoostV is optimal whenever $T \in [\log(n); \sqrt{n}]$. Furthermore, [40] conjecture that this lower bound holds for all $T \leq c \cdot n$ for some constant $c > 0$. If true, AdaBoostV would attain the optimal trade-off between hypotheses T and gap $v = \Theta(\sqrt{\log(n)/T})$. Their conjecture was published as an open problem at the Journal of Machine Learning Research.

We refute the conjectured lower bound from [40] by introducing a new algorithm called SparsiBoost. Assuming $T \leq n/2$ SparsiBoost promises a gap $v = O(\sqrt{\log(n/T)/T})$.[1] For some T, the SparsiBoost upper bound $v = O(\sqrt{\log(n/T)/T})$ is better (smaller) than the AdaBoostV upper bound $v = O(\sqrt{\log(n)/T}))$. This is easiest to see in the extreme case where $T = n/2$. In this case, AdaBoostV promises $v = O(\sqrt{2\log(n)/n}) = O(\sqrt{\log(n)/n})$ which is asymptotically worse (larger) than SparsiBoost.

$$v = O\left(\sqrt{2\frac{\log(2\cdot n/n)}{n}}\right) = O\left(\sqrt{\frac{1}{n}}\right). \tag{2.5}$$

[1]The $T \leq n/2$ ensures that $\log_2(n/T) \geq 1 \neq 0$. See the full article for general T.

More generally, SparsiBoost promises an asymptotically better (smaller) v for any $T = n^{1-o(1)}$ since

$$v = O\left(\sqrt{\frac{\log(n/n^{1-o(1)})}{T}}\right) = O\left(\sqrt{\frac{\log(n^{o(1)})}{T}}\right) = o\left(\sqrt{\frac{\log(n)}{T}}\right). \tag{2.6}$$

If SparsiBoost is asymptotically better than AdaBoostV, and AdaBoostV matches the [40, 32] lower bound, it may seem that SparsiBoost violates the lower bound. This is not the case because the lower bound only holds for $T \in [\log(n); \sqrt{n}]$. Indeed, for the relevant parameters, the SparsiBoost upper bound becomes $v = O(\sqrt{\log(n/T)/T}) = O(\sqrt{\log(n)/T})$ which does not violate the lower bound.

Notably, SparsiBoost explains why both [32] and [40] could not extend the $v = \Omega(\sqrt{\log(n)/T})$ lower bound to hold for all $T \le c \cdot n$ for some constant $c > 0$: such a lower bound is false. This made us consider whether we could prove the "less restrictive" lower bound $v = \Omega(\sqrt{\log(n/T)/T})$ for $T \le c \cdot n$ for some constant $c > 0$. In particular, we proved that there exists training examples $(x_1, y_1), \dots, (x_n, y_n)$ and a hypothesis set H such that for all relevant values of T, the gap will be at least $\Omega(\sqrt{\log(n/T)/T})$. This implies that SparsiBoost attains the optimal trade-off between hypotheses T and gap $v = \Theta(\sqrt{\log(n/T)/T})$. The following Section 2.2 sketches the core ideas underlying SparsiBoost, emphasizing our sparsification result which may be of independent interest.

Doubts on Margin Theory. Breiman [12] raised doubts on margin theory. He found that Arc-GV experimentally got better margins than AdaBoost on 98 percent of the training data. Margin theory would thus predict Arc-GV generalizes better, however, experimentally AdaBoost attained the best validation error. It was later found by [45] that the comparison was "unfair." See the full article for these details.

2.2 Our Algorithm: SparsiBoost [1]

High-level idea. SparsiBoost uses AdaBoostV to train $c \cdot T$ hypotheses for a carefully chosen $c > 1$, even though we must output only T hypotheses. AdaBoostV promises that the resulting classifier has a gap of at most $v = O(\sqrt{\log(n)/(cT)})$ which, for the correct choice of c, is much better (smaller) than our goal $v = O(\sqrt{\log(n/T)/T})$. SparsiBoost then carefully removes the additional $cT - T$ hypotheses while attempting to preserve all margins. The resulting T hypotheses has a gap $v = O(\sqrt{\log(n/T)/T})$.

Sketch. Let $f_{cT}(x) = \sum_{t=1}^{cT} w_t h_t(x)$ be the classifier returned from AdaBoostV with weights $w \in \mathbb{R}^{cT}$. We want to remove the additional $cT - T$ hypotheses while attempting to preserve margin(x_i) for $i = 1, \dots, n$. Let U be an $n \times cT$ matrix that contains the margin of all hypotheses $h_1, \dots h_{cT}$ on all training examples $x_1, \dots x_n$, that is, $U_{ij} = y_i h_j(x_i)$. We can normalize $w = w/||w||_1$ without changing the predictions of sign$(f(x))$. This conveniently lets us compute margins by

$$(Uw)_i = y_i \frac{\sum_{t=1}^{cT} w_t h_t(x_i)}{||w||_1} = \text{margin}(x_i) \qquad \text{such that} \tag{2.7}$$

$$p_{cT} := \text{"minimal margin of } f_{cT}(x)\text{"} = \min_i (Uw)_i. \tag{2.8}$$

Removing the additional $cT - T$ hypotheses while preserving all margins can then be phrased as follows: find a vector $w' \in \mathbb{R}^{cT}$ with $||w'||_1 = 1$ that has at most T non-zero entries $||w'||_0 \le T$ such that "the error" $||Uw - Uw'||_\infty$ is not too large. [2] Our Sparsification Theorem allows us to do exactly this.

Theorem 1. *(Sparsification Theorem [1]) For $U \in [-1, 1]^{n \times m}, w \in \mathbb{R}^m$ where $||w||_1 = 1, T \le m$ and $n/T \ge 2$ there exists a vector w' where $||w'||_1 = 1$ and $||w'||_0 \le T$ so $||Uw - Uw'||_\infty = O(\sqrt{\log(n/T)/T})$.*

Using our Sparsification Theorem, we sketch pseudocode for SparsiBoost in Algorithm 1.

[2]The infinity norm of $x \in \mathbb{R}^d$ is defines as $||x||_\infty := \max_{i=1}^d |x_i|$.

Algorithm 1 SparsiBoost

```
Input:   Training data $\langle(x_1,y_1),\dots,(x_n,y_n)\rangle$ and target hypotheses $T$.
Output:  $f(x)=\sum_t w'_t h_t(x)$ with gap $v=O(\sqrt{\log(n/T)/T})$ where $||w'||_0 \le T$.

1: Train $f_{cT}(x)=\sum_{t=1}^{cT} w_t h_t(x)$ using AdaBoostV for $c=\lceil \log(n)/\log(n/T)\rceil$.
2: Let $U\in[-1,+1]^{n\times cT}$ with $U_{ij}=y_i h_j(x_i)$.
3: Normalize $w=w/||w||_1$ and use Theorem 1 to find $w'$.

Return:  $f(x)=\sum_t w'_t h_t(x)$.
```

We now sketch how to prove Algorithm 1 promises a gap $v=O(\sqrt{\log(n/T)/T})$. In line 1, AdaBoostV returns $f_{cT}(x)$ promising a minimal margin p_{cT} with gap $v=p^*-p_{cT}=O(\sqrt{\log(n)/(cT)})$. In line 2, the matrix U is defined such that the minimal margin can be written as $p_{cT}=\min_i(Uw)_i$. Finally, in line 3, we use Theorem 1 to compute w' which is used to construct the output $f(x)$ with a minimal margin $p=\min_i(Uw')_i$. Combining these observations allows us to bound the gap of $f(x)$.

$$
\begin{aligned}
v &= p^*-p && (2.9)\\
&= (p^*-p_{cT})+(p_{cT}-p) && (2.10)\\
&\le O(\sqrt{\log(n)/(cT)})+||Uw-Uw'||_\infty && (2.11)\\
&\le O(\sqrt{\log(n)/(cT)})+O(\sqrt{\log(n/T)/T}). && (2.12)
\end{aligned}
$$

By choosing $c=\lceil\log(n)/\log(n/T)\rceil$, we get that $p^*-p=O(\sqrt{\log(n/T)/T})$.

Preserving All Margins. Theorem 1 guarantees that $||Uw-Uw'||_\infty=\max_i|(Uw)_i-(Uw')_i|$ is not too large. SparsiBoost uses the guarantee to preserve the minimal margin. However, the guarantee is much stronger than the *minimal* margin, the guarantee holds for *all* margins. We can thus use Theorem 1 to reduce the number of hypotheses of any voting classifier while approximately preserving *all* margins. We demonstrated this with LightGBM [30], an efficient implementation of Gradient Boosting [36, 30]. We first train $T=500$ hypotheses with LightGBM and plot margins at $T'=80$ and $T=500$ in Figure 2.3. We then "sparsify" the $T=500$ classifier from 500 to 80 hypotheses by using Theorem 1. The margins of the sparsified classifier are more similar to the $T=500$ classifier than the $T'=80$ classifier.

To our surprise, the margins of the $T=500$ classifier are typically smaller (worse) than the margins of the $T'=80$ classifier. Margin theory thus favors the $T'=80$ classifier, however, contrary to margin theory, we found that both the $T=500$ classifier and the sparsified classifier have better validation error. Later work clarifies why margins are insufficient for explaining gradient boosting [23]. Informally, [23] find that most hypotheses (trees) return near-zero values on all but a few training points, barely altering most predictions. They prove a generalization bound that incorporates the magnitude of these predictions. Experimentally, their improved bound seems to predict the performance of gradient boosting.

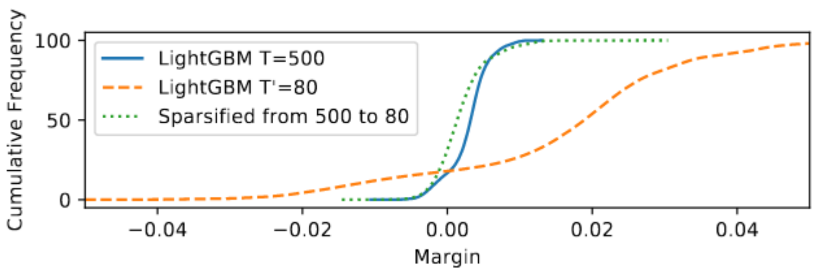


Figure 2.3: Each line represents a classifier, depicting the cumulative margins as done in Figure 2.2.

2.3 Our Lower Bound: SparsiBoost is Optimal

We claimed there exists training examples $(x_1,y_1),\dots,(x_n,y_n)$ and a hypothesis set $H=\{h_1,...,h_n\}$ such that for all relevant values of T, any classifier $\sum_{j=1}^{T} w_j h_j$ will have a gap of at least $\Omega(\sqrt{\log(n/T)/T})$. Let $A_{ij} := h_j(x_i)y_i \in \{\pm 1\}$ so the minimal margin of $\sum_{j=1}^{n} \hat{w}_j h_j(x_i)$ is $\min_i (Aw)_i$. It then suffices to show that there exists a matrix $A\in\{\pm1\}^{n\times n}$ such that:

$$\overbrace{\max_{w\in\mathbb{R}^n,\ ||w||_1=1} \ \min_i \ (Aw)_i}^{\rho^*\text{ : best (largest) minimal margin}} \quad - \quad \overbrace{\max_{w\in\mathbb{R}^n,\ ||w||_1=1,\ ||w||_0\le T} \ \min_i \ (Aw)_i}^{\rho^*_T\text{ : best (largest) minimal margin with }\le T\text{ hypotheses}} \quad \ge \Omega(\sqrt{\log(n/T)/T}) \tag{2.13}$$

For such a matrix A, any data set X and hypothesis set H for which $A_{ij}=h_j(x_i)y_i$ implies the lower bound. The following theorem states a suitable matrix A exists.

Theorem 2. *[1] There exists a universal constant $C>0$ such that for all sufficiently large n and all T with $\ln n \le T \le n/C$ there exists a matrix $A\in\{\pm1\}^{n\times n}$ such that*

$$1)\quad \rho^* \quad \ge \quad -O(1/\sqrt{n}) \tag{2.14}$$

$$2)\quad \rho^*_T \quad \le \quad -\Omega(\sqrt{\lg(n/T)/T}) \tag{2.15}$$

This leads to a lower bound on the best (smallest) possible gap:

$$\rho^* - \rho^*_T \ge -O(1/\sqrt{n}) - (-\Omega(\sqrt{\lg(n/T)/T}) = \Omega(\sqrt{\lg(n/T)/T}). \tag{2.16}$$

Note how point 1 and 2 are contrary to each other. Informally, point 1 states that the data set X is not too hard for H, in the sense that H can attain a larger (better) minimal margin than $-O(1/\sqrt{n})$. On the other hand, point 2 states that X is not too easy for H, in the sense that any T hypotheses cannot attain a larger (better) margin than $-\Omega(\sqrt{\lg(n/T)/T})$. These two contrary statements, X is not too hard or too easy for H, ensures that the gap between ρ^* and ρ^*_T is at least $\Omega(\sqrt{\lg(n/T)/T})$.

The first idea is to separate these two contrary points. To this end, we prove that any matrix which satisfies point 2 can be turned into a matrix that satisfies both points. The following lemma states that there exists a matrix which satisfies point 2.

Lemma 1. *[1] There exists a universal constant $C>0$ such that for all sufficiently large n and all T with $\ln(n)\le T\le n/C$, there exists $B\in\{\pm1\}^{n\times n}$ such that for every $w\in\mathbb{R}^n$ with $||w||_0\le T$ and $||w||_1=1$*

$$\min_i (Bw)_i \le -\Omega(\sqrt{\ln(n/T)/T}) \tag{2.17}$$

We now prove Theorem 2 using Lemma 1.

Proof. Let $B\in\{\pm1\}^{n\times n}$ satisfy Lemma 1. Spencer's theorem [1] states that there exists a vector $x\in\{\pm1\}^n$ such that $||Bx||_\infty = O(\sqrt{n\ln(en/n)}) = O(\sqrt{n})$. Construct a new matrix $A_{ij}=B_{ij}x_j$ such that the j'th column is scaled by x_j. Let $v_i = 1/n$ then $Av = Bx/n$ so $||Av||_\infty = ||Bx||_\infty/n = O(\sqrt{n}/n) = O(1/\sqrt{n})$. Because $||x||_\infty := \max_i |x_i|$ then $\min_i(Av)_i \ge -O(1/\sqrt{n})$ so A satisfies point 1.

It remains to show that A also satisfies point 2. Consider any $w\in\mathbb{R}^d$ so $||w||_0\le T$ and $||w||_1=1$. If we let $\hat{w}_j = w_j\cdot x_j$ then $Aw=B\hat{w}$ so $\min_i(Aw)_i = \min_i(B\hat{w})_i$. Note that $\hat{w}$ satisfies the necessary conditions because x can only flip signs: $x\in\{\pm1\}^n$ implies $||\hat{w}||_1 = ||w||_1 = 1$ and $||\hat{w}_0|| = ||w||_0 \le T$. As a result $\min_i(Aw)_i = \min_i(B\hat{w})_i \le -\Omega(\sqrt{\ln(n/T)/T})$ so A satisfies point 2.

□

The proof of Lemma 1 is not a part of this thesis. It can be found in the full ArXiV version [1].

2.4 From Prior Generalization Bounds [8, 12, 21] to Our Bound [2]

In the previous section, we saw how Breiman's minimal margin generalization bound [8] led to the development of several algorithms [10, 24, 43, 44]. Breiman's generalization bound is particularly simple because it depends only on the minimal margin. There exists other bounds which, in a certain sense, depends on the margins of all training examples [8, 21, 50]. Similar to how [40] conjectured it was impossible to improve AdaBoostV, we may question whether it is possible to improve the best generalization upper bound [21]. Indeed, we managed to prove a margin-based generalization *lower bound*, which almost matches the strongest margin-based generalization *upper bound*. We shall start by reviewing the previous margin-based generalization upper bounds [8, 12, 21] comparing their relative strengths. Then, first when we appreciate how the previous *upper* bounds relate to each other, will we consider our *lower* bound.

Recall that $\sum_t \alpha_t h_t(x)$ is a voting classifier if $\alpha_t \geq 0$ and $\sum_j \alpha_j = 1$. The first margin-based generalization bound was due to Schapire and Freund [8] and takes the following form. Let X be a set and let D denote an unknown distribution over $X \times \{-1,+1\}$ and assume we have training data $S = \langle (x_1,y_1),\ldots,(x_n,y_n)\rangle$ where $(x_i,y_i) \sim D$. Then, with high probability over sampling S from D^n, it holds for every margin $\theta \in (0,1]$ that all voting classifiers f satisfies:

$$\underbrace{\Pr_{(x,y)\sim D}[f(x)y<0]}_{E_{out}\text{: out-of-sample error}} \leq \underbrace{\Pr_{(x,y)\sim S}[yf(x)<\theta]}_{\Pr_{S,\theta}\text{: fraction of S with margin }<\theta} + O\left(\underbrace{\sqrt{\frac{\log(|H|)\log(n)}{\theta^2 n}}}_{\Omega_{SF}\text{: ``complexity'' term}}\right) \tag{2.18}$$

The left-hand side is the out-of-sample error E_{out}. We want a classifier f that minimizes E_{out}. This is difficult because E_{out} depends on D, which is an unknown distribution. In particular, the only information we have about D stem from S which contains n independently and identically distributed samples $(x_i,y_i) \sim D$. Since the upper bound in Equation (2.18) holds for any voting classifier, we may instead find a voting classifier f that minimizes the upper bound. Informally, the upper bound is small for f which have large margins on a large fraction of S.

The upper bound consists of two parts: $\Pr_{S,\theta}$ and Ω_{SF}. The first part $\Pr_{S,\theta}$ uses the notation "$(x,y) \sim S$" which means that (x,y) are drawn uniformly at random from S. As a result, $\Pr_{S,\theta}$ is the fraction of training examples (x,y) from S where $\text{margin}(x) = yf(x) < \theta$. The second part Ω_{SF} depends on $|H|$ which limits how "complicated" the hypothesis set H can be.[3] Furthermore, Ω_{SF} decrease with larger θ. The margin θ thus controls a trade-off between $\Pr_{S,\theta}$ and Ω_{SF}. A large θ improves (decreases) Ω_{SF} but weakens (increases) $\Pr_{(x,y)\sim S}$. We can minimize $\Pr_{S,\theta}$ during training by finding a voting classifier f that maximizes the margins of most training examples. This is what we meant by "informally, the upper bound is small for f with large margins on a large fraction of S" in the previous paragraph.

As we have previously seen, Breiman [12] introduced a generalization bound that depends only on the minimal margin. We restate it here for convenience. With high probability over sampling $S \sim D^n$ it holds for all voting classifiers f that:

$$\underbrace{\Pr_{(x,y)\sim D}[f(x)y<0]}_{E_{out}\text{: out-of-sample error}} \leq O\left(\underbrace{\frac{\log(|H|)\log(n)}{\widehat{\theta}^2 n}}_{\Omega_B\text{: ``complexity'' term}}\right) \quad \text{where} \quad \underbrace{\widehat{\theta} := \min_{i=1}^{n} \text{margin}(x_i).}_{\text{minimal margin}} \tag{2.19}$$

The Schapire-Freund bound holds for every margin $\theta \in (0,1]$, so we can compare it to Breiman's bound by considering the special case where $\theta = \widehat{\theta}$. This simplifies the Schapire-Freund bound because $\Pr_{S,\widehat{\theta}} = 0$, i.e., there are 0 points in S that have a smaller margin θ than the minimal margin $\widehat{\theta}$.

[3]We assume H is finite. A similar bounds can be proven for infinte H if one replaces $|H|$ by $d\log(n)$ where d is the Vapnik–Chervonenkis (VC) dimension of H. This is true for all three bounds in this section [8, 12, 21].

$$\underbrace{\Pr_{(x,y)\sim D}[f(x)y<0]}_{E_{out}\text{: out-of-sample error}} \le O\left(\underbrace{\sqrt{\frac{\log(|H|)\log(n)}{\widehat{\theta}^2 n}}}_{\Omega_{SF}\text{: ``complexity'' term}}\right) \quad \text{where} \quad \underbrace{\widehat{\theta} := \min_{i=1}^{n} \text{margin}(x_i).}_{\text{minimal margin}} \tag{2.20}$$

The only difference between these two bounds is a square root in the complexity terms, that is, $\Omega_{SF} = \sqrt{\Omega_B}$. We care about probabilities < 1 so the square root increases (weakens) the upper bound, therefore Breiman's bound is the stronger bound. Even though Breiman's bound is stronger, the minimal margin is very sensitive to outliers. In particular, one adversarially chosen outlier may force the minimal margin to be negative such that Breiman's bound does not apply.

The currently strongest bound is the k'th margin bound due to [21] which interpolates between the Schapire-Freund bound and Breiman's bound. The k'th margin bound states that with high probability over S it holds for every margin $\theta \in (0,1]$ and every voting classifier f that

$$\underbrace{\Pr_{(x,y)\sim D}[f(x)y<0]}_{E_{out}} \le \underbrace{\Pr_{(x,y)\sim S}[yf(x)<\theta]}_{\Pr_{S,\theta}} + O\left(\underbrace{\frac{\log(|H|)\log(n)}{\theta^2 n}}_{\Omega_k} + \sqrt{\underbrace{\Pr_{(x,y)\sim S}[yf(x)<\theta]}_{\Pr_{S,\theta}}\underbrace{\frac{\log(|H|)\log(n)}{\theta^2 n}}_{\Omega_k}}\right) \tag{2.21}$$

If we choose $\theta = \widehat{\theta}$ then $\Pr_{S,\theta} = 0$ and the only term left is Ω_k which recovers Breiman's bound. Furthermore, $O(\Omega_k + \sqrt{\Pr_{S,\theta}\,\Omega_k})$ is no larger than the corresponding term in the Schapire-Freund bound, $O(\Omega_k + \sqrt{\Pr_{S,\theta}\,\Omega_k}) = O(\sqrt{\Omega_k})$. This is true because (1) $\Pr_{S,\theta} \le 1$ and (2) the bound is meaningful only when $\Omega_k < 1$ for which $\Omega_k < \sqrt{\Omega_k}$. Therefore, the k'th margin bound is, asymptotically, never worse than the Schapire-Freund bound. This concludes our review of the margin-based generalization *upper bounds* from [8, 12, 21]. We now turn our attention towards our own margin-based generalization *lower bound.*

For simplicity, we start by considering a special case of our generalization lower bound which is comparable to Breiman's generalization upper bound. This simplifies everything because we only need to consider the minimal margin instead of "all margins." Informally, Breiman's bound tells us that, with high probability, *all* voting classifiers with large (good) minimal margin $\widehat{\theta}$ leads to low (good) out-of-sample error $E_{out} \le \Omega_B$. Conversely, a matching lower bound would tell us that, with at least some probability, there *exist* a voting classifiers with large (good) minimal margin $\widehat{\theta}$ which has at least out-of-sample error $E_{out} \ge \Omega_B$. If this was the case, any improvements to Breiman's generalization upper bound would violate our generalization lower bound. We state the special case of our lower bound in the following corollary.

Corollary 1. *For every sufficiently large integer N, it holds that for every $\theta \in (1/N, 1/40)$ and every $(\theta^{-2}\log(N))^{1+\Omega(1)} \le n \le 2^{N^{O(1)}}$ that there exists a set X, a hypotheses set H over X and a distribution D over $X \times \{-1,+1\}$ such that* $\log(|H|) = \Theta(\log(N))$ *and with probability at least* $1/100$ *over sampling $S \sim D^n$ there exists a voting classifier f such that*

$$1: \quad \underbrace{\min_{(x,y)\in S} f(x)y}_{\widehat{\theta}\text{: minimal margin}} > \theta \qquad \textit{“good” minimal margin}$$

$$2: \quad \underbrace{\Pr_{(x,y)\sim D}[f(x)y<0]}_{E_{out}} \ge \Omega\left(\underbrace{\frac{\log(|H|)\log(n)}{\theta^2 n}}_{\Omega_k}\right) \qquad \textit{“bad” generalization}$$

Point 1 tells us that the classifier f attains a minimal margin larger than θ (good minimal margin). Point 2 tells us that f attains larger generalization error than Ω_k (bad generalization). If we insert $\widehat{\theta} \geq \theta$ from point 1 into point 2 we get that

$$E_{out} \geq \Omega\left(\frac{\log(|H|)\log(n)}{\theta^2 n}\right) > \Omega\left(\frac{\log(|H|)\log(n)}{(\min_{(x,y)\in S} f(x)y))^2 n}\right) = \Omega\left(\frac{\log(|H|)\log(n)}{\widehat{\theta}^2 n}\right). \tag{2.22}$$

This matches Breiman's generalization bound and thus rule out any improvements.

The general case of our lower bound relies on all margins instead of the minimal margin $\widehat{\theta}$. In particular, it relies on $\Pr_{S,\theta} \leq \tau$ for some $\tau \in [0, 49/100]$. The inequality $\Pr_{S,\theta} \leq \tau$ means that no more than a fraction τ of the training examples have margin less than θ, i.e., the voting classifier f attains overall "good margins" removing the emphasis on the minimal margin. Informally, the theorem states that there exists a classifier with "good margins" that has out-of-sample error E_{out} which is (almost) at least that predicted by the k'th margin bound. It is therefore (almost) impossible to improve the k'th margin generalization upper bound. We clarify what "almost" means after stating the theorem.

Theorem 3. *For every sufficiently large integer N, it holds that for every $\theta \in (1/N, 1/40), \tau \in [0, 49/100]$ and n so $(\log(N)/\theta^{-2})^{1+\Omega(1)} \leq n \leq 2^{N^{O(1)}}$ that there exists a set X, a hypothesis set H over X and a distribution D over $X \times \{-1,+1\}$ such that $\log(|H|) = \Theta(\log(N))$ and with probability at least $1/100$ over sampling $S \sim D^n$ there exists a voting classifier f so*

$$1.\ \underbrace{\Pr_{(x,y)\sim S}[yf(x) < \theta]}_{\Pr_{S,\theta}} \leq \tau \qquad \text{“good” margins} \tag{2.23}$$

$$2.\ \underbrace{\Pr_{(x,y)\sim D}[f(x)y < 0]}_{E_{out}} \geq \tau + \Omega\left(\underbrace{\frac{\log(|H|)\log(n)}{\theta^2 n}}_{\Omega_B} + \sqrt{\tau \cdot \frac{\log(|H|)}{\theta^2 n}}\right) \qquad \text{“bad” generalization} \tag{2.24}$$

Similar to Corollary 1, point 1 tells us that f attains "good" margins and point 2 tells us that f has "bad" generalization. If we insert $\Pr_{S,\theta} \leq \tau$ from point 1 into point 2 we see that

$$E_{out} \geq \Pr_{(x,y)\sim S}[yf(x) < \theta] + \Omega\left(\frac{\log(|H|)\log(n)}{\theta^2 n} + \sqrt{\Pr_{(x,y)\sim S}[yf(x) < \theta] \cdot \frac{\log(|H|)}{\theta^2 n}}\right) \tag{2.25}$$

Comparing this to Equation (2.21), we see that it is only off by a factor $\log(n)$ inside the square root. This is what we meant by the bound being "almost" tight. It is possible one can improve the k'th margin bound by this factor $\log(n)$, however, further improvements are ruled out by our lower bound.

Algorithmic Variant. Theorem 3 tells us there exists a classifier with good margins and bad generalization error. While this is sufficient for a margin-based generalization lower bound, it says nothing about any particular training algorithm. We proved a variant of Theorem 3 that holds for all training algorithms. Informally, it states the following: there exists a hypothesis set H such that for any algorithm A there exists a distribution D such that, with high probability, the classifier $f_{A,S}$ found by A on a sample $S \sim D^n$ has bad generalization error. We cannot guarantee that $f_{A,S}$ gets good margins (because we make no assumptions on A), however, we can guarantee that there exists a classifier with good margins which A could have chosen.

Deriving Corollary 1. To see Corollary 1 follows Theorem 3, choose $\tau = 0$ so $\Pr_{S,\theta} = 0$. This means that there are no points with margin less than θ, that is, the minimal margin must be at least $\widehat{\theta} > \theta$. Furthermore, substituting $\tau = 0$ into point 2 gives exactly the bound in Corollary 1. In the following section we a variant of Corollary 1 (see Theorem 4) without assuming Theorem 3. The goal of proving Theorem 4 is to prepare the reader for the much more complicated proof of Theorem 3 in [2].

2.5 Our Generalization Lower Bound: Breiman's Bound is Optimal

Our proofs in [2] take up 12 pages. This section sketches the minimal margin special case within 1 page. The goal is to prepare the reader for the more involved proof of Theorem 3 (see Theorem 2 in [2]). Let $C(H) := \{\sum_j \alpha_j h_j \mid h_j \in H, \sum_j \alpha_j = 1, \alpha_j \geq 0\}$. Intuitively, we want a hypothesis set H and a "hard" distribution D over X, such that, with constant probability over sampling $S \sim D^n$, there exists a classifier $f \in C(H)$ with "good" margins but "bad" generalization error.

Theorem 4. *For sufficiently large N, every* $\theta \in (1/N, 1/40)$ *and n so* $\ln(N)/\theta^2 \leq n^{1/2}$ *and* $n \leq 2^{N^{O(1)}}$ *there exists a set X, hypothesis set H and distribution D, such that:* $\ln(|H|) = \Theta(\ln(N))$ *and with probability at least* $49/100$ *over sampling* $S \sim D^n$ *there exists a voting classifier* $f \in C(H)$ *such that:*

1. $\min_{(x,y)\in S} f(x)y \geq \theta$ *"good" minimal margin* (2.26)
2. $\mathbb{E}_{(x,y)\sim D}[\mathbb{1}_{f(x)y<0}] \geq \Omega\left(\frac{\ln(|H|)\ln n}{\theta^2 n}\right)$ *"bad" generalization* (2.27)

Let $X = \{x_1, ..., x_u\}$ with labels $l(x) = 1$ for all $x \in X$. The distribution D then assigns uniform probability to all points $\Pr_{x\sim D}[x = x_i] = 1/u$. Let $u = O(n/\ln(n))$ where $n = |S|$. By a coupon-collector argument, one can show that, with probability at least $1/2$ over sampling $S \sim D^n$, there are at least $n^{1/2} \geq \ln(N)/\theta^2$ "unseen" points $|X \backslash S|$. We then create an "adversarial classifer" which wrongly classifies a set $I_S \subseteq X \backslash S$ containing $|I_S| = \ln(N)/\theta^2$ "unseen" points. The points in I_S from $X \backslash S$ are chosen uniformly at random.

$$h_S(x) := \begin{cases} 1 & \text{if } x \notin I_S \\ -1 & \text{if } x \in I_S \end{cases} \tag{2.28}$$

Note that h_S has the best best possible minimal margin $\min_{(x,y)\in S} h_s(x)y = 1$. At the same time, we can lower bound the generalization error of h_S:

$$\mathbb{E}_{x\sim D}[\mathbb{1}_{l(x)h_S(x)<0}] = \sum_{x_1,\ldots,x_u} \mathbb{1}_{l(x_i)h_S(x_i)<0} \cdot \Pr_D[x_i] = \sum_{x\in I_S} \Pr_D[x] = \frac{|I_s|}{u} \geq \Omega\left(\frac{\ln(N)\ln(n)}{\theta^2 n}\right) \tag{2.29}$$

We now construct the hypothesis set H so $\ln(|H|) = \Theta(\ln(N))$. Instead of showing the existence of H so $h_S \in C(H)$, we show that there exists H with a $f \in C(H)$ such that f "approximates" h_S.

Lemma 2. *(Claim 5 in [2]) For* $|X| \leq 2^{N^{O(1)}}$ *there exists a hypothesis set H with* $\ln(|H|) = \Theta(\ln(N))$ *so*

$$\Pr_{l \in_R \mathbb{L}}[\exists f \in C(H) : \forall x \in X : l(x)f(x) \geq \theta] \geq 1 - 1/N \tag{2.30}$$

$$\mathbb{L} := \{l \in \{\pm 1\}^u \ : \ |\{l_i = -1\}| = \ln(N)/\theta^2\} \tag{2.31}$$

By $l \in_R \mathbb{L}$ *we mean that l is chosen uniformly at random from* $\mathbb{L}$.

Note that $l(h_S) := (h_S(x_1), \ldots, h_S(x_u)) \in \mathbb{L}$. We now state Equation (2.33) which we justify below.

$$1 - 1/N \leq \Pr_{l\in_R\mathbb{L}}[\exists f \in C(H) : \forall x \in X : l(x) \quad f(x) \geq \theta] \tag{2.32}$$

$$= \Pr_{S\sim D^n}[\exists f \in C(H) : \forall x \in X : h_S(x)f(x) \geq \theta \quad | \quad |X\backslash S| \geq \ln(N)/\theta^2 \ ] \tag{2.33}$$

The equality is true, because, when we condition on $|X \backslash S| \geq \ln(N)/\theta^2$, the resulting $l(h_S)$ are distributed uniformly at random $l(h_S) \in_R \mathbb{L}$. To see why $l(h_S) \in_R \mathbb{L}$, recall that $\Pr_{x\sim D}[x = x_i] = \Pr_{x\sim D}[x = x_j]$ for all i, j and the points in I_S are uniformly random from $X \backslash S$.

There thus exists H with $\ln(|H|) = \Theta(\ln(N))$ such that, with probability at least $1/2 \cdot (1 - 1/N)$ over sampling $S \sim D^n$, the following two conditions holds: (1) $|X \backslash S| \geq \ln(N)/\theta^2$ and (2) there exists $f \in C(H)$ such that $h_S(x)f(x) \geq \theta$ for all $x \in X$. We now show that that for $N > 100$ the two conditions implies that, with probability at least $49/100$, the resulting f has good minimal margin and bad generalization. **Good minimal margin:** note that $h_S(x) = 1$ for $x \in S$, so $h_S(x)f(x) \geq \theta$ implies that $f(x) \geq \theta$ for all $x \in S$. Since $y = l(x) = 1$ we get that $\min_{(x,y)\in S} f(x)y \geq \theta$ as wanted. **Bad generalization:** because $\theta > 0$ then $\text{sign}(f) = \text{sign}(h)$ so $E_{out}(f) = E_{out}(h) \geq \Omega(\ln(|H|)\ln(n)/(\theta^2 n))$.

Chapter 3

Neural Networks

Neural networks are currently used in many different domains, e.g., protein structure prediction [27], drug discovery [42], image classification [33] and natural language processing [13]. Recent work on *scaling laws* [13, 26] demonstrate that, empirically, certain neural networks consistently improve when one increase data, parameters and computing power. While this may not sound too surprising, it suggests that for areas with practically unlimited data, we can improve machine learning models by simply increasing computational resources. According to estimates by [6], the amount of computing power used to train large neural networks increased by a factor 300 000 from 2012 to 2018, see Figure 3.1. On average, this corresponds to a doubling every 3.4 months, much faster than the historic doubling of the number of transistors on a microchip every 2 years, commonly referred to as Moore's law [39]. From this perspective, algorithmic research presents a desirable alternative to naively increasing our computational budget. If we identify algorithmic problems within neural network training, we may find faster algorithms that does exactly the same, which would allow us to use more data and more parameters without increasing our computational resources. This chapter presents such an algorithmic improvement.

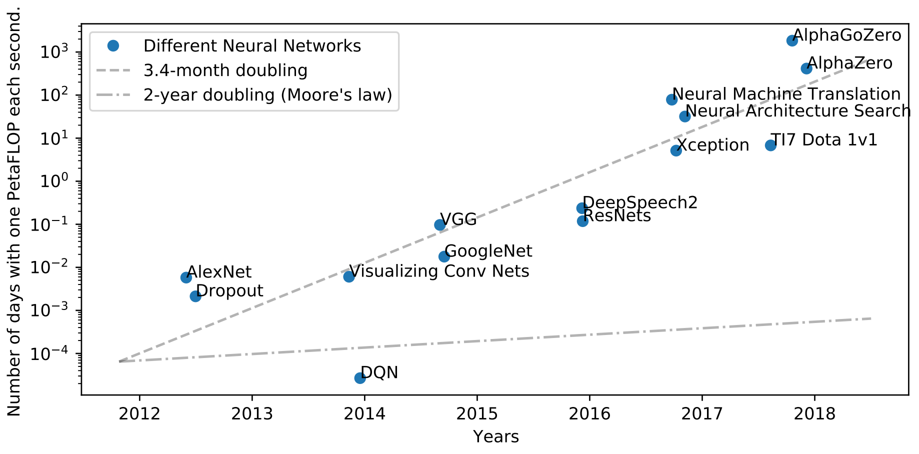


Figure 3.1: Figure from [6]. Each dot show the total amount of compute used to train a model and the year the model was published. The compute budget seems to have increased faster than Moore's law.

What are Neural Networks? The literature contains many different types of neural networks. For our purposes, it is convenient to start with Fully Connected Neural Networks (FCNNs), and extend to other types only when we need to. A FCNN is a function $f:\mathbb{R}^d\rightarrow\mathbb{R}^k$ composed of affine transformations $L_i(x)=W_ix+b_i$ where $W_i\in\mathbb{R}^{d_i\times d_{i+1}}, b_i\in\mathbb{R}^{d_{i+1}}$ and entry-wise non-linearities like $\sigma(x)=\max(0,x)$. For example, $f(x)=L_1(\sigma(L_2(x)))$ is a 2-layer FCNN.

3.1 Neural Networks and the Singular Value Decomposition

Neural networks sometimes use time-consuming operations on their weight matrices. For example, previous work [31] use a loss function which depends on matrix determinants. For a $d \times d$ weight matrix W it takes $O(d^3)$ to compute $\det(W)$ naively. We can compute $\det(W)$ faster given the Singular Value Decomposition (SVD) $W = U\Sigma V^T$ for diagonal Σ and orthogonal $U^TU = V^TV = I$. In particular, we can compute $|\det(W)| = \prod_{i=1}^{d} |\Sigma_{ii}|$ in $O(d)$ time, however, computing the SVD takes $O(d^3)$ time. Previous work mitigate the $O(d^3)$ computation by representing W implicitly in its SVD [52].

We now exemplify neural network training. Consider a neural network $n(X) = \max(WX, 0.1 \cdot WX)$ where $X \in \mathbb{R}^{d\times m}$ and $W \in \mathbb{R}^{d\times d}$. We want to train $n(x)$ with gradient descent while the loss function depends on $|\det(W)|$ as in [15, 16, 31]. The training loop is sketched in Algorithm 2.

Algorithm 2 Training Loop

```
1: for $i = 1$ to $k$ do
2:     Take mini-batch $X \in \mathbb{R}^{d\times m}$ with $m$ examples from the training data.
3:     Compute the SVD $W = U\Sigma V^T$.
4:     Compute $D := \log|\det(W)| = \sum_{i=1}^{d} \log(|\Sigma_{ii}|)$.
5:     Take the gradient descent step $W = W - \eta\nabla_W \text{loss}(X, n(X), D)$ for a step size $\eta > 0$.
6: end for
```

Throughout all of training, line 3 takes $O(kd^3)$ time, but allows us to compute line 4 in $O(kd)$ time. Previous work mitigate the SVD computation in line 3 by implicitly representing W in its SVD [52]. In theory, this reduces the time complexity of line 3 and 4 from $O(kd^3)$ to $O(kd)$ time, however, we find no improvements in practice for $d < 512$, see Figure 3.2. The poor performance occurs because the technique from [52] is sequential and thus ill-suited for GPUs. In particular, their algorithm performs $O(d)$ sequential matrix-vector multiplications. [52] tried to mitigate the poor GPU performance by using a "more parallel" $O(kd^3)$ algorithm. While their $O(kd^3)$ algorithm is faster in practice, it is, asymptotically, no faster than computing the SVD naively. Our main result is a new algorithm, FastH, which reduces the number of sequential stages from $O(d)$ sequential matrix-*vector* multiplications to $O(d/m + m)$ sequential matrix-*matrix* multiplications **without** increasing the asymptotic time complexity. This allows us to use SVDs in Neural Networks much faster than naively computing the SVDs, see Figure 3.2.

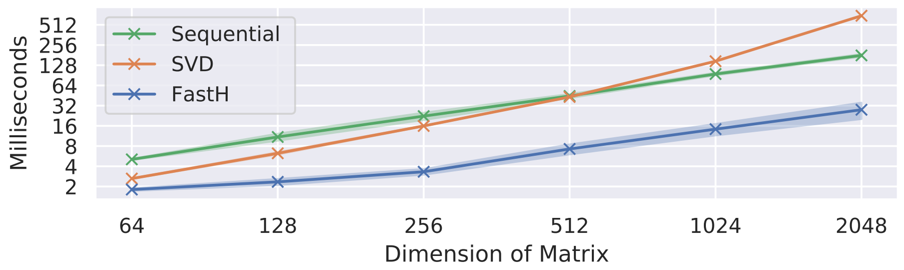


Figure 3.2: Time of line 2-4 in Algorithm 2 with different approaches to handling the SVD in line 3. **Seq**: time of the sequential algorithm from [52]. **SVD**: time to compute the SVD using PyTorch [41] **FastH**: time of our algorithm.

The use of SVDs with FastH goes beyond matrix determinants. One can achieve algorithmic improvements for several matrix operations used during training of neural networks, we present four examples in Section 3.1.1. We then sketch the technique from [52] in Section 3.2 and clarify why it struggles on GPUs. We subsequently introduce our FastH algorithm in Section 3.3. In Section 3.4.1 we show how to decrease the sequential stages used by FastH from $O(d/m + m)$ to $O(d/m + \log(m))$. In Section 3.4.2 we show that $O(1)$ sequential stages are sufficient to "express" the SVD.

3.1.1 Applications

Many operations used by neural networks can be computed faster when given the SVD. In this subsection we sketch five different applications.

Determinant. As alluded to earlier, some neural networks [15, 16, 31] need to compute the matrix determinant $|\det(W)|$. The naive algorithm takes $O(d^3)$ time. Using the SVD, we see that

$$|\det(W)| = |\det(U\Sigma V^T)| = |\det(U)\cdot\det(\Sigma)\cdot\det(V)| = |\det(\Sigma)| = \left|\prod_{i=1}^{d}\Sigma_{ii}\right|. \tag{3.1}$$

The SVD thus allows us to compute $|\det(W)| = |\prod_{i=1}^{d}\Sigma_{ii}|$ in $O(d)$ time.

Inverse. Some neural networks need to compute matrix inverses [15, 16, 31]. The naive algorithm takes $O(d^3)$ time. Using the SVD, we see that

$$W^{-1} = (U\Sigma V^T)^{-1} = (V^T)^{-1}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^T. \tag{3.2}$$

Since Σ is diagonal, we can compute its inverse in $O(d)$ time. This subsequently allows us to compute $W^{-1}x$ for $x \in \mathbb{R}^d$ in $O(d^2)$ time.

Weight Decay. Regularizers help Neural Networks mitigate overfitting. One of the most popular regularizers is weight decay, which simply adds $||W||_F^2 := \sum_{i,j=1}^{d} W_{ij}^2$ to the loss function. It takes $O(d^2)$ time to compute $\sum_{i,j=1}^{d} W_{ij}^2$. Using the SVD, we see that

$$||W||_F^2 = ||U\Sigma V^T||_F^2 = ||\Sigma||_F^2 = \sum_{i=1}^{d}\Sigma_{ii}^2. \tag{3.3}$$

The SVD thus allows us to compute weight decay $||W||_F^2 = \sum_{i=1}^{d}\Sigma_{ii}^2$ in $O(d)$ time. Furthermore, the SVD provides fine-grained control over the singular values $\Sigma_{11},\ldots,\Sigma_{dd}$. For example, we may enforce $\Sigma_{ii} \in [1 \pm \varepsilon]$ for some small $\varepsilon > 0$ as done in [52].

Spectral Normalization [9, 38]. Some neural networks normalize weights to $W = W/(\max_i \Sigma_{ii})$. Usually, one approximates $\max_i \Sigma_{ii}$ with the *power iteration* algorithm [38], which takes $O(d^2 r)$ time, where larger r reduces approximation errors. If we have access to the SVD, we have access to Σ and can compute $\Sigma = \Sigma/(\max_i \Sigma_{ii})$ in $O(d)$ time.

Taylor Series. We can represent symmetric matrices $W = W^T$ in their eigendecomposition $W = U\Sigma U^T$. For any function with a Taylor expansion $f(x) = \sum_{t=0}^{\infty} c_t x^t$, we can define a "similar" matrix function $f(W) := \sum_{t=0}^{\infty} c_t W^t$. Given $W = U\Sigma U^T$, we can compute $f(W) = Uf(\Sigma)U^T$ where $f(\Sigma)_{ii} = f(\Sigma_{ii})$ since

$$f(W) = \sum_{t=0}^{\infty} c_t W^t = \sum_{t=0}^{\infty} c_t (U\Sigma U^T)^t = \sum_{t=0}^{\infty} c_t U\Sigma^t U^T = U\left(\sum_{t=0}^{\infty} c_t\Sigma^t\right)U^T = Uf(\Sigma)U^T \tag{3.4}$$

For example, this allows us to compute the matrix exponential [51] $e^W = Ue^{\Sigma}U^T$ in $O(d)$ time.

3.2 From Prior Work [52] to Our Results [3]

Idea from [52]. Consider Algorithm 2. Instead of computing the SVD in line 3, we are going to represent W in its SVD such that the gradient update in line 5 updates W while maintaining the SVD. Informally, this allows us to use the SVD all k steps without computing the SVD k times.

Sketch of [52]. Consider line 5 in Algorithm 2 which computes $W = W - \eta\nabla_W$. Instead, we will update each factor of the SVD $W = U\Sigma V^T$.

$$U = U - \eta\nabla_U \quad \Sigma = \Sigma - \eta\nabla_\Sigma \quad V = V - \eta\nabla_V. \tag{3.5}$$

Note that Σ remains diagonal under the update $\Sigma = \Sigma - \eta\nabla_\Sigma$. This is not true for the orthogonal matrices U and V. That is, U may not remain orthogonal after the update $U = U - \eta\nabla_U$. This issue can be resolved by representing the orthogonal matrices as vectors $v_1, ..., v_d \in \mathbb{R}^d$ through the following product.

$$V = \prod_{i=1}^{d} H(v_i) \qquad \text{where} \qquad H(v) := I - 2\frac{vv^T}{||v||_2^2}. \tag{3.6}$$

We call $H(v) \in \mathbb{R}^{d\times d}$ the Householder matrix for $v \in \mathbb{R}^d$. For any $v \neq 0$ the corresponding Householder matrix $H(v)$ is orthogonal.[1] Since products of orthogonal matrices are orthogonal, it follows that V is orthogonal. One can further prove that any orthogonal matrix can be represented as the product of $O(d)$ Householder matrices [48].[2] Finally, V remains orthogonal under gradient descent updates $v_i = v_i - \eta\nabla_{v_i}$ if $v_i \neq \eta\nabla_{v_i}$. We then represent W in its SVD with weights $u_1, ..., u_d, v_1, ..., v_d, \sigma \in \mathbb{R}^d$.

$$W = \overbrace{\left[\prod_{i=1}^{d} H(u_i)\right]}^{U} \cdot \overbrace{\text{diagonal}(\sigma)}^{\Sigma} \cdot \overbrace{\left[\prod_{i=1}^{d} H(v_i)\right]}^{V^T} \tag{3.7}$$

This allows us to implicitly represent W in its SVD. Importantly, our SVD representation supports gradient descent updates with respect to u_i, v_i and σ. It normally takes $O(d^2m)$ time to compute $W \cdot X$ for $X \in \mathbb{R}^{d\times m}$. We now argue that if W is represented as in Equation (3.7) we can still compute $W \cdot X$ in $O(d^2m)$ time. Notice that we can evaluate one Householder multiplication $H(v) \cdot X$ in just $O(dm)$ time:

$$H(v) \cdot X = \left(I - 2\frac{vv^T}{||v||_2^2}\right)X = X - 2\frac{v(v^TX)}{||v||_2^2}. \tag{3.8}$$

We can thus compute $2d$ Householder multiplications in $O(d^2m)$ time, if we evaluate the multiplications sequentially from right to left. For example, we compute $V^T \cdot X$ as follows:

$$V^T \cdot X = \overbrace{H(v_1)\cdot}^{\text{Stage d}} \; \ldots \overbrace{H(v_{d-2})\cdot}^{\text{Stage 3}} \; \overbrace{H(v_{d-1})\cdot}^{\text{Stage 2}} \; \overbrace{H(v_d)\cdot X}^{\text{Stage 1}} \tag{3.9}$$

Not faster in practice? In practice, we found that the $O(d^3)$ SVD computation is often faster than the $O(d^2m)$ implicit SVD representation. In Figure 3.2, we consider $m = 32$ and $d = 128, 256, \ldots, 2048$. For $d \leq 512$ the $O(d^3)$ algorithm was faster than the $O(d^2m)$ algorithm. The poor performance of the $O(d^2m)$ algorithm occurs because the big-O notation does not consider constant factors and parallelization. Parallelization is important because the experiments was performed on a GPU, on which neural networks are almost exclusively trained. In particular, the $O(d^2m)$ algorithm requires us to sequentially evaluate $2d$ Householder multiplications. In other words, for $d = 2048$, we are asking our highly parallel GPU to perform 4096 sequential Householder multiplications.

[1]To see $H(v)$ is orthogonal try to multiple out $H(v)^TH(v)$ and realize it gives the identity matrix I.

[2]The proof Theorem 1 in [48] is particularly insightful, it utilizes a QR algorithm that relies on Householder matrices.

3.3 Our Algorithm: FastH [3]

Problem. Given $X \in \mathbb{R}^{d\times m}$ and $v_1,\ldots,v_d \in \mathbb{R}^d$ evaluate $H(v_1)\cdots H(v_d)\cdot X$ in $O(d^2m)$ time with less than $O(d)$ matrix-matrix multiplications.

Idea. There are two naive algorithms. Sequential: evaluate $\prod_{i=1}^d H(v_i)X$ sequentially as done in [52]. Parallel: compute $V = \prod_{i=1}^d H(v_i)$ by multiplying the Householder matrices together in parallel, then, afterwards, compute the matrix multiplication VX. The sequential algorithm has good time complexity but poor parallelization, contrary to the parallel algorithm, which has poor time complexity and good parallelization. It turns out one can make a hybrid algorithm that is "more parallel" than the sequential algorithm without increasing the good time complexity.

Solution Sketch. Let us first clarify what we mean by the "sequential" and "parallel" algorithm. The sequential algorithm computes $H_1(H_2(\ldots(H_dX)\ldots))$ in the order indicated by the parentheses. The order of the parentheses ensures that one only multiples Householder matrices with X instead of multiplying Householder matrices together. This yields the desired $O(d^2m)$ time complexity, however, it is exactly the order of evaluation that makes the algorithm sequential. In contrast, we can rearrange the parentheses so the Householder matrices are multiplied together in parallel, e.g., for $d=8$ we compute:

$$V = [(H_1\cdot H_2)\cdot(H_3\cdot H_4)]\cdot[(H_5\cdot H_6)\cdot(H_7\cdot H_8)] \tag{3.10}$$

This order of parentheses allows us to compute several matrix multiplications in parallel. However, if we evaluate all d matrix multiplications naively it will lead to a $O(d^4)$ time complexity.[3] This leads us to a crucial question. Is it possible to exploit the Householder structure to compute $H_i\cdots H_{i+t}$ faster? A linear algebra results from 1987 gives us exactly what we need [11]. We paraphrase their results to our needs and notation in Lemma 3.

Lemma 3. *Given $v_1,\ldots,v_t \in \mathbb{R}^d$ one can compute $A,B\in\mathbb{R}^{d\times t}$ such that $I+AB^T = \prod_{i=1}^t H(v_i)$ in $O(dt^2)$ time with $O(t)$ sequential matrix-matrix multiplications.*

The main idea is then to split $v_1,\ldots,v_d$ into d/t groups. For each group, we use Lemma 3 to compute A and B such that $G_i = I+AB^T$ equals the product of the i'th group and $G_1\cdots G_{d/t} = \prod_{i=1}^d H(v_i)$. We can compute each G_i in parallel. Furthermore, the structure of G_i allow us to compute G_iX in $O(dtm)$ time.

$$G_iX = (I+AB^T)X = X + A(B^TX) \tag{3.11}$$

We can finally compute WX by sequentially evaluating $G_1(G_2(\ldots(G_{d/t}X)\ldots)$ in $O(dtm\cdot d/t) = O(d^2m)$ time. In Algorithm 3, we present high-level pseudo-code for the algorithm.

Algorithm 3 FastH: Pseudocode

```
1: // step 1 (parallel algorithm)
2: Compute $G_1,\ldots,G_{d/t}$ in parallel using Lemma 3.
3:
4: // step 2 (sequential algorithm)
5: Compute $G_1(G_2\cdots(G_{d/t}X)\cdots)$
```

Time Complexity. For general t, step 1 takes $O(d/t\cdot dt^2) = O(d^2t)$ time while step 2 takes $O(d/t\cdot dtm) = O(d^2m)$, for a total time complexity of $O(d^2(t+m))$. If we choose $t=m$, we get the desired $O(d^2m)$ time complexity.

[3]Even if we exploit the trick from Equation (3.8) to multiply $P_i = H_iH_{i+1}$ in $O(d^2)$ time, we end up with $d/2$ product matrices P_i which we need to multiply together. If we do not exploit any structure of the P_i's this will take $O(d^3)$ for each multiplication leading to $O(d^4)$.

Sequential Matrix-Matrix Multiplications. In step 1, we compute each G_i in parallel. However, to compute the G_i's we perform $O(t)$ sequential matrix-matrix multiplications. In step 2, we compute d/t sequential $G_i \cdot X$ matrix-matrix multiplications. The total number of sequential matrix-matrix multiplications is thus $O(d/t+t)$. For $t=m$ FastH thus performs $O(d/m+m)$ sequential matrix-matrix multiplications instead of $O(d)$ sequential matrix-vector multiplications **without** increasing the asymptotic time complexity. This concludes our introduction to FastH. The introduction has deliberately disregarded several details, e.g., all gradient computations. We encourage the curious reader to read the full version for such details. Furthermore, to fully utilize the parallelism of GPUs, we implemented FastH in CUDA. The implementation with PyTorch bindings can be found at github.com/alexandermath/fasth.

3.4 Later Improvements

3.4.1 From $O(d/t+t)$ to $O(d/t+\log(t))$ Sequential Stages

The above results are the main content of our article [3] accepted for publication at NeurIPS. At the time of writing, we managed to decrease (improve) the number of sequential matrix-matrix operations. This section presents the improved algorithm.

FastH performs $O(d/t+t)$ sequential matrix-matrix multiplications. The "$+t$" sequential matrix-matrix multiplications is caused by Lemma 3 from Algorithm 3's Step 1. It is thus sufficient to improve Lemma 3 from $O(t)$ to $O(\log t)$ sequential matrix-matrix operations, this would make FastH perform the desired $O(d/t+\log t)$ matrix-matrix multiplications. We first sketch Lemma 3 and show why it uses $O(t)$ matrix-matrix operations. We then show how $O(t)$ can be reduced to $O(\log t)$.

Lemma 3 with $O(t)$. Given $v_1,\dots v_t \in \mathbb{R}^d$ compute $A_t, B_t \in \mathbb{R}^{d\times t}$ such that $I+A_tB_t^T = \prod_{i=1}^t H(v_i)$. We argue by induction. Base case $t=1$: let $A_1 = -2v_1/||v_1||^2$ and $B_1 = v_1$ it then holds that $I+A_1B_1^T = I-2v_1v_1^T/||v_1||^2 = H(v_1)$ by definition of $H(v)$. Induction step: assume that there exists $A_k, B_k \in \mathbb{R}^{d\times k}$ such that $I+A_kB_k^T = \prod_{i=1}^k H(v_k)$, we want to show the same is true for $k+1$. Let $a = -2v_{k+1}/||v_{k+1}||^2$ and $b = v_{k+1}$ so $(I+ab^T) = H(v_{k+1})$, we then construct A_{k+1} and B_{k+1} as follows:

$$\begin{aligned} I + \overbrace{\begin{pmatrix} | & | \\ A_k & (I+A_kB_k^T)a \\ | & | \end{pmatrix}}^{A_{k+1}} \cdot \overbrace{\begin{pmatrix} | & | \\ B_k & b \\ | & | \end{pmatrix}^T}^{B_{k+1}^T} &= I + A_k \cdot B_k^T + (I+A_kB_k^T)a \cdot b^T && (3.12) \\ &= I + A_kB_k^T + ab^T + A_kB_k^T ab^T && (3.13) \\ &= (I+A_kB_k^T)(I+ab^T) = \prod_{i=1}^{k+1} H(v_i). && (3.14) \end{aligned}$$

This means that B_t has columns $b_i = v_i$ which require no computation. It is thus sufficient to compute A_t. We iteratively compute A_{k+1} by adding a new column $(I+A_kB_k^T)a = \prod_{i=1}^k H(v_i)a$ to A_k. We can compute $(I+A_kB_k)a = a + A_k(B_ka)$ with $O(1)$ sequential matrix-vector multiplications in $O(dt)$ time. It thus takes $O(t)$ sequential matrix-vector multiplications to compute A_t in $O(dt^2)$ time.

Lemma 3 with $O(\log t)$. Assume t is a power of 2. Suppose we could merge $(I+A_kB_k^T) = \prod_{i=1}^k H(v_i)$ and $(I+\hat{A}_k\hat{B}_k^T) = \prod_{i=1}^k H(\hat{v}_i)$ into $(I+A_{2k}B_{2k}^T) = \prod_{i=1}^k H(v_i)\prod_{i=1}^k H(\hat{v}_i)$. We could then compute (A_t, B_t) by merging $(A_{t/2}, B_{t/2})$ with $(\hat{A}_{t/2}, \hat{B}_{t/2})$. We can recursively compute both $(A_{t/2}, B_{t/2})$ and $(\hat{A}_{t/2}, \hat{B}_{t/2})$ the same way. The recursion forms a balanced binary tree where each node represents a merge computation. Notably, all merge computations (nodes) at the same level of the tree do not depend on each other and can thus be computed in parallel. As a result, the number of sequential stages is no more than the height of the tree $O(\log t)$. In the following we (1) show how to merge, and (2) prove that all merge steps take no more than $O(dt^2)$ time.

We constructively prove how to merge by induction. Base case $k = 1$: we can merge $H(v_1)$ and $H(\hat{v}_1)$ with Lemma 3. Induction step: construct A_{2k} and B_{2k} as follows.

$$I + \overbrace{\begin{pmatrix} | & | \\ A_k & (I + A_k B_k^T)\hat{A}_k \\ | & | \end{pmatrix}}^{A_{2k}} \overbrace{\begin{pmatrix} | & | \\ B_k & \hat{B}_k \\ | & | \end{pmatrix}^T}^{B_{2k}^T} = I + A_k \cdot B_k^T + (I + A_k B_k^T)\hat{A}_k \cdot \hat{B}_k^T \tag{3.15}$$

$$= I + A_k B_k^T + \hat{A}_k \hat{B}_k^T + A_k B_k^T \hat{A}_k \hat{B}_k^T \tag{3.16}$$

$$= (I + A_k B_k^T)(I + \hat{A}_k \hat{B}_k^T) = \prod_{i=1}^{k} H(v_i) \prod_{i=1}^{k} H(\hat{v}_i). \tag{3.17}$$

So B_{2k} has columns $v_1, \ldots, v_k, \hat{v}_1, \ldots, \hat{v}_k$ which require no computation. We compute A_{2k} by adding the k columns $(I + A_k B_k)\hat{A}_k$ to A_k, which takes $O(dk^2)$ time and $O(1)$ sequential matrix-matrix multiplications.

It is now left to show that all merge computations take no more than $O(dt^2)$ time. Consider all merge computations $((A_k, B_k), (\hat{A}_k, \hat{B}_k))$ of size $k = 2^i$. There are $t/(2 \cdot k) = t/2^{i+1}$ such merge computations, each take $O(dk^2) = O(d2^{i \cdot 2})$ time, so for size $k = 2^i$ all merges take $O(t/2^{i+1} \cdot d2^{i \cdot 2}) = O(dt2^{i-1})$ time. The time to merge all sizes $k = 2^0, 2^1, \ldots, 2^{\log(t)-1}$ is thus:

$$O\left(\sum_{i=0}^{\log_2(t)-1} dt2^{i-1}\right) = O(dt^2) \tag{3.18}$$

This allows us run FastH in $O(d^2 t)$ time with $O(d/t + \log(t))$ sequential matrix-matrix multiplications. We note that the above "merge generalization" was introduced by [11], however, they were unable to use it in their QR algorithm because the Householder matrices are computed sequentially (step 1 on page 12 is computed by Algorithm 4.1). The reason it works in our setting, is that we need not compute the Householder vectors sequentially, we already know all of them because they are our weights.

3.4.2 $O(1)$ Sequential Stages are Sufficient for Expressivity [4]

The sequential algorithm from [37, 52] computes $H(v_1) \cdots H(v_d) \cdot X$ in $O(d^2 m)$ time with $O(d)$ sequential matrix-vector multiplications. Informally, the main difficulty lies in handling the d Householder matrices. FastH groups these d matrices and rely on the WY decomposition to use only $O(d/m + \log(m))$ sequential matrix-matrix multiplications without increasing the time complexity. This is faster than computing the SVD, however, there is still room for improvement. In particular, one may wonder whether it is possible to create an algorithm with just $O(1)$ sequential stages.

This lead us to consider a different approach. Instead of using d Householder matrices, we tried to find a way to compute a Householder vector $v = f(x)$ such that one Householder matrix $H(f(v))$ can represent any product of Householder matrices. In other words, can we find a function $f : \mathbb{R}^d \to \mathbb{R}^d$ such that $H(f(x))x = \prod_{i=1}^{d} H(v_i)x = Ux$ for $x \in \mathbb{R}^d$? This would allow us to perform any orthogonal transformation with just one sequential multiplication (excluding those used to compute f).

Since neural networks are universal function approximators, it seemed likely that a sufficiently powerful neural network f would be capable of finding a Householder transformation specific for x such that $H(f(x))x = Ux$. However, it turns out that we do not even need a neural network, if we choose $f(x) = (I - U)x$ it holds that $H(f(x))x = Ux$ for any $x \in \mathbb{R}^d$. The proof is surprisingly simple if one accepts the following fact: for $x, y \in \mathbb{R}^d$ with $||x||_2 = ||y||_2$ then $H(x - y)x = y$. Using this fact, we see that for $W = I - U$ it follows that $H(Wx)x = H(x - Ux)x = Ux$ for all $x \in \mathbb{R}^d$ because orthogonal matrices U preserve norm $||Ux|| = ||x||$. Notably, the transformation $g(x) = H(Wx)x$ takes $O(d^2 m)$ time to compute for m examples while using only $O(1)$ sequential multiplications.

We intended to use $g(x)$ in invertible neural networks [15, 16, 31], however, it is not clear whether $g(x)$ is invertible.

Problem. Given $y = g(x) = H(Wx)x$ and invertible W is x unique and can we compute x?

At first glance this problem may seem trivial. The main difficulty arise from the way y depends on x in two ways, which makes it hard to solve for x. Nonetheless, we managed to find conditions on W for which $y = H(Wx)x$ has an unique solution x. [4] Unfortunately, the restrictions on W are rather strict: we require that $W = W^T$ and put an awkward inequality on the eigenvalues $2/3 \cdot \lambda_{\min}(W) > \lambda_{\max}(W)$.

Theorem 5. *Let $g(x) = H(Wx)x$ with $g(0) := 0$, then g is invertible on $\mathbb{R}^d$ with $d \geq 2$ if $W = W^T$ and W has eigenvalues which satisfy $3/2 \cdot \lambda_{\min}(W) > \lambda_{\max}(W)$.*

Proof Sketch. We want to show that g is invertible, i.e., for invertible $W = W^T \in \mathbb{R}^d$ we want to show that for every $y \in \mathbb{R}^d$ there exists an unique x such that $y = g(x) = H(Wx)x$. Our first insight is to show that g is invertible on $\mathbb{R}^d$ if f is "approximately invertible" on the unit sphere $S^{d-1} = \{x \in \mathbb{R}^d \mid ||x||_2^2 = 1\}$. The notion of "approximately invertible" we use is that $g(x)$ has a non-zero Jacobian determinant

$$J := \det\left(\frac{\partial g(x)}{\partial x}\right) \neq 0. \tag{3.19}$$

It then suffices to show that g has a non-zero Jacobian determinant on the unit sphere S^{d-1}. We then prove the following formula for the Jacobian determinant of $g(x)$.

$$J = H(Wx)A - 2\frac{Wxx^TW}{||Wx||^2} \quad \text{where} \quad A = I - 2\frac{x^TW^Tx}{||Wx||^2}W. \tag{3.20}$$

This formula has a lot of structure. Besides containing a Householder multiplication, we can write it as a rank-1 update, i.e., the formula has the form $M + uv^T$ for a matrix M and vectors u, v. This allows us to use the *matrix determinant lemma* $\det(M + uv^T) = \det(M)(1 + v^TM^{-1}u)$ to prove that

$$\det(J) = -\det(A)\left(1 + 2 \cdot \underbrace{\frac{v^TA^{-1}u}{||u||^2}}_{\Gamma}\right) \quad \text{where} \quad v^T = x^TW, \quad u = Wx \quad \text{and} \quad A = I - 2\frac{x^TW^Tx}{||Wx||^2}W.$$

Notice that $\det(J) \neq 0$ if both $\det(A) \neq 0$ and $1 + 2 \cdot \Gamma \neq 0$. We can bound Γ by an interval that depends on the eigenvalues of A^{-1}, because Γ cannot attain a larger (or smaller) value than when $u = v$ is the eigenvector of A^{-1} with the largest (or smallest) eigenvalue. The eigenvalue bound on Γ allows us to prove $1 + 2 \cdot \Gamma \neq 0$ if $W^T = W$ and $2/3 \cdot \lambda_{\min}(W) > \lambda_{\max}(W)$. This condition also ensures that A^{-1} exists which means that $\det(A) \neq 0$.

[4]The proof was based on helpful discussion with Mikael Møller Høgsgaard, Chris Schwiegelshohn, Allan Grønlund, Kasper Green Larsen and Ben Grossmann. The discussion with Ben went through the online forum www.math.stackexchange.com and can be found online.

3.5 Related Work

Orthogonal Matrices in Neural Networks. Training of Neural Networks can be halted by exploding or vanishing gradients. To mitigate this issue, [7] suggested using unitary matrices $\overline{U}^T U = I$ where $\overline{U}$ is the complex conjugate of $U \in \mathbb{C}^{d\times d}$. Unitary matrices preserve norms $||Ux||_2 = ||x||_2$ for any $x \in \mathbb{C}^d$, which ensures that the norm of gradients do not explode or vanish during training. They chose to represent an unitary matrix U with $O(d)$ parameters so UX takes $O(md\log d)$ time to compute for $X \in \mathbb{R}^{d\times m}$. Their representation cannot represent all unitary matrices.

The work by [37] proved, loosely speaking, that for the type of neural network used in [7], one can represent a neural network with "small" unitary matrices by another neural network with "slightly larger" orthogonal matrices. This led [37] to represent an orthogonal matrix $Q \in \mathbb{R}^{d\times d}$ with $O(d)$ Householder matrices. Their representation can represent any orthogonal matrix using $O(d^2)$ parameters, and it takes $O(d^2m)$ time to compute $Q \cdot X$. To mitigate the slow $O(d)$ sequential stages, their experiments less than d Householder matrices, e.g., they used 16 Householder matrices to represent a 128×128 matrix.

The work by [25] represents an orthogonal matrix using a scaled Cayley transform $(I+W)^{-1}(I-W)D$ for a skew symmetric matrix $W^T = -W$ and a diagonal matrix D. Their representation takes $O(d^2)$ parameters and can express any orthogonal matrix, however, in general the matrix inversion takes $O(d^3)$ time to compute. Notably, they circumvent $O(d)$ sequential stages.

The work by [34] represents orthogonal matrices by using the matrix exponential e^W of a skew symmetric matrix $W^T = -W$. Their representation uses $O(d^2)$ parameters and can express any orthogonal matrix with $\det(Q) = +1$ but not those with $\det(Q) = -1$. They approximate e^W to machine precision using a $O(d^3)$ time algorithm, circumventing $O(d)$ sequential stages.

The work by [52] extended the use of orthogonal matrices from [37] to represent a matrix W in its SVD $W = U\Sigma V^T$. While W is not norm-preserving, the SVD allows them to bound how much W can skew norms by fixing the singular values $\Sigma_{ii} \in [1 \pm \varepsilon]$. Their representation also relies on Householder matrices. To circumvent the many sequential operations, they suggest using a $O(d^3)$ algorithm to compute the product of the d Householder matrices.

Our work [3] represents Q by d Householder matrices, which use $O(d^2)$ parameters and can represent any orthogonal matrix. FastH computes $Q \cdot X$ in $O(d^2m)$ time with $O(d/m + \log(m))$ sequential stages. For $m > 1$ this circumvents both the $O(d)$ sequential stages and the $O(d^3)$ time complexity. To do this FastH utilizes techniques similar to those found in "parallel" QR algorithms [11, 46]. The main difference between FastH and "parallel" QR algorithms, is that FastH is given Householder matrices and asked to compute $H_1 \cdot H_d \cdot X$, while QR algorithms are given A and asked to compute $QR = A$ where $Q = H_1 \cdots H_d$. This difference is important, because it allows us to circumvent sequentially computing H_{i+1} dependent on $H_i \cdots H_1 \cdot A$

Simultaneously to our work, [35] tried to mitigate the $O(d)$ sequential stages of the Householder approach by using the CWY decomposition $Q = I - US^{-1}U^T$ for triangular S. Their method relies on a matrix inversion, which in general takes $O(d^3)$ time. That said, they only need to invert a triangular matrix, for which they could use the backward substitution algorithm to compute $S^{-1}X$ in $O(d^2m)$ time but $O(d)$ sequential stages.

Chapter 4

Articles

4.1 [1] Optimal Minimal Margin Maximization with Boosting

Optimal Minimal Margin Maximization with Boosting

Allan Grønlund[*][†] Kasper Green Larsen[‡] Alexander Mathiasen[§]

Abstract

Boosting algorithms produce a classifier by iteratively combining base hypotheses. It has been observed experimentally that the generalization error keeps improving even after achieving zero training error. One popular explanation attributes this to improvements in margins. A common goal in a long line of research, is to maximize the smallest margin using as few base hypotheses as possible, culminating with the AdaBoostV algorithm by [Rätsch and Warmuth, 2005]. The AdaBoostV algorithm was later conjectured to yield an optimal trade-off between number of hypotheses trained and the minimal margin over all training points [Nie et al., 2013]. Our main contribution is a new algorithm refuting this conjecture. Furthermore, we prove a lower bound which implies that our new algorithm is optimal.

1 Introduction

Boosting is one of the most famous and succesful ideas in learning. Boosting algorithms are meta algorithms that produce highly accurate classifiers by combining already existing less accurate classifiers. Probably the most famous boosting algorithm is AdaBoost by Freund and Schapire [Freund and Schapire, 1995], who won the 2003 Gödel Prize for their work.

AdaBoost was designed for binary classification and works by combining base hypotheses learned by a given base learning algorithm into a weighted sum that represents the final classifier. This weighed set of base hypotheses is constructed iteratively in rounds, each round constructing a new base hypothesis that focuses on the training data misclassified by the previous base hypotheses constructed. More precisely, AdaBoost takes training data $D = \{(x_i, y_i) \mid x_i \in \mathbb{R}^d, y_i \in \{-1,+1\}\}_{i=1}^n$ and constructs a linear combination classifier $\text{sign}(\sum_{t=1}^T \alpha_t h_t(x))$, where h_t is the base hypothesis learned in the t'th iteration and α_t is the corresponding weight.

It has been proven that AdaBoost decreases the training error exponentially fast if each base hypothesis is slightly better than random guessing on the weighed data set it is trained on [Freund et al., 1999]. Concretely, if ϵ_t is the error of h_t on the weighed data set used to learn h_t then the linear combination has training error at most $\exp(-2\sum_{t=1}^T (1/2-\epsilon_t)^2)$. If each ϵ_t is at most a half minus a fixed constant, then the training error is less than $1/n$ after $O(\lg n)$ rounds which means the all training points are classified correctly. Quite surprisingly, experiments show that continuing the AdaBoost algorithm even after the training data is perfectly classified, making the model more and more complex, continues to improve generalization [Schapire et al., 1998]. The most prominent approach to explaining this generalization phenomenon considers *margins* [Schapire et al., 1998].

[*]All authors contributed evenly.

[†]Aarhus University. Email: jallan@cs.au.dk.

[‡]Aarhus University. Email: larsen@cs.au.dk. Supported by a Villum Young Investigator Grant and an AUFF Starting Grant.

[§]Aarhus University. Email: alexander.mathiasen@gmail.com. Supported by an AUFF Starting Grant.

The margin of a point x_i is

$$\text{margin}(x_i) = \frac{y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)}{\sum_{t=1}^{T} |\alpha_t|}.$$

For binary classification, if each $h_t(x) \in [-1, +1]$, then the margin of a point is a number between -1 and +1. Notice that a point has positive margin if it is classified correctly and negative margin if it is classified incorrectly. It has been observed experimentally that the margins of the training points usually increase when training, even after perfectly classifying the training data. This has inspired several bounds on generalization error that depend on the distribution of margins [Schapire et al., 1998, Breiman, 1999, Koltchinskii et al., 2001, Wang et al., 2008, Gao and Zhou, 2013]. The conceptually simplest of these bounds depend only on the minimal margin, which is the margin of the point x_i with minimal margin. The point x_i with minimal margin can be interpreted as the point the classifier struggles the most with. This has inspired a series of algorithms with guarantees on the minimal margin [Breiman, 1999, Grove and Schuurmans, 1998, Bennett et al., 2000, Rätsch and Warmuth, 2002, Rätsch and Warmuth, 2005].

These algorithms have the following goal: Let $\mathcal{H}$ be the (possibly infinite) set of all base hypotheses that may be returned by the base learning algorithm. Suppose the best possible minimal margin on some training data for any linear combination of $h \in \mathcal{H}$ is ρ^*, i.e.

$$\rho^* = \max_{\alpha \neq 0} \left(\min_i \frac{y_i \sum_{h \in \mathcal{H}} \alpha_h h(x_i)}{\sum_{h \in \mathcal{H}} |\alpha_h|} \right).$$

Given some precision parameter v, the goal is to construct a linear combination with minimal margin at least $\rho = \rho^* - v$ using as few hypotheses as possible. In this case we say that the linear combination has a gap of v. The current state of the art is AdaBoostV [Rätsch and Warmuth, 2005]. It guarantees a gap of v using $O(\lg(n)/v^2)$ hypotheses. It was later conjectured that there exists data sets D and a corresponding set of base hypotheses $\mathcal{H}$, such that any linear combination of base hypotheses from $\mathcal{H}$ must use at least $\Omega(\lg(n)/v^2)$ hypotheses to achieve a gap of v for any $\sqrt{\lg n/n} \leq v \leq a_1$ for some constant $a_1 > 0$. This would imply optimality of AdaBoostV. This conjecture was published as an open problem in the Journal of Machine Learning Research [Nie et al., 2013].

Our main contribution is a refutal of this conjecture. We refute the conjecture by introducing a new algorithm called SparsiBoost, which guarantees a gap of v with just $T = O(\lg(nv^2)/v^2)$ hypotheses. When $v \leq n^{o(1)}/\sqrt{n}$, SparsiBoost has $T = O(\lg(n^{o(1)})/v^2) = o(\lg(n)/v^2)$, which is asymptotically better than AdaBoostV's $T = O(\lg(n)/v^2)$ guarantee. Moreover, it also refutes the conjectured lower bound. Our algorithm involves a surprising excursion to the field of combinatorial discrepancy minimization. We also show that our algorithm is the best possible. That is, there exists data sets D and corresponding set of base hypotheses $\mathcal{H}$, such that any linear combination of base hypotheses from $\mathcal{H}$ with a gap of v, must use at least $T = \Omega(\lg(nv^2)/v^2)$ hypotheses.

This work thus provides the final answer to over a decade's research into understanding the trade-off between minimal margin and the number of hypotheses: Given a gap v, the optimal number of hypotheses is $T = \Theta(\lg(nv^2)/v^2)$ for any $\sqrt{1/n} \leq v \leq a_1$ where $a_1 > 0$ is a constant. Notice that smaller values for v are irrelevant since it is always possible to achieve a gap of zero using $n+1$ base hypotheses. This follows from Carathéodory's Theorem.

1.1 Previous Work on Minimal Margin

Upper Bounds. [Breiman, 1999] introduced Arc-GV, which was the first algorithm that guaranteed to find a finite number of hypotheses $T < \infty$ with gap zero ($v = 0$). As pointed out by [Rätsch and Warmuth, 2005], one can think of Arc-GV as a subtle variant of AdaBoost where the weights α_t of the hypotheses are slightly changed. If AdaBoost has hypothesis weight α_t, then Arc-GV chooses the hypothesis weight $\alpha'_t = \alpha_t + x$ for some x that depends on the minimal margin of $h_1, \ldots, h_t$. A few years later, [Grove and Schuurmans, 1998] and [Bennett et al., 2000] introduced DualLPBoost and LPBoost which both have similar guarantees.

[Rätsch and Warmuth, 2002] introduced AdaBoost$_\rho$, which was the first algorithm to give a guarantee on the gap achieved in terms of the number of hypotheses used. Their algorithm takes a parameter $\rho \le \rho^*$ that serves as the target margin one would like to achieve. It then guarantees a minimal margin of $\rho - \mu$ using $T = O(\lg(n)/\mu^2)$ hypotheses. One would thus like to choose $\rho = \rho^*$. If ρ^* is unknown, it can be found up to an additive approximation of v by binary searching using AdaBoost$_\rho$. This requires an additional $O(\lg 1/v)$ calls to AdaBoost$_\rho$, resulting in $O(\lg(n)/v^2)\lg(1/v)$ iterations of training a base hypothesis to find the desired linear combination of $T = O(\lg(n)/v^2)$ base hypotheses. Similar to Arc-GV, AdaBoost$_\rho$ differs from AdaBoost only in choosing the weights α_t. Instead of having the additional term depend on the minimal margin of $h_1, \ldots, h_t$, it depends only on the estimation parameter ρ.

A few years later, [Rätsch and Warmuth, 2005] introduced AdaBoostV. It is a clever extension of AdaBoost$_\rho$ that uses an adaptive estimate of ρ^* to remove the need to binary search for it. It achieves a gap of v using $T = O(\lg(n)/v^2)$ base hypotheses and no extra iterations of training.

Lower Bounds. [Klein and Young, 1999] showed a lower bound for a seemingly unrelated game theoretic problem. It was later pointed out by [Nie et al., 2013] that their result implies the following lower bound for boosting: there exists a data set of n points and a corresponding set of base hypotheses $\mathcal{H}$, such that any linear combination of $T \in [\lg n; \sqrt{n}]$ base hypotheses must have a gap of $v = \Omega(\sqrt{\lg(n)/T})$. Rewriting in terms of T we get $T = \Omega\left(\lg(n)/v^2\right)$ for $\sqrt{\lg(n)}/n^{1/4} \le v \le a_1$ for some constant $a_1 > 0$.

[Nie et al., 2013] conjectured that Klein and Young's lower bound of $v = \Omega(\sqrt{\lg(n)/T})$ holds for all $T \le a_1 \cdot n$ for some constant $a_1 > 0$. Rewriting in terms of T, they conjecture that $T = \Omega(\lg(n)/v^2)$ holds for $\sqrt{a_2/n} \le v \le a_3$ where $a_2, a_3 > 0$ are some constants.

1.2 Our Results On Minimal Margin

Our main result is a novel algorithm, called SparsiBoost, which refutes the conjectured lower bound in [Nie et al., 2013]. Concretely, SparsiBoost guarantees a gap of v with just $T = O(\lg(nv^2)/v^2)$ hypotheses. At a first glance it might seem SparsiBoost violates the lower bound of Klein and Young. Rewriting in terms of v, our upper bound becomes $v = O(\sqrt{\lg(n/T)/T})$. When $T \le \sqrt{n}$ (the range of parameters where their lower bound applies), this becomes $v = O(\sqrt{\lg(n)/T})$ which does not violate Klein and Young's lower bound. Moreover, our upper bound explains why both [Klein and Young, 1999] and [Nie et al., 2013] were not able to generalize the lower bound to all $T = O(n)$: When $T = n^{1-o(1)}$, our algorithm achieves a gap of $v = O(\sqrt{\lg(n^{o(1)})/T}) = o(\sqrt{\lg(n)/T})$.

The high level idea of SparsiBoost is as follows: Given a desired gap v, we use AdaBoostV to find $m = O(\lg(n)/v^2)$ hypotheses $h_1, \ldots, h_m$ and weights $w_1, \ldots, w_m$ such that $\sum_i w_i h_i$ achieves a gap of $v/2$. We then carefully "sparsify" the vector $w = (w_1, \ldots, w_m)$ to obtain another vector w' that has at most $T = O(\lg(nv^2)/v^2)$ non-zeroes. Our sparsification is done such that the margin of every single data point changes by at most $v/2$ when replacing $\sum_i w_i h_i$ by $\sum_i w'_i h_i$. In particular, this implies that the minimum margin, and hence gap, changes by at most $v/2$. We can now safely ignore all hypotheses h_i where $w'_i = 0$ and we have obtained the claimed gap of at most $v/2 + v/2 = v$ using $T = O(\lg(nv^2)/v^2)$ hypotheses.

Our algorithm for sparsifying w gives a general method for sparsifying a vector while approximately preserving a matrix-vector product. We believe this result may be of independent interest and describe it in more detail here: The algorithm is given as input a matrix $U \in [-1,+1]^{n\times m}$ and a vector $w \in \mathbb{R}^m$ where $\|w\|_1 = 1$. It then finds a vector w' such that $\|Uw - Uw'\|_\infty = O(\lg(n/T)/T)$, $\|w'\|_0 \le T$ and $\|w'\|_1 = 1$. Here $\|x\|_1 = \sum_i |x_i|$, $\|x\|_\infty = \max_i |x_i|$ and $\|x\|_0$ denotes the number of non-zero entries of x. When we use this result in SparsiBoost, we will define the matrix U as the "margin matrix" that has $u_{ij} = y_i h_j(x_i)$. Then $(Uw)_i = \text{margin}(x_i)$ and the guarantee $\|Uw - Uw'\|_\infty = O(\lg(n/T)/T)$ will ensure that the margin of every single point changes by at most $O(\lg(n/T)/T)$ if we replace the weights w by w'. Our algorithm for finding w' is based on a novel connection to the celebrated but seemingly unrelated "six standard devitations suffice" result by [Spencer, 1985] from the field of combinatorial discrepancy minimization.

When used in SparsiBoost, the matrix U is defined from the output of AdaBoostV, but the vector sparsification algorithm could just as well be applied to the hypotheses output by any boosting algorithm. Thus our results give a general method for sparsifying a boosting classifier while approximately preserving the margins of all points.

We complement our new upper bound with a matching lower bound. More concretely, we prove that there exists data sets D of n points and a corresponding set of base hypotheses $\mathcal{H}$, such that any linear combination of T base hypotheses must have a gap of at least $v = \Omega(\sqrt{\lg(n/T)/T})$ for any $\lg n \le T \le a_1 n$ where $a_1 > 0$ is a constant. Rewriting in terms of T, one must use $T = \Omega(\lg(nv^2)/v^2)$ hypotheses from $\mathcal{H}$ to achieve a gap of v. This holds for any v satisfying $\sqrt{a_2/n} < v \le a_3$ for constants $a_2, a_3 > 0$. Interestingly, our lower bound proof also uses the discrepancy minimization upper bound by [Spencer, 1985] in a highly non-trivial way. Our lower bound also shows that our vector sparsification algorithm is optimal for any $T \le n/C$ for some universal constant $C > 0$.

1.3 Doubts on Margin Theory and other Margin Bounds

The first margin bound on boosted classifiers was introduced by [Schapire et al., 1998]. Shortly after, [Breiman, 1999] introduced a sharper minimal margin bound alongside Arc-GV. Experimentally Breimann found that Arc-GV produced better margins than AdaBoost on 98 percent of the training data, however, AdaBoost still obtained a better test error. This seemed to contradict margin theory: according to margin theory, better margins should imply better generalization. This caused Breimann to doubt margin theory. It was later discovered by [Reyzin and Schapire, 2006] that the comparison was unfair due to a difference in the complexity of the base hypotheses used by AdaBoost and Arc-GV. [Reyzin and Schapire, 2006] performed a variant of Breimann's experiments with decision stumps to control the hypotheses complexity. They found that even though Arc-GV produced a better minimal margin, AdaBoost produced a larger margin on almost all other points [Reyzin and Schapire, 2006] and that AdaBoost generalized better.

A few years later, [Wang et al., 2008] introduced a sharper margin bound than Breimann's minimal margin bound. The generalization bound depends on a term called the *Equilibrium Margin*, which itself depends on the margin distribution in a highly non-trivial way. This was followed by the k-margin bound by [Gao and Zhou, 2013] that provide generalization bounds based on the k'th smallest margin for any k. The bound gets weaker with increasing k, but stronger with increasing margin. In essence, this means that we get stronger generalization bounds if the margins are large for small values of k.

Recall from the discussion in Section 1.2 that our sparsification algorithm preserves all margins to within $O(\sqrt{\lg(n/T)/T})$ additive error. We combined this result with AdaBoostV to get our algorithm SparsiBoost which obtained an optimal trade-off between minimal margin and number of hypotheses. While minimal margin might be insufficient for predicting generalization performance, our sparsification algorithm actually preserves the full distribution of margins. Thus according to margin theory, the sparsified classifier should approximately preserve the generalization performance of the full unsparsified classifier. To demonstrate this experimentally, we sparsified a classifier trained with LightGBM [Ke et al., 2017], a highly efficient open-source implementation of Gradient Boosting [Mason et al., 2000, Friedman, 2001]. We compared the margin distribution and the test error of the sparsified classifier against a LightGBM classifier trained directly to have the same number of hypotheses. Our results (see Section 4) show that the sparsified classifier has a better margin distribution and indeed generalize better than the standard LightGBM classifier.

2 SparsiBoost

In this section we introduce SparsiBoost. The algorithm takes the following inputs: training data D, a target number of hypotheses T and a base learning algorithm A that returns hypotheses from a class $\mathcal{H}$ of possible base hypotheses. SparsiBoost initially trains $c \cdot T$ hypotheses for some appropriate c, by running AdaBoostV with the base learning algorithm A. It then removes the extra $c \cdot T - T$ hypotheses while attempting to preserve the margins on all training examples.

In more detail, let $h_1, ..., h_{cT} \in \mathcal{H}$ be the hypotheses returned by AdaBoostV with weights $w_1, ..., w_{cT}$. Construct a margin matrix U that contains the margin of every hypothesis h_j on every point x_i such that $u_{ij} = y_i h_j(x_i)$. Let w be the vector of hypothesis weights, meaning that the j'th coordinate of w has the weight w_j of hypothesis h_j. Normalize $w = w/\|w\|_1$ such that $\|w\|_1 = 1$. The product Uw is then a vector that contains the margins of the final linear combination on all points: $(Uw)_i = y_i \sum_{j=1}^{cT} w_j h_j(x_i) = \text{margin}(x_i)$. Removing hypotheses while preserving the margins can be formulated as sparsifying w to w' while minimizing $\|Uw - Uw'\|_\infty$ subject to $\|w'\|_0 \le T$ and $\|w'\|_1 = 1$.

See Algorithm 1 for pseudocode.

Algorithm 1 SparsiBoost

Input: Training data $D = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in X$ for some input space X and $y_i \in \{-1, +1\}$. Target number of hypotheses T and base learning algorithm A.

Output: Hypotheses $h_1, \ldots, h_k$ and weights $w_1, \ldots, w_k$ with $k \le T$, such that $\sum_i w_i h_i$ has gap $O(\sqrt{\lg(2 + n/T)/T})$ on D.

```
1. Run AdaBoostV with base learning algorithm $A$ on training data $D$ to get $cT$ hypotheses $h_1, \ldots, h_{cT}$ and weights $w_1, \ldots, w_{cT}$ for the integer $c = \lceil \lg(n)/\lg(2 + n/T) \rceil$.
2. Construct margin matrix $U \in [-1, +1]^{n \times cT}$ where $u_{ij} = y_i h_j(x_i)$.
3. Form the vector $w$ with $i$'th coordinate $w_i$ and normalize $w \leftarrow w/\|w\|_1$ so $\|w\|_1 = 1$.
4. Find $w'$ such that $\|w'\|_0 \le T$, $\|w'\|_1 = 1$ and $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2 + n/T)/T})$.
5. Let $\pi(j)$ denote the index of the $j$'th non-zero entry of $w'$.
6. Return hypotheses $h_{\pi(1)}, \ldots, h_{\pi(\|w'\|_0)}$ with weights $w'_{\pi(1)}, \ldots, w'_{\pi(\|w'\|_0)}$.
```

We still haven't described how to find w' with the guarantees shown in Algorithm 1 step 4., i.e. a w'

with $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/T)/T})$. It is not even clear that such w' exists, much less so that it can be found efficiently. Before we dive into the details of how to find w', we briefly demonstrate that indeed such a w' would be sufficient to establish our main theorem:

Theorem 2.1. *SparsiBoost is guaranteed to find a linear combination w' of at most T base hypotheses with gap $v = O(\sqrt{\lg(2+n/T)/T})$.*

Proof. We assume throughout the proof that a w' with the guarantees claimed in Algorithm 1 can be found. Suppose we run AdaBoostV to get cT base hypotheses $h_1, \ldots, h_{cT}$ with weights $w_1, \ldots, w_{cT}$. Let ρ_{cT} be the minimal margin of the linear combination $\sum_i w_i h_i$ on the training data D, and let ρ^* be the optimal minimal margin over all linear combinations of base hypotheses from $\mathcal{H}$. As proved in [Rätsch and Warmuth, 2005], AdaBoostV guarantees that the gap is bounded by $\rho^* - \rho_{cT} = O(\sqrt{\lg(n)/(cT)})$. Normalize $w = w/\|w\|_1$ and let U be the margin matrix $u_{ij} = y_i h_j(x_i)$ as in Algorithm 1. Then $\rho_{cT} = \min_i (Uw)_i$. From our assumption, we can efficiently find w' such that $\|w'\|_1 = 1$, $\|w'\|_0 \le T$ and $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/T)/T})$. Consider the hypotheses that correspond to the non-zero entries of w'. There are at most T. Let ρ_T be their minimal margin when using the corresponding weights from w'. Since w' has unit ℓ_1-norm, it follows that $\rho_T = \min_i (Uw')_i$ and thus $|\rho_T - \rho_{cT}| \le \max_i |(Uw)_i - (Uw')_i|$, i.e. $|\rho_{cT} - \rho_T| \le \|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/T)/T})$. We therefore have:

$$\rho^* - \rho_T = (\rho^* - \rho_{cT}) + (\rho_{cT} - \rho_T) \le$$
$$O(\sqrt{\lg(n)/(cT)}) + O(\sqrt{\lg(2+n/T)/T}).$$

By choosing $c = \lg(n)/\lg(2+n/T)$ (as in Algorithm 1) we get that $\rho^* - \rho_T = O(\sqrt{\lg(2+n/T)/T})$. □

The core difficulty in our algorithm is thus finding an appropriate w' (step 4 in Algorithm 1) and this is the focus of the remainder of this section. Our algorithm for finding w' gives a general method for sparsifying a vector w while approximately preserving every coordinate of the matrix-vector product Uw for some input matrix U. The guarantees we give are stated in the following theorem:

Theorem 2.2. *(Sparsification Theorem) For all matrices $U \in [-1,+1]^{n\times m}$, all $w \in \mathbb{R}^m$ with $\|w\|_1 = 1$ and all $T \le m$, there exists a vector w' where $\|w'\|_1 = 1$ and $\|w'\|_0 \le T$, such that $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/T)/T})$.*

Theorem 2.2 is exactly what was needed in the proof of Theorem 2.1. Our proof of Theorem 2.2 will be constructive in that it gives an algorithm for finding w'. To keep the proof simple, we will argue about running time at the end of the section.

The first idea in our algorithm and proof of Theorem 2.2, is to reduce the problem to a simpler task, where instead of reducing the number of hypotheses directly to T, we only halve the number of hypotheses:

Lemma 2.1. *For all matrices $U \in [-1,+1]^{n\times m}$ and $w \in \mathbb{R}^m$ with $\|w\|_1 = 1$, there exists w' where $\|w'\|_0 \le \|w\|_0/2$ and $\|w'\|_1 = 1$, such that $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})$.*

To prove Theorem 2.2 from Lemma 2.1, we can repeatedly apply Lemma 2.1 until we are left with a vector with at most T non-zeroes. Since the loss $O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})$ has a $\sqrt{1/\|w\|_0}$ factor, we can use the triangle inequality to conclude that the total loss is a geometric sum that is asymptotically dominated by the very last invocation of the halving procedure. Since the last invocation has $\|w\|_0 > T$ (otherwise we would have stopped earlier), we get a total loss of $O(\sqrt{\lg(2+n/T)/T})$ as desired. The formal proof can be found in Section 2.1.

The key idea in implementing the halving procedure Lemma 2.1 is as follows: Let $\pi(j)$ denote the index of the j'th non-zero in w and let $\pi^{-1}(j)$ denote the index i such that w_i is the j'th non-zero entry of w. First we construct a matrix A where the j'th column of A is equal to the $\pi(j)$'th column of U scaled by the weight $w_{\pi(j)}$. The sum of the entries in the i'th row of A is then equal to the i'th entry of Uw (since $\sum_j a_{ij} = \sum_j w_{\pi^{-1}(j)} u_{i\pi^{-1}(j)} = \sum_{j:w_j \neq 0} w_j u_{ij} = \sum_j w_j u_{ij} = (Uw)_i$). Minimizing $\|Uw - Uw'\|_\infty$ can then be done by finding a subset of columns of A that approximately preserves the row sums. This is formally expressed in Lemma 2.2 below. For ease of notation we define $a \pm [x]$ to be the interval $[a-x, a+x]$ and $\pm[x]$ to be the interval $[-x, x]$.

Lemma 2.2. *For all matrices $A \in [-1,1]^{n\times T}$ there exists a submatrix $\hat{A} \in [-1,1]^{n\times k}$ consisting of $k \leq T/2$ distinct columns from A, such that for all i, it holds that $\sum_{j=1}^{k} \hat{a}_{ij} \in \frac{1}{2}\sum_{j=1}^{T} a_{ij} \pm \left[O(\sqrt{T\lg(2+n/T)})\right]$.*

Intuitively we can now use Lemma 2.2 to select a subset S of at most $T/2$ columns in A. We can then replace the vector w with w' such that $w'_i = 2w_i$ if $i = \pi^{-1}(j)$ for some $j \in S$ and $w'_i = 0$ otherwise. In this way, the i'th coordinate $(Uw')_i$ equals the i'th row sum in $\hat{A}$, scaled by a factor two. By Lemma 2.2, this in turn approximates the i'th row sum in A (and thus $(Uw)_i$) up to additively $O(\sqrt{T\lg(2+n/T)})$.

Unfortunately our procedure is not quite that straightforward since $O(\sqrt{T\lg(2+n/T)})$ is way too large compared to $O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0}) = O(\sqrt{\lg(2+n/T)/T})$. Fortunately Lemma 2.2 only needs the coordinates of A to be in $[-1,1]$. We can thus scale A by $1/\max_i |w_i|$ and still satisfy the constraints. This in turn means that the loss is scaled down by a factor $\max_i |w_i|$. However, $\max_i |w_i|$ may be as large as 1 for highly unbalanced vectors. Therefore, we start by copying the largest $T/3$ entries of w to w' and invoke Lemma 2.2 twice on the remaining $2T/3$ entries. This ensures that the $O(\sqrt{T\lg(2+n/T)})$ loss in Lemma 2.2 gets scaled by a factor at most $3/T$ (since $\|w\|_1 = 1$, all remaining coordinates are less than or equal to $3/T$), while leaving us with at most $T/3 + (2T/3)/4 = T/3 + T/6 = T/2$ non-zero entries as required. Since we normalize by at most $3/T$, the error becomes $\|Uw - Uw'\|_\infty = O(\sqrt{T\lg(2+n/T)}/T) = O(\sqrt{\lg(2+n/T)/T})$ as desired. As a last technical detail, we also need to ensure that w' satisfies $\|w'\|_1 = 1$. We do this by adding an extra row to A such that $a_{(n+1)j} = w_j$. In this way, preserving the last row sum also (roughly) preserves the ℓ_1-norm of w and we can safely normalize w' as $w' \leftarrow w'/\|w\|_1$. The formal proof is given in Section 2.2.

The final step of our algorithm is thus to select a subset of at most half of the columns from a matrix $A \in [-1,1]^{n\times T}$, while approximately preserving all row sums. Our idea for doing so builds on the following seminal result by Spencer:

Theorem 2.3. *(Spencer's Theorem [Spencer, 1985]) For all matrices $A \in [-1,+1]^{n\times T}$ with $T \leq n$, there exists $x \in \{-1,+1\}^T$ such that $\|Ax\|_\infty = O(\sqrt{T\ln(en/T)})$. For all matrices $A \in [-1,+1]^{n\times T}$ with $T > n$, there exists $x \in \{-1,+1\}^T$ such that $\|Ax\|_\infty = O(\sqrt{n})$.*

We use Spencer's Theorem as follows: We find a vector $x \in \{-1,+1\}^T$ with $\|Ax\|_\infty = O(\sqrt{T\ln(en/T)})$ if $T \leq n$ and with $\|Ax\|_\infty = O(\sqrt{n}) = O(\sqrt{T})$ if $T > n$. Thus we always have $\|Ax\|_\infty = O(\sqrt{T\lg(2+n/T)})$. Consider now the i'th row of A and notice that $|\sum_{j:x_j=1} a_{ij} - \sum_{j:x_j=-1} a_{ij}| \leq \|Ax\|_\infty$. That is, for every single row, the sum of the entries corresponding to columns where x is 1, is almost equal (up to $\pm\|Ax\|_\infty$) to the sum over the columns where x is -1. Since the two together sum to the full row sum, it follows that the subset of columns with $x_i = 1$ *and* the subset of columns with $x_i = -1$ *both* preserve the row sum as required by Lemma 2.2. Since x has at most $T/2$ of either $+1$ or -1, it follows that we can find the desired subset of columns. We give the formal proof of Lemma 2.2 using Spencer's Theorem in Section 2.3

Algorithm 2 Sparsification

Input: Matrix $U \in [-1,1]^{n\times m}$, vector $w \in \mathbb{R}^m$ with $\|w\|_1 = 1$ and target $T \leq m$.
Output: A vector $w' \in \mathbb{R}^m$ with $\|w'\|_1 = 1$, $\|w'\|_0 \leq T$ and $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/T)/T})$.

```
1.  Let $w' \leftarrow w$.
2.  While $\|w'\|_0 > T$:
3.      Let $R$ be the indices of the $\|w'\|_0/3$ entries in $w'$ with largest absolute value.
4.      Let $\omega := \max_{i \notin R} |w_i|$ be the largest value of an entry outside $R$.
5.      Do Twice:
6.          Let $\pi(1), \pi(2), \ldots, \ldots, \pi(k)$ be the indices of the non-zero entries in $w'$ that are not in $R$.
7.          Let $A \in [-1,1]^{(n+1)\times k}$ have $a_{ij} = u_{i\pi(j)} w'_{\pi(j)}/\omega$ for $i \leq n$ and $a_{(n+1)j} = |w'_{\pi(j)}|/\omega$.
8.          Invoke Spencer's Theorem to find $x \in \{-1,1\}^k$ such that $\|Ax\|_\infty = O(\sqrt{k \lg(2+n/k)})$.
9.          Let $\sigma \in \{-1,1\}$ denote the sign such that $x_i = \sigma$ for at most $k/2$ indices $i$.
10.         Update $w'_i$ as follows:
11.             If there is a $j$ such that $i = \pi^{-1}(j)$ and $x_j = \sigma$: set $w'_i \leftarrow 2w'_i$.
12.             If there is a $j$ such that $i = \pi^{-1}(j)$ and $x_j \neq \sigma$: set $w'_i \leftarrow 0$.
13.             Otherwise ($i \in R$ or $w'_i = 0$): set $w'_i \leftarrow w'_i$.
14.     Update $w' \leftarrow w'/\|w'\|_1$.
15. Return $w'$.
```

We have summarized the entire sparsification algorithm in Algorithm 2.
We make a few remarks about our sparsification algorithm and SparsiBoost.

Running Time. While Spencer's original result (Theorem 2.3) is purely existential, recent follow up work [Lovett and Meka, 2015] show how to find the vector $x \in \{-1,+1\}^n$ in expected $\tilde{O}((n+T)^3)$ time, where $\tilde{O}$ hides polylogarithmic factors. A small modification to the algorithm was suggested in [Larsen, 2017]. This modification reduces the running time of Lovett and Meka's algorithm to expected $\tilde{O}(nT + T^3)$. This is far more desirable as T tends to be much smaller than n in boosting. Moreover, the nT term is already paid by simply running AdaBoostV. Using this in Step 7. of Algorithm 2, we get a total expected running time of $\tilde{O}(nT + T^3)$. We remark that these algorithms are randomized and lead to different vectors x on different executions.

Non-Negativity. Examining Algorithm 2, we observe that the weights of the input vector are only ever copied, set to zero, or scaled by a factor two. Hence if the input vector w has non-negative entries, then so has the final output vector w'. This may be quite important if one interprets the linear combination over hypotheses as a probability distribution.

Importance Sampling. Another natural approach one might attempt in order to prove our sparsification result, Theorem 2.2, would be to apply importance sampling. Importance sampling samples T entries from w with replacement, such that each entry i is sampled with probability $|w_i|$. It then returns the vector w' where coordinate i is equal to $\text{sign}(w_i)n_i/T$ where n_i denotes the number of times i was sampled and $\text{sign}(w_i) \in \{-1,1\}$ gives the sign of w_i. Analysing this method gives a w' with $\|Uw - Uw'\|_\infty = \Theta(\sqrt{\lg(n)/T})$ (with high probability), i.e. slightly worse than our approach based on discrepancy minimization. The loss in the lg is enough that if we use importance sampling in SparsiBoost, then we get no improvement over simply stopping AdaBoostV after T iterations.

2.1 Repeated Halving

As discussed earlier, our matrix-vector sparsification algorithm (Theorem 2.2 and Algorithm 2) was based on repeatedly invoking Lemma 2.1, which halves the number of non-zero entries. In this section, we prove formally that the errors resulting from these halving steps are dominated by the very last round of halving. For convenience, we first restate the two results:

Restatement of Theorem 2.2. *(Sparsification Theorem) For all matrices $U \in [-1,+1]^{n\times m}$, all $w \in \mathbb{R}^m$ with $\|w\|_1 = 1$ and all $T \le m$, there exists a vector w' where $\|w'\|_1 = 1$ and $\|w'\|_0 \le T$, such that $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/T)/T})$.*

Restatement of Lemma 2.1. *For all matrices $U \in [-1,+1]^{n\times m}$ and $w \in \mathbb{R}^m$ with $\|w\|_1 = 1$, there exists w' where $\|w'\|_0 \le \|w\|_0/2$ and $\|w'\|_1 = 1$, such that $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})$.*

We thus set out to prove Theorem 2.2 assuming Lemma 2.1.

Proof. Let us call the initial weight vector $w^{(0)} = w$. We use Lemma 2.1 repeatedly and get $w^{(1)}, w^{(2)}, \dots, w^{(k)} = w'$ such that $\|w'\|_0 \le T$ as wanted and every $w^{(i)}$ has $\|w^{(i)}\|_1 = 1$. Let $T_i = \|w^{(i)}\|_0$ and notice this gives a sequence of numbers $T_0, T_1, \dots, T_k$ where $T_i \le T_{i-1}/2$, $T_{k-1} > T$ and $T_k \le T$. In particular, it holds that $T_{k-1}2^{k-i-1} \le T_i$. The total difference $\|Uw^{(0)} - Uw^{(k)}\|_\infty$ is then by the triangle inequality no more than $\sum_{i=0}^{k-1}\|Uw^{(i)} - Uw^{(i+1)}\|_\infty$. Each of these terms are bounded by Lemma 2.1 which gives us

$$O\left(\sum_{i=0}^{k-1}\sqrt{\lg(2+n/T_i)/T_i}\right) = O\left(\sum_{i=0}^{k-1}\sqrt{\lg(2+n/T_{k-1})/(T_{k-1}2^{k-i-1})}\right) =$$
$$O\left(\sum_{i=0}^{\infty}\sqrt{\lg(2+n/T_{k-1})/(T_{k-1}2^{i})}\right) = O\left(\left(\sqrt{\lg(2+n/T_{k-1})/T_{k-1}}\right)\sum_{i=0}^{\infty}1/\sqrt{2^i}\right) =$$
$$O\left(\sqrt{\lg(2+n/T)/T}\right).$$

The last step follows since $T_{k-1} > T$. We have thus shown that the final vector $w' = w^{(k)}$ has $\|w'\|_0 \le T$, $\|w'\|_1 = 1$ and $\|Uw - Uw'\|_\infty = \|Uw^{(0)} - Uw^{(k)}\|_\infty = O\left(\sqrt{\lg(2+n/T)/T}\right)$. This completes the proof of Theorem 2.2. □

2.2 Halving via Row-Sum Preservation

In this section, we give the formal details of how to use our result on row-sum preservation to halve the number of non-zeroes in a vector w. Let us recall the two results:

Restatement of Lemma 2.1. *For all matrices $U \in [-1,+1]^{n\times m}$ and $w \in \mathbb{R}^m$ with $\|w\|_1 = 1$, there exists w' where $\|w'\|_0 \le \|w\|_0/2$ and $\|w'\|_1 = 1$, such that $\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})$.*

Restatement of Lemma 2.2. *For all matrices $A \in [-1,1]^{n\times T}$ there exists a submatrix $\hat{A} \in [-1,1]^{n\times k}$ consisting of $k \le T/2$ distinct columns from A, such that for all i, it holds that $\sum_{j=1}^{k}\hat{a}_{ij} \in \frac{1}{2}\sum_{j=1}^{T}a_{ij} \pm \left[O(\sqrt{T\lg(2+n/T)})\right]$.*

We now use Lemma 2.2 to prove Lemma 2.1.

Proof. Our procedure corresponds to steps 2.-13. in Algorithm 2. To clarify the proof, we expand the steps a bit and introduce some additional notation. Let w be the input vector and let R be the indices of the $\|w\|_0/3$ entries in w with largest absolute value. Define $\bar{w}$ such that $\bar{w}_i = w_i$ if $i \in R$ and $\bar{w}_i = 0$ otherwise, that is, $\bar{w}$ contains the largest $\|w\|_0/3$ entries of w and is zero elsewhere. Similarly, define $\hat{w} = w - \bar{w}$ as the vector containing all but the largest $\|w\|_0/3$ entries of w.

We will use Lemma 2.2 twice in order to obtain a vector w'' with $\|w''\|_0 \leq \|\hat{w}\|_0/4$, $\|w''\|_1 \in \|\hat{w}\|_1 \pm [O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})]$ and $\|U\hat{w} - Uw''\|_\infty = O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})$. Moreover, w'' will only be non-zero in entries where $\hat{w}$ is also non-zero. We finally set $w' = (\bar{w}+w'')/\|\bar{w}+w''\|_1$ as our sparsified vector.

We first argue that if we can indeed produce the claimed w'', then w' satisfies the claims in Lemma 2.1: Observe that $\|w'\|_0 = \|\bar{w}\|_0 + \|w''\|_0 \leq \|w\|_0/3 + (2\|w\|_0/3)/4 \leq \|w\|_0/2$ as desired. Clearly we also have $\|w'\|_1 = 1$ because of the normalization. Now observe that:

$$\begin{aligned}
\|Uw - Uw'\|_\infty &\leq \|Uw - U(\bar{w}+w'')\|_\infty + \|U(\bar{w}+w'') - Uw'\|_\infty \\
&= \|U(\bar{w}+\hat{w}) - U(\bar{w}+w'')\|_\infty + \|U(w'\|\bar{w}+w''\|_1) - Uw'\|_\infty \\
&= \|U\hat{w} - Uw''\|_\infty + \|Uw'(\|\bar{w}+w''\|_1 - 1)\|_\infty \\
&= O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0}) + \|Uw'\|_\infty(\|\bar{w}+w''\|_1 - 1) \\
&= O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0}) + \|Uw'\|_\infty(\|\bar{w}\|_1 + \|w''\|_1 - 1).
\end{aligned}$$

In the last step, we used that w'' has non-zeroes only where $\hat{w}$ has non-zeroes (and thus the non-zeroes of $\bar{w}$ and w'' are disjoint). Since $\|w'\|_1 = 1$ and all entries of U are in $[-1,1]$, we get $\|Uw'\|_\infty \leq 1$. We also see that:

$$\begin{aligned}
\|\bar{w}\|_1 + \|w''\|_1 - 1 &\in \|\bar{w}\|_1 + \|\hat{w}\|_1 - 1 \pm [O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})] \\
&= \|\bar{w}+\hat{w}\|_1 - 1 \pm [O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})] \\
&= \|w\|_1 - 1 \pm [O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})] \\
&= \pm[O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})].
\end{aligned}$$

We have thus shown that

$$\|Uw - Uw'\|_\infty = O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})$$

as claimed. So what remains is to argue that we can find w'' with the claimed properties. Finding w'' corresponds to the Do Twice part of Algorithm 2 (steps 3.-12.). We first compute $\omega = \max_i |\hat{w}_i|$ and let $w'' = \hat{w}$. We then execute the following twice:

1. Define $\pi(1), \ldots, \pi(\ell)$ as the list of all the indices i where $w''_i \neq 0$. Also define $\pi^{-1}(j)$ as the index i such that $i = \pi(j)$, i.e. $\pi^{-1}(j)$ is the index of the j'th non-zero coordinate of w''.

2. Form the matrix $A \in [-1,1]^{(n+1)\times\ell}$ where $a_{ij} = u_{i\pi(j)} w''_{\pi(j)}/\omega$ for $i \leq n$ and $a_{(n+1)j} = |w''_{\pi(j)}|/\omega$.

3. Invoke Lemma 2.2 to obtain a matrix $\hat{A}$ consisting of no more than $k \leq \ell/2$ distinct columns from A where for all rows i, we have $\sum_{j=1}^{k} \hat{a}_{ij} \in \frac{1}{2}\sum_{j=1}^{\ell} a_{ij} \pm \left[O(\sqrt{T \lg(2+n/T)})\right]$.

4. Update w'' as follows:

(a) We let $w''_i \leftarrow 2w''_i$ if there is a j such that $i = \pi^{-1}(j)$ and j is a column in $\hat{A}$.

(b) Otherwise we let $w''_i \leftarrow 0$.

After the two executions, we get that the number of non-zeroes in w'' is at most $\|\hat{w}\|_0/4$ as claimed. Step 4. above effectively scales every coordinate of w'' that corresponds to a column that was included in $\hat{A}$ by a factor two. It sets coordinates not chosen by $\hat{A}$ to 0. What remains is to argue that $\|U\hat{w} - Uw''\|_\infty = O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})$ and that $\|w''\|_1 \in \|\hat{w}\|_1 \pm [O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})]$. For this, let $A \in [-1,1]^{(n+1)\times T}$ denote the matrix formed in step 2. during the first iteration. Then $T = \|\hat{w}\|_0$. Let $\bar{A} \in [-1,1]^{(n+1)\times k}$ denote the matrix returned in step 3. of the first iteration. Similarly, let $\hat{A}$ denote the matrix formed in step 2. of the second iteration and let $\hat{\hat{A}} \in [-1,1]^{(n+1)\times k'}$ denote the matrix returned in step 3. of the second iteration. We see that $\hat{A} = 2\bar{A}$. By Lemma 2.2, it holds for all rows i that:

$$\begin{aligned}
2\sum_{j=1}^{k'} \hat{\hat{a}}_{ij} &\in \sum_{j=1}^{k} \hat{a}_{ij} \pm \left[O(\sqrt{k\lg(2+n/k)})\right] \\
&= 2\sum_{j=1}^{k} \bar{a}_{ij} \pm \left[O(\sqrt{k\lg(2+n/k)})\right] \\
&\subseteq \left(\sum_{j=1}^{T} a_{ij} \pm \left[O(\sqrt{k\lg(2+n/k)})\right]\right) \pm \left[O(\sqrt{T\lg(2+n/T)})\right] \\
&\subseteq \sum_{j=1}^{T} a_{ij} \pm \left[O(\sqrt{T\lg(2+n/T)})\right].
\end{aligned}$$

But $2\sum_{j=1}^{k'} \hat{\hat{a}}_{ij} = \sum_j u_{ij} w''_j/\omega = (Uw'')_i/\omega$ and $\sum_{j=1}^{T} a_{ij} = \sum_j u_{ij}\hat{w}_j/\omega = (U\hat{w})_i/\omega$. Hence:

$$\begin{aligned}
(Uw'')_i/\omega &\in (U\hat{w})_i/\omega \pm \left[O(\sqrt{T\lg(2+n/T)})\right] \Rightarrow \\
(Uw'')_i &\in (U\hat{w})_i \pm \left[O(\omega\sqrt{T\lg(2+n/T)})\right]
\end{aligned}$$

which implies that $\|U\hat{w} - Uw''\|_\infty = O(\omega\sqrt{T\lg(2+n/T)})$. But $T = \|\hat{w}\|_0 \le \|w\|_0$ and $\omega = \max_i |\hat{w}|_i$. Since $\|w\|_1 = 1$ and $\bar{w}$ contained the largest $\|w\|_0/3$ entries, we must have $\omega \le 3/\|w\|_0$. Inserting this, we conclude that

$$\|U\hat{w} - Uw''\|_\infty = O(\sqrt{\|w\|_0\lg(2+n/\|w\|_0)}/\|w\|_0) = O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0}).$$

The last step is to prove that $\|w''\|_1 \in \|\hat{w}\|_1 \pm [O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})]$. Here we focus on row $(n+1)$ of A and use that we showed above that $2\sum_{j=1}^{k'} \hat{\hat{a}}_{(n+1)j} = \sum_{j=1}^{T} a_{(n+1)j} \pm \left[O(\sqrt{T\lg(2+n/T)})\right]$. This time, we have $\sum_{j=1}^{T} a_{(n+1)j} = \sum_{j=1}^{T} |\hat{w}_j|/\omega = \|\hat{w}\|_1/\omega$ and $2\sum_{j=1}^{k'} \hat{\hat{a}}_{(n+1)j} = \sum_j |w''_j|/\omega = \|w''\|_1/\omega$. Therefore we conclude that $\|w''\|_1 \in \|\hat{w}\|_1 \pm \left[O(\omega\sqrt{T\lg(2+n/T)})\right] \subseteq \|\hat{w}\|_1 \pm [O(\sqrt{\lg(2+n/\|w\|_0)/\|w\|_0})]$ as claimed. □

2.3 Finding a Column Subset

In this section we give the detailed proof of how to use Spencer's theorem to select a subset of columns in a matrix while approximately preserving its row sums. The two relevant lemmas are restated here for convenience:

Restatement of Lemma 2.2. *For all matrices $A \in [-1,1]^{n\times T}$ there exists a submatrix $\hat{A} \in [-1,1]^{n\times k}$ consisting of $k \le T/2$ distinct columns from A, such that for all i, it holds that $\sum_{j=1}^{k} \hat{a}_{ij} \in \frac{1}{2}\sum_{j=1}^{T} a_{ij} \pm \left[O(\sqrt{T \lg(2+n/T)})\right]$.*

and

Restatement of Theorem 2.3. *(Spencer's Theorem [Spencer, 1985]) For all matrices $A \in [-1,+1]^{n\times T}$ with $T \le n$, there exists $x \in \{-1,+1\}^T$ such that $\|Ax\|_\infty = O(\sqrt{T \ln(en/T)})$. For all matrices $A \in [-1,+1]^{n\times T}$ with $T > n$, there exists $x \in \{-1,+1\}^T$ such that $\|Ax\|_\infty = O(\sqrt{n})$.*

We now prove Lemma 2.2 using Spencer's Theorem:

Proof. Let $A \in [-1,1]^{n\times T}$ as in Lemma 2.2 and let a_i denote the i'th row of A. By Spencer's Theorem there must exist $x \in \{-1,+1\}^T$ s.t. $\|Ax\|_\infty = \max_i |\langle a_i, x\rangle| = O(\sqrt{T \ln(en/T)})$ if $T \le n$ and $\|Ax\|_\infty = O(\sqrt{n}) = O(\sqrt{T})$ if $T > n$. This ensures that $\|Ax\|_\infty = O(\sqrt{T \lg(2+n/T)})$ Either the number of $+1$'s or -1's in x will be $\le \frac{T}{2}$. Assume without loss of generality that the number of $+1$'s is $k \le \frac{T}{2}$. Construct a matrix $\hat{A}$ with columns of A corresponding to the entries of x that are $+1$. Then $\hat{A} \in [-1,1]^{n\times k}$ for $k \le \frac{T}{2}$ as wanted. It is then left to show that $\hat{A}$ preserves the row sums. For all i, we have:

$$\begin{aligned}
|\langle a_i, x\rangle| = \left|\sum_{j=1}^{T} a_{ij}x_j\right| = O(\sqrt{T \lg(2+n/T)}) \quad &\Rightarrow \quad \sum_j a_{ij}x_j \in \pm\left[O(\sqrt{T \lg(2+n/T)})\right] \\
&\Rightarrow \sum_{j:x_j=+1} a_{ij} - \sum_{j:x_j=-1} a_{ij} \in \pm\left[O(\sqrt{T \lg(n/T)})\right] \\
&\Rightarrow \sum_{j:x_j=+1} a_{ij} \in \sum_{j:x_j:-1} a_{ij} \pm \left[O(\sqrt{T \lg(n/T)})\right].
\end{aligned}$$

Using this we can bound the i'th row sum of $\hat{A}$

$$\begin{aligned}
\sum_{j=1}^{k} \hat{a}_{ij} = \sum_{j:x_j=+1} a_{ij} &= \frac{1}{2}\sum_{j:x_j=+1} a_{ij} + \frac{1}{2}\sum_{j:x_j=+1} a_{ij} \\
&\in \frac{1}{2}\sum_{j:x_j=+1} a_{ij} + \frac{1}{2}\left(\sum_{j:x_j=-1} a_{ij} \pm \left[O(\sqrt{T \lg(2+n/T)})\right]\right) \\
&= \frac{1}{2}\sum_j a_{ij} \pm \left[O(\sqrt{T \lg(2+n/T)})\right].
\end{aligned}$$

This concludes the proof. □

3 Lower Bound

This section concerns our lower bound, which states that there exist a data set and corresponding set of base hypotheses $\mathcal{H}$, such that if one uses only T of the base hypotheses in $\mathcal{H}$, then one cannot obtain a gap smaller than $\Omega(\sqrt{\lg(n/T)/T})$. Similarly to the approach taken in [Nie et al., 2013], we model a data set $D = \{(x_i, y_i)\}_{i=1}^{n}$ of n data points and a corresponding set of k base hypotheses $\mathcal{H} = \{h_1, \ldots, h_k\}$ as an $n \times k$ matrix A. The entry $a_{i,j}$ is equal to $y_i h_j(x_i)$. We prove our lower bound for binary classification where the hypotheses take values only amongst $\{-1,+1\}$, meaning that $A \in \{-1,+1\}^{n\times k}$. Thus an entry $a_{i,j}$ is $+1$ if hypothesis h_j is correct on point x_i and it is -1 otherwise.

We remark that proving the lower bound under the restriction that $h_j(x_i)$ is among $\{-1,+1\}$ instead of $[-1,+1]$ only strengthens the lower bound.

Notice that if $w \in \mathbb{R}^k$ is a vector with $\|w\|_1 = 1$, then $(Aw)_i$ gives exactly the margin on data point (x_i, y_i) when using the linear combination $\sum_j w_j h_j$ of base hypotheses. The optimal minimum margin ρ^* for a matrix A is thus equal to $\rho^* := \max_{w\in\mathbb{R}^k:\|w\|_1=1} \min_i (Aw)_i$. We now seek a matrix A for which ρ^* is at least $\Omega(\sqrt{\lg(n/T)/T})$ larger than $\min_i(Aw)_i$ for all w with $\|w\|_0 \le T$ and $\|w\|_1 = 1$. If we can find such a matrix, it implies the existence of a data set (rows) and a set of base hypotheses (columns) for which any linear combination of up to T base hypotheses has a gap of $\Omega(\sqrt{\lg(n/T)/T})$. The lower bound thus holds regardless of how an algorithm would try to determine which linear combination to construct.

When showing the existence of a matrix A with a large gap, we will fix $k = n$, i.e. the set of base hypotheses $\mathcal{H}$ has cardinality equal to the number of data points. The following theorem shows the existence of the desired matrix A:

Theorem 3.1. *There exists a universal constant $C > 0$ such that for all sufficiently large n and all T with $\ln n \le T \le n/C$, there exists a matrix $A \in \{-1,+1\}^{n\times n}$ such that: 1) Let $v \in \mathbb{R}^n$ be the vector with all coordinates equal to $1/n$. Then all coordinates of Av are greater than or equal to $-O(1/\sqrt{n})$. 2) For every vector $w \in \mathbb{R}^n$ with $\|w\|_0 \le T$ and $\|w\|_1 = 1$, it holds that:* $\min_i (Aw)_i \le -\Omega\left(\sqrt{\lg(n/T)/T}\right)$.

Quite surprisingly, Theorem 3.1 shows that for any T with $\ln n \le T \le n/C$, there is a matrix $A \in \{-1,+1\}^{n\times n}$ for which the *uniform* combination of base hypotheses $\sum_{j=1}^n h_j/n$ has a minimum margin that is much higher than anything that can be obtained using only T base hypotheses. More concretely, let A be a matrix satisfying the properties of Theorem 3.1 and let $v \in \mathbb{R}^n$ be the vector with all coordinates $1/n$. Then $\rho^* := \max_{w\in\mathbb{R}^k:\|w\|_1=1} \min_i(Aw)_i \ge \min_i(Av)_i = -O(1/\sqrt{n})$. By the second property in Theorem 3.1, it follows that any linear combination of at most T base hypotheses must have a gap of $-O(1/\sqrt{n}) - \left(-\Omega\left(\sqrt{\lg(n/T)/T}\right)\right) = \Omega\left(\sqrt{\lg(n/T)/T}\right)$. This is precisely the claimed lower bound and also shows that our vector sparsification algorithm from Theorem 2.2 is optimal for any $T \le n/C$. To prove Theorem 3.1, we first show the existence of a matrix $B \in \{-1,+1\}^{n\times n}$ having the second property. We then apply Spencer's Theorem (Theorem 2.3) to "transform" B into a matrix A having both properties. We find it quite surprising that Spencer's discrepancy minimization result finds applications in both our upper and lower bound.

The following lemma states that there exists a matrix with the second property in Theorem 3.1

Lemma 3.1. *There exists a universal constant $C > 0$ such that for all sufficiently large n and all T with $\ln n \le T \le n/C$, there exists a matrix $A \in \{-1,+1\}^{n\times n}$ such that for every vector $w \in \mathbb{R}^n$ with $\|w\|_0 \le T$ and $\|w\|_1 = 1$ it holds that:* $\min_i(Aw)_i \le -\Omega\left(\sqrt{\ln(n/T)/T}\right)$.

We now show how to prove Theorem 3.1 by using Lemma 3.1 and Spencer's Theorem.

Proof. Let B be a matrix satisfying the statement in Lemma 3.1. Using Spencer's Theorem (Theorem 2.3), we get that there exists a vector $x \in \{-1,+1\}^n$ such that $\|Bx\|_\infty = O(\sqrt{n\ln(en/n)}) = O(\sqrt{n})$. Now form the matrix A which is equal to B, except that the i'th column is scaled by x_i. Then $A\mathbf{1} = Bx$ where $\mathbf{1}$ is the all-ones vectors. Normalizing the all-ones vector by a factor $1/n$ yields the vector v with all coordinates equal to $1/n$. Moreover, it holds that $\|Av\|_\infty = \|Bx\|_\infty/n = O(1/\sqrt{n})$, which in turn implies that $\min_i(Av)_i \ge -O(1/\sqrt{n})$.

Now consider any vector $w \in \mathbb{R}^n$ with $\|w\|_0 \le T$ and $\|w\|_1 = 1$. Let $\tilde{w}$ be the vector obtained from w by multiplying w_i by x_i. Then $Aw = B\tilde{w}$. Furthermore $\|\tilde{w}\|_1 = \|w\|_1 = 1$ and $\|\tilde{w}\|_0 = \|w\|_0 \le T$. It follows from Lemma 3.1 and our choice of B that $\min_i(Aw)_i = \min_i(B\tilde{w})_i \le -\Omega\left(\sqrt{\ln(n/T)/T}\right)$. □

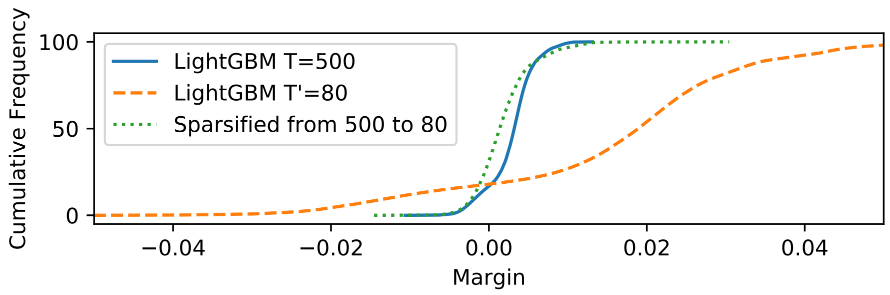


Figure 1: The plot depicts the cumulative margins of three classifiers: (1) a LightGBM classifier with 500 hypotheses (2) a classifier sparsified from 500 to 80 hypotheses and (3) a LightGBM classifier with 80 hypotheses.

The proof of Lemma 3.1 is omitted in this thesis. It can be found in the full ArXiV version of [Grønlund et al., 2019].

4 Experiments

Gradient Boosting [Mason et al., 2000, Friedman, 2001] is probably the most popular boosting algorithm in practice. It has several highly efficient open-source implementations [Chen and Guestrin, 2016, Ke et al., 2017, Prokhorenkova et al., 2018] and obtain state-of-the-art performance in many machine learning tasks [Ke et al., 2017]. In this section we demonstrate how our sparsification algorithm can be combined with Gradient Boosting. For simplicity we consider a single dataset in this section, the Flight Delay dataset [air,], see Appendix A for similar results on other dataset.

We train a classifier with $T = 500$ hypotheses using LightGBM [Ke et al., 2017] which we sparsify using Theorem 2.2 to have $T' = 80$ hypotheses. The sparsified classifier is guaranteed to preserve all margins of the original classifier to an additive $O(\sqrt{\lg(n/T')/T'})$. The cumulative margins of the sparsified classifier and the original classifier are depicted in Figure 1. Furthermore, we also depict the cumulative margins of a LightGBM classifier trained to have $T' = 80$ hypotheses. First observe the difference between the LightGBM classifiers with $T = 500$ and $T' = 80$ hypotheses (blue and orange in Figure 1). The margins of the classifier with $T = 500$ hypotheses vary less. It has fewer points with a large margin, but also fewer points with a small margin. The margin distribution of the sparsified classifier with $T' = 80$ approximates the margin distribution of the LightGBM classifier with $T = 500$ hypotheses. The $T = 500$ classifier had best generalization error, so we might expect the sparsified classifier to generalize better than the $T' = 80$ classifier. To investigate this, we performed additional experiments computing AUC and classification accuracy of several sparsified classifiers and LightGBM classifiers on a test set (we show the results for multiple sparsified classifiers due to the randomization in the discrepancy minimization algorithms). The experiments indeed show that the sparsified classifiers outperform the LightGBM classifiers with the same number of hypotheses. See Figure 2 for test AUC and test classification accuracy.

Further Experiments and Importance Sampling. Inspired by the experiments in [Wang et al., 2008, Ke et al., 2017, Chen and Guestrin, 2016] we also performed the above experiments on the Higgs [Whiteson, 2014] and Letter [Dheeru and Karra Taniskidou, 2017] datasets. See Appendix A

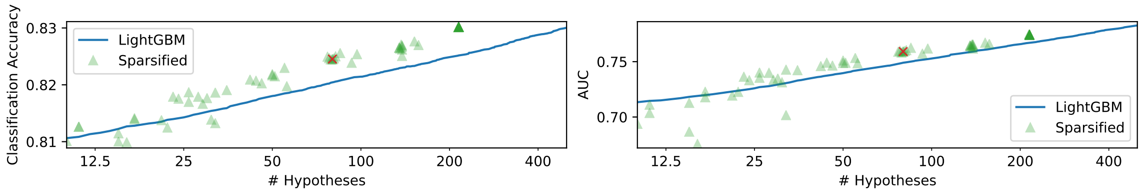


Figure 2: The plot depicts test AUC and test classification accuracy of a LightGBM classifier during training as the number of hypotheses increase (in blue). Notice the x-axis is logarithmically scaled. The final classifier with 500 hypotheses was sparsified with Theorem 2.2 multiple times to have between $T/2$ to $T/16$ hypotheses. The green triangles show test AUC and test accuracy of the resulting sparsified classifiers. The red cross represents the sparsified classifier used to plot the cumulative margins in Figure 1.

for (1) further experimental details and (2) for cumulative margin, AUC and test accuracy plots on all dataset for different values of n and T.

As mentioned in Section 2, one could use importance sampling for sparsification. It has a slightly worse theoretical guarantee, but might work better in practice. Appendix A also contains test AUC and test accuracy of the classifiers that result from using importance sampling instead of our algorithm based on discrepancy minimization. Our algorithm and importance sampling are both random so the experiments were repeated several times. On average over the experiments, our algorithm obtains a better test AUC and classification accuracy than importance sampling.

Minimal Margin Experiment: AdaBoostV guarantees a gap of v for $T = O(\lg(n)/v^2)$ hypotheses while SparsiBoost guarantees the same gap using just $T = O(\lg(nv^2)/v^2)$ hypotheses. It might be that the asymptotic bounds do not reflect the relative practical performance of the two algorithms. To investigate this, we compare the minimal margin obtained by SparsiBoost against that of AdaBoost and AdaBoostV in the experimental setting of [Reyzin and Schapire, 2006] using $T = 100$ decision stumps. The experiment is performed on the Letter, Breast Cancer, Diabetes, Ionosphere and German dataset, all available in the UCI repository[Dua and Graff, 2017][1]. Since this experiment only concerns minimal margin, and not whether minimal margin predicts test error, we train on the entire dataset. See Figure 3 for a plot of minimal margin during training for the Letter and Diabetes datasets, all other plots can be found in Figure 11 located in the appendix. AdaBoostV significantly outperforms AdaBoost as one would expect, since AdaBoostV explicitly maximize the minimal margin which AdaBoost does not. SparsiBoost obtains an even better minimal margin on the Letter and Diabetes dataset. Throughout all datasets (see Figure 11 in the appendix), SparsiBoost obtains similar or better minimal margin compared to both AdaBoostV and AdaBoost.

A Python/NumPy implementation of our sparsification algorithm (Theorem 2.2) can be found at:

`https://github.com/AlgoAU/DiscMin`

[1] For convenience we loaded the breast cancer data using scikit learn [Pedregosa et al., 2011].

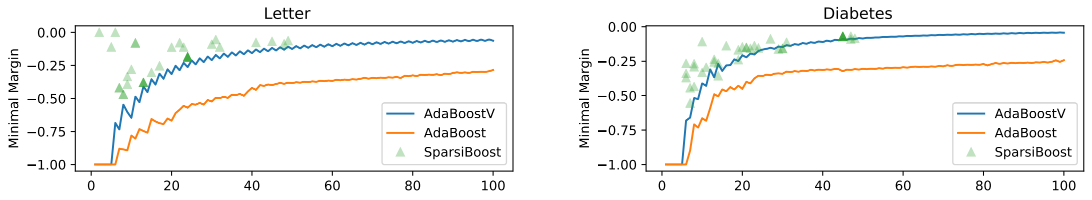


Figure 3: Minimal margin obtained by AdaBoost, AdaBoostV and SparsiBoost on the Letter and Diabetes data set using decision trees of depth 1. The trees are trained using sklearns [Pedregosa et al., 2011] DecisionTreeClassifier.

5 Conclusion

A long line of research into obtaining a large minimal margin using few hypotheses [Breiman, 1999, Grove and Schuurmans, 1998, Bennett et al., 2000, Rätsch and Warmuth, 2002] culminated with the AdaBoostV [Rätsch and Warmuth, 2005] algorithm. AdaBoostV was later conjectured by [Nie et al., 2013] to provide an optimal trade-off between minimal margin and number of hypotheses. In this article, we introduced SparsiBoost which refutes the conjecture of [Nie et al., 2013]. Furthermore, we show a matching lower bound, which implies that SparsiBoost is optimal.

The key idea behind SparsiBoost, is a sparsification algorithm that reduces the number of hypotheses while approximately preserving the entire margin distribution. Experimentally, we combine our sparsification algorithm with LightGBM. We find that the sparsified classifiers obtains a better margin distribution, which typically yields a better test AUC and test classification error when compared to a classifier trained directly to the same number of hypotheses.

References

[air,] Flight delay data. `https://github.com/szilard/benchm-ml#data`.

[Bennett et al., 2000] Bennett, K. P., Demiriz, A., and Shawe-Taylor, J. (2000). A column generation algorithm for boosting. In *ICML*, pages 65–72.

[Breiman, 1999] Breiman, L. (1999). Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

[Dheeru and Karra Taniskidou, 2017] Dheeru, D. and Karra Taniskidou, E. (2017). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

[Dua and Graff, 2017] Dua, D. and Graff, C. (2017). UCI machine learning repository.

[Freund et al., 1999] Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612.

[Freund and Schapire, 1995] Freund, Y. and Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer.

[Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

[Gao and Zhou, 2013] Gao, W. and Zhou, Z.-H. (2013). On the doubt about margin explanation of boosting. *Artificial Intelligence*, 203:1–18.

[Grønlund et al., 2019] Grønlund, A., Larsen, K. G., and Mathiasen, A. (2019). Optimal Minimal Margin Maximization with Boosting. In *International Conference on Machine Learning*. PMLR.

[Grove and Schuurmans, 1998] Grove, A. J. and Schuurmans, D. (1998). Boosting in the limit: Maximizing the margin of learned ensembles. In *AAAI/IAAI*, pages 692–699.

[Ke et al., 2017] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154.

[Klein and Young, 1999] Klein, P. and Young, N. (1999). On the number of iterations for dantzig-wolfe optimization and packing-covering approximation algorithms. *Lecture Notes in Computer Science*, 1610:320–327.

[Koltchinskii et al., 2001] Koltchinskii, V., Panchenko, D., and Lozano, F. (2001). Some new bounds on the generalization error of combined classifiers. In *Advances in neural information processing systems*, pages 245–251.

[Larsen, 2017] Larsen, K. G. (2017). Constructive discrepancy minimization with hereditary L2 guarantees. *CoRR*, abs/1711.02860.

[Lovett and Meka, 2015] Lovett, S. and Meka, R. (2015). Constructive discrepancy minimization by walking on the edges. *SIAM Journal on Computing*, 44(5):1573–1582.

[Mason et al., 2000] Mason, L., Baxter, J., Bartlett, P. L., and Frean, M. R. (2000). Boosting algorithms as gradient descent. In Solla, S. A., Leen, T. K., and Müller, K., editors, *Advances in Neural Information Processing Systems 12*, pages 512–518. MIT Press.

[McKinney et al., 2010] McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

[Nie et al., 2013] Nie, J., Warmuth, M., Vishwanathan, S., and Zhang, X. (2013). Open problem: Lower bounds for boosting with hadamard matrices. *Journal of Machine Learning Research*, 30:1076–1079.

[Oliphant, 2006] Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Prokhorenkova et al., 2018] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2018). Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pages 6637–6647.

[Rätsch and Warmuth, 2002] Rätsch, G. and Warmuth, M. K. (2002). Maximizing the margin with boosting. In *COLT*, volume 2375, pages 334–350. Springer.

[Rätsch and Warmuth, 2005] Rätsch, G. and Warmuth, M. K. (2005). Efficient margin maximizing with boosting. *Journal of Machine Learning Research*, 6(Dec):2131–2152.

[Reyzin and Schapire, 2006] Reyzin, L. and Schapire, R. E. (2006). How boosting the margin can also boost classifier complexity. In *Proceedings of the 23rd international conference on Machine learning*, pages 753–760. ACM.

[Schapire et al., 1998] Schapire, R. E., Freund, Y., Bartlett, P., Lee, W. S., et al. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686.

[Spencer, 1985] Spencer, J. (1985). Six standard deviations suffice. *Transactions of the American mathematical society*, 289(2):679–706.

[Wang et al., 2008] Wang, L., Sugiyama, M., Yang, C., Zhou, Z.-H., and Feng, J. (2008). On the margin explanation of boosting algorithms. In *COLT*, pages 479–490. Citeseer.

[Whiteson, 2014] Whiteson, D. (2014). Higgs data set. `https://archive.ics.uci.edu/ml/datasets/HIGGS`.

A Additional Experiments and Experimental Details

We train a LightGBM classifier and sparsify it in two ways: Theorem 2.2 and importance sampling. Both sparsification algorithms are random so we repeated both sparsifications 10 times. The experiment was performed on the following dataset:

- Inspired by the XGBoost article [Chen and Guestrin, 2016] we used the Higgs dataset [Whiteson, 2014]. We shuffled the 10^7 training examples and selected the first 10^6 examples for training and the following 10^6 points for examples.
- Inspired by the LightGBM article [Ke et al., 2017] we used the Flight Delay dataset [air,]. The dataset contain delay times and was turned into binary classification by predicting if the flight was delayed or not. All features of the 10^7 training examples were one-hot encoded with the Pandas [McKinney et al., 2010] function *get_dummies()* which yielded 660 features. We shuffled the 10^7 training examples and selected the first 10^6 examples for training and the following 10^6 examples for testing.
- Inspired by the equivalence margin article [Wang et al., 2008] we used the Letter dataset [Dheeru and Karra Taniskidou, 2017]. It was turned into a binary classification problem as done in [Wang et al., 2008]. We shuffled the 20000 examples and used the first 10000 for training and the last 10000 for testing.

The initial classifiers were trained with LightGBM using default parameters (except for the Letter dataset where we used LightGBM decision stumps inspired by [Wang et al., 2008]). The rest of this article contain figures that show different variants our experiments. Each figure concerns a single dataset for a choice of *number of hypotheses* T and *number of points* n. For example Figure 4 contains results for our experiment on the Flight Delay dataset with $T = 100$ hypotheses and $n = 250000$ training and test points. It contains a margin plot similar to Figure 1 and test AUC/accuracy plot similar to Figure 2. See Table 1 for an overview of all experiments.

Dataset	**n**	**T**
Letter	10000	100
Flight Delay	250000	100
Flight Delay	500000	250
Flight Delay	1000000	500
Higgs	250000	100
Higgs	500000	250
Higgs	1000000	500

Table 1: The different experimental settings.

Finally, we would like to acknowledge the NumPy, Pandas and Scikit-Learn libraries [Oliphant, 2006, McKinney et al., 2010, Pedregosa et al., 2011].

A.1 Correcting Sparsified Predictions

In some cases the classification accuracy of a sparsified classifier is very poor even though the test AUC is good (e.g. accuracy 60% but AUC 0.80). The AUC depends on the relative ordering of predictions while classification accuracy depends on whether each prediction is above or below zero. This lead us to believe that the poor test classification accuracy was caused by a bad offset. Maybe we should predict +1 if points where above -0.01 instead of 0. In other words, it seemed that the sparsified classifiers skewed the bias term of the original classifier. To fix this we computed the bias term that yielded the largest classification accuracy on the training set. This can be done by sorting predictions and then trying every possible offset, one for each point, taking just $O(n \lg(n))$ time. For the airline dataset this typically improved the sparsified classifiers accuracy from 60% to 80%.

In this way we "corrected" the bias of all sparsified predictions. All test classification accuracies reported are corrected in this sense (including Figure 2).

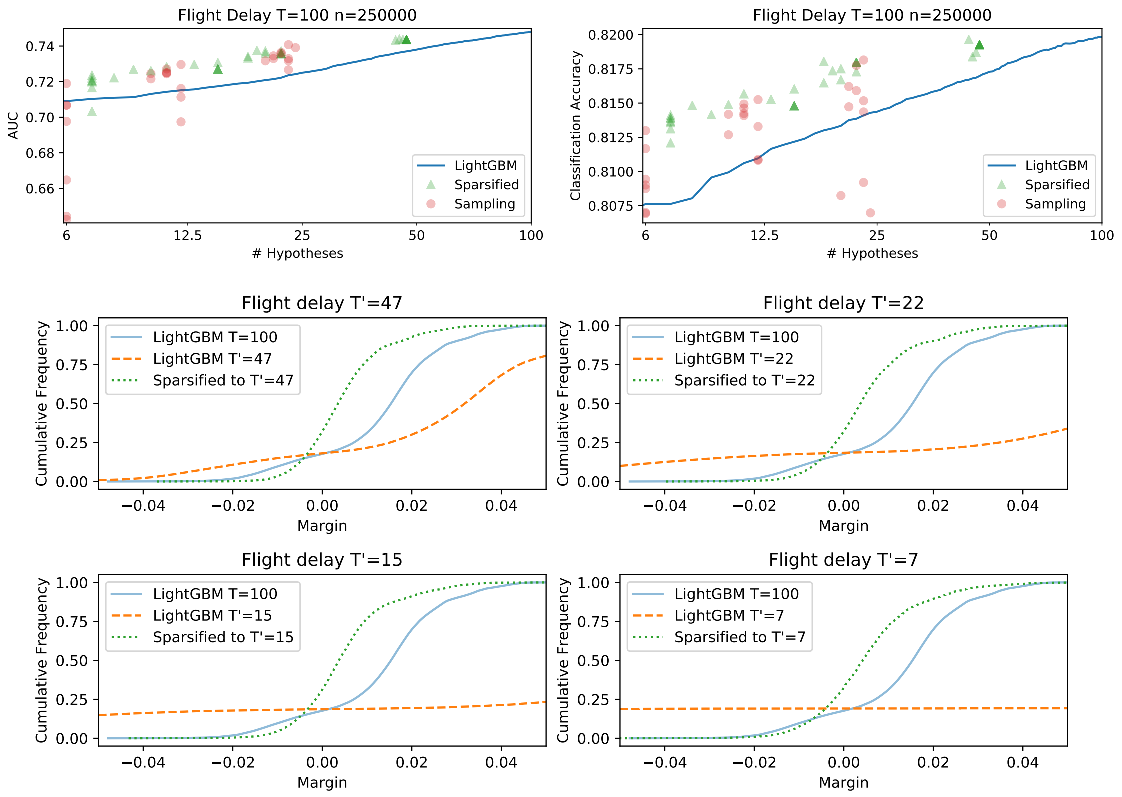


Figure 4: Similar to Figure 2 and Figure 1, but for Flight Delay with $T = 100$ and 250000 training and test points.

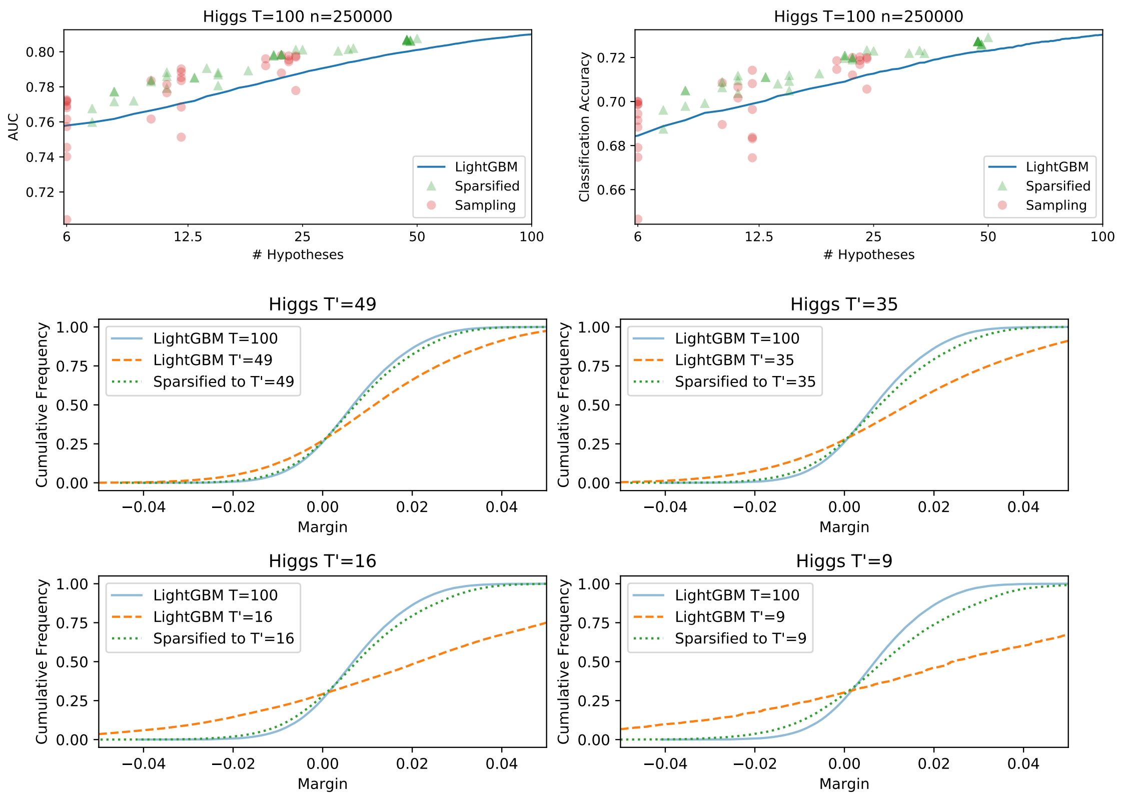


Figure 5: Similar to previous plot, but for Higgs with $T = 100$ hypotheses and 250000 training and test points.

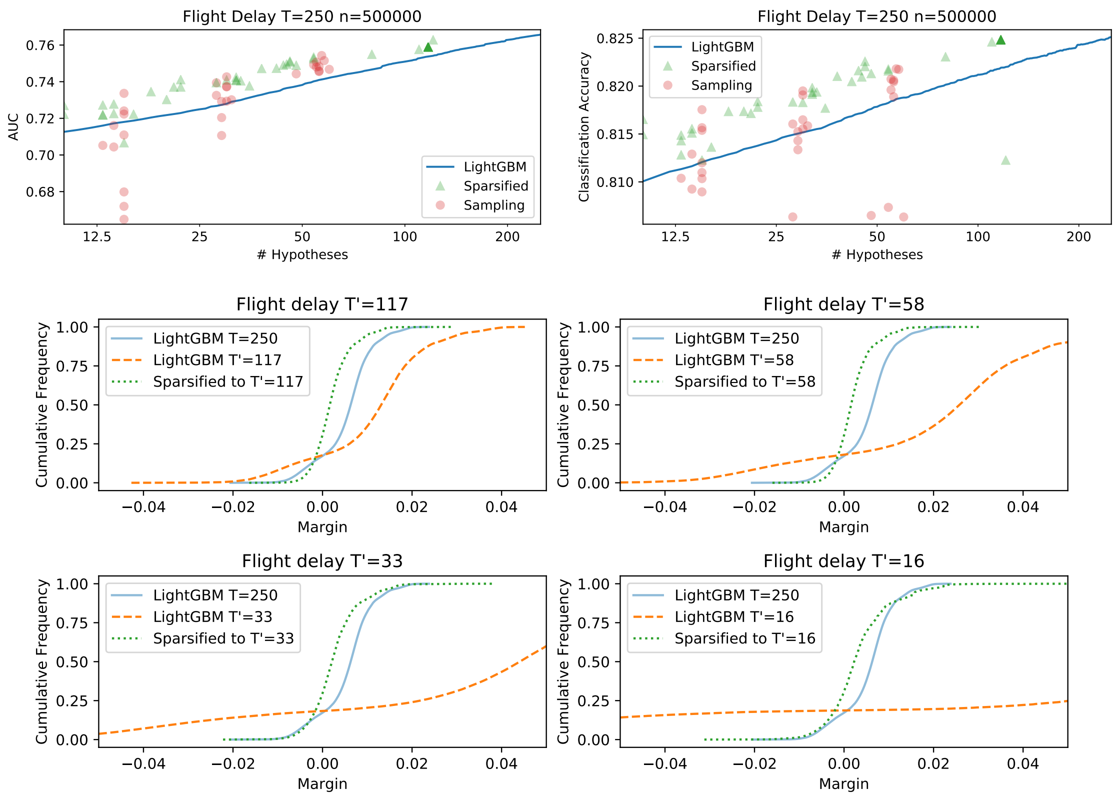


Figure 6: Similar to previous plot, but for Flight Delay with $T = 250$ hypotheses and 500000 training and test points.

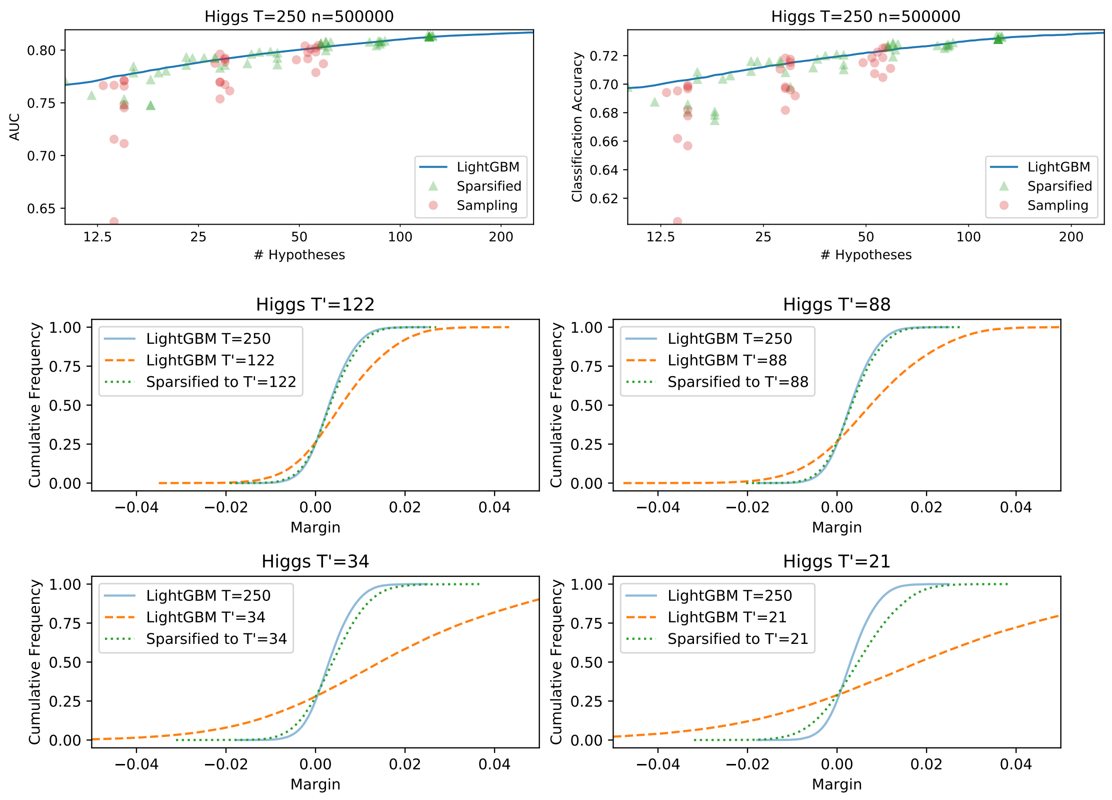


Figure 7: Similar to previous plot, but for Higgs with $T = 250$ hypotheses and 500000 training and test points.

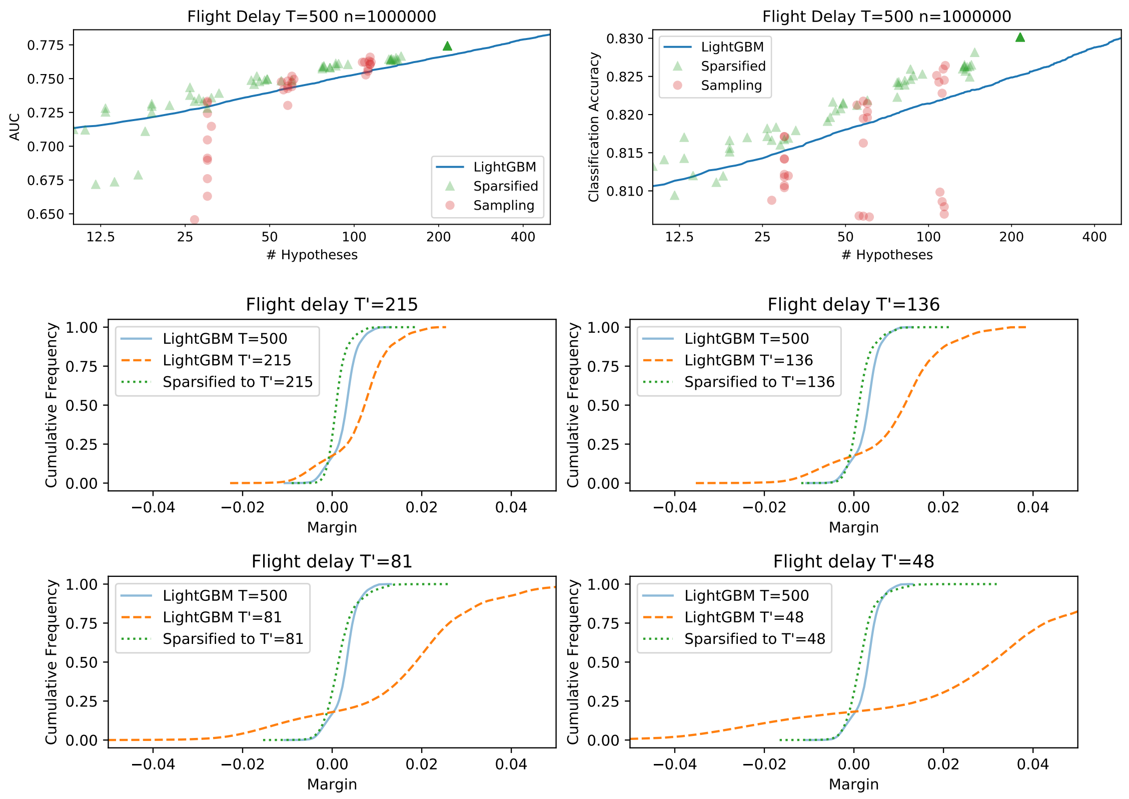


Figure 8: Similar to previous plot, but for Flight Delay with $T = 500$ hypotheses and 1000000 training and test points.

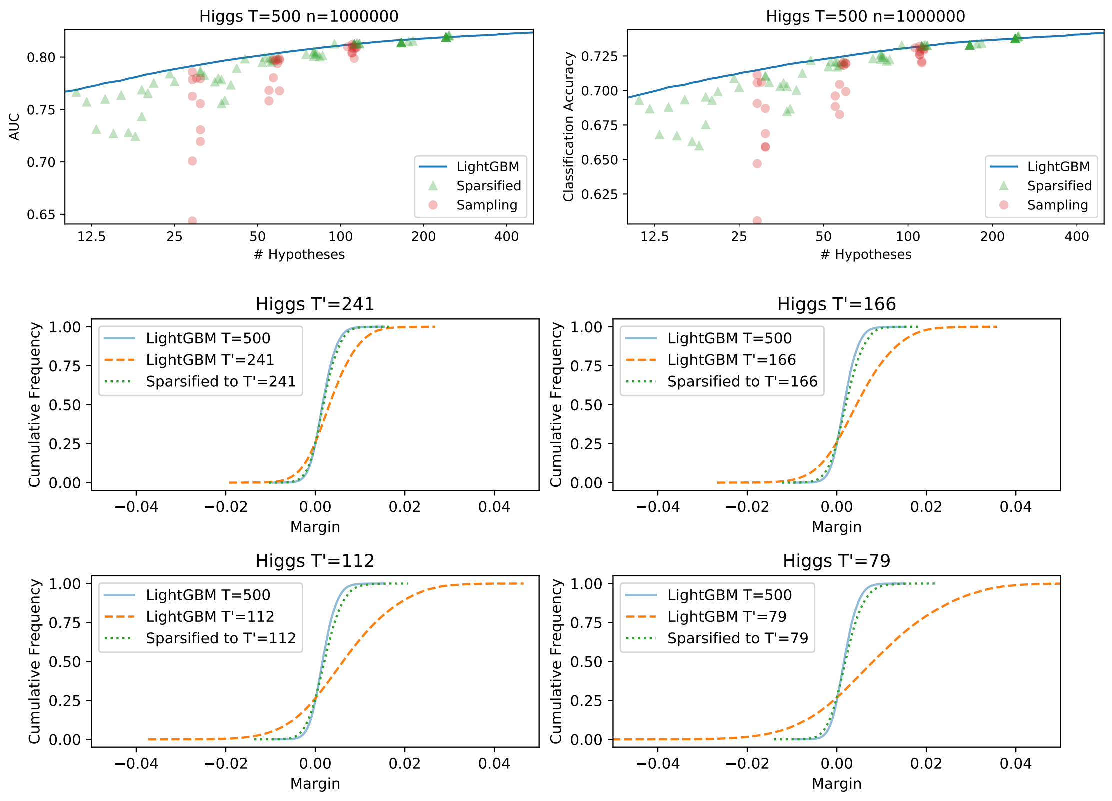


Figure 9: Similar to previous plot, but for Higgs with $T = 500$ hypotheses and 1000000 training and test points.

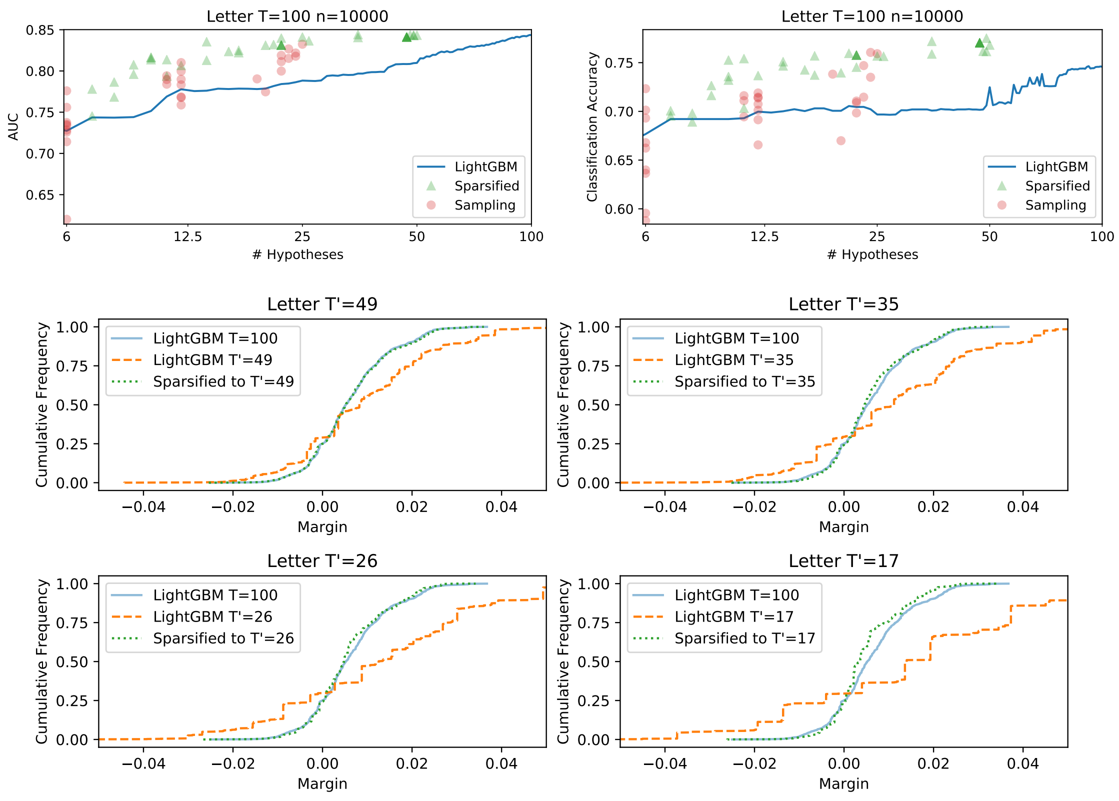


Figure 10: Similar to previous plot, but for Letter with $T = 100$ and $n = 10000$ training and test points. Furthermore, LightGBM used decision stumps (inspired by [Wang et al., 2008]), that is, we choose num_leaves = 2. Quite surprisingly the sparsified classifiers obtain a better classification accuracy than the original classifier. All sparsified classifiers predictions were corrected as described in Appendix A.1. Since the original classifier has better AUC than the sparsified ones, we believe this is caused by a poor bias for the original classifier.

A.2 Minimal Margin Experiments

Figure 11 contain plots of minimal margin during training of AdaBoost, AdaBoostV and SparsiBoost. SparsiBoost obtains similar or better minimal margin on all dataset. Similar to the AUC experiments, the sparsification was repeated 10 times. On Letter, Diabetes and German, SparsiBoost obtains a better minimal margin than AdaBoost and AdaBoostV. On the Ionosphere and Breast dataset it obtains a similar minimal margin to AdaBoost and AdaBoostV. Notice that on those datasets the difference between AdaBoost and AdaBoostV is almost zero, especially if compared with the difference on Letter, Diabetes and German. It might be that AdaBoost and AdaBoostV obtain close to the best minimal margin possible with decision stumps, which would then leave SparsiBoost no room for improvement.

Surprisingly Good minimal margin. We found that in some cases, classifiers with very few hypotheses beat all other classifiers by achieving a minimal margin of 0. On the dataset were this phenomena occurred, the final classifier produced by AdaBoostV has negative minimal margin. Since a minimal margin of 0 is better than any negative minimal margin, the sparsification algorithm improves the performance of the final classifier of AdaBoostV by removing most of its hypotheses. It turns out that the sparsification algorithm finds the trivial hypothesis $f(x) = 0$ which obtains a minimal margin of 0. This can be seen by the definition of the margin of a point x_i:

$$\text{margin}(x_i) = \frac{y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)}{\sum_{t=1}^{T} |\alpha_t|} = \frac{y_i \sum_{t=1}^{T} \alpha_t 0}{\sum_{t=1}^{T} |\alpha_t|} = 0.$$

We found experimentally that the sparsification algorithm is capable of exploiting this as follows. Firstly, we found that AdaBoostV sometimes produces the same decision stump with opposite weights, i.e. $\alpha_l h_l = -\alpha_k h_k$ for indices l, k. When this happens, the sparsification algorithm could then remove all hypotheses except h_l and h_k. The minimal margin of such as classifier is 0 which can be seen as follows:

$$\text{margin}(x_i) = \frac{y_i(\alpha_k h_k - \alpha_k h_k)}{|\alpha_k| + |\alpha_k|} = 0$$

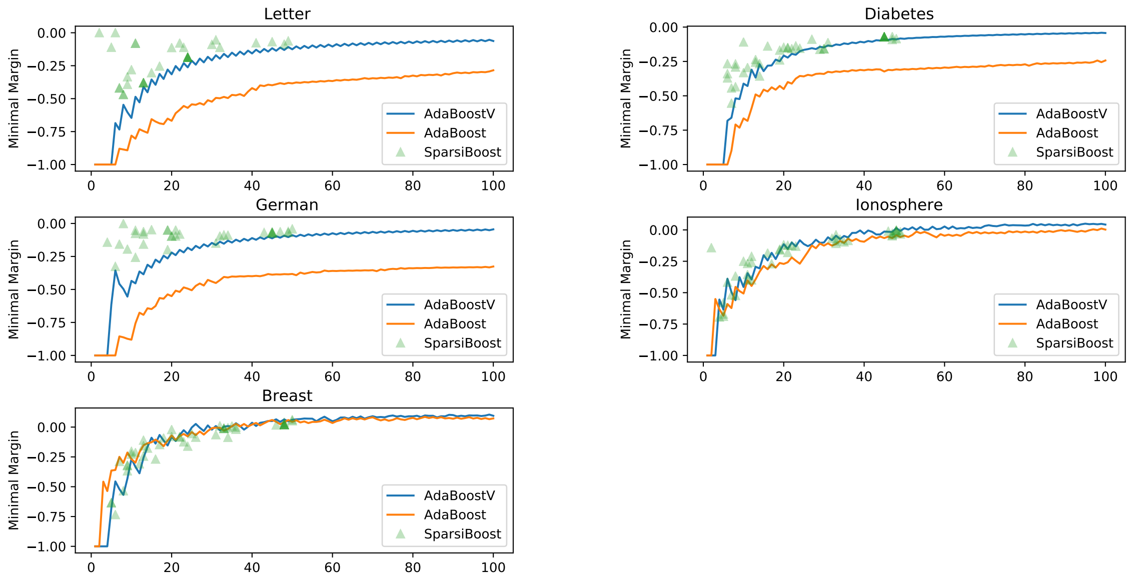


Figure 11: Plot of minimal margin during training of AdaBoost, AdaBoostV and SparsiBoost.

4.2 [2] Margin-Based Generalization Lower Bounds for Boosted Classifiers

Margin-Based Generalization Lower Bounds for Boosted Classifiers

Allan Grønlund [*] Lior Kamma [†] Kasper Green Larsen [‡]

Alexander Mathiasen [§] Jelani Nelson [¶]

Abstract

Boosting is one of the most successful ideas in machine learning. The most well-accepted explanations for the low generalization error of boosting algorithms such as AdaBoost stem from margin theory. The study of margins in the context of boosting algorithms was initiated by Schapire, Freund, Bartlett and Lee (1998) and has inspired numerous boosting algorithms and generalization bounds. To date, the strongest known generalization (upper bound) is the kth margin bound of Gao and Zhou (2013). Despite the numerous generalization upper bounds that have been proved over the last two decades, nothing is known about the tightness of these bounds. In this paper, we give the first margin-based lower bounds on the generalization error of boosted classifiers. Our lower bounds nearly match the kth margin bound and thus almost settle the generalization performance of boosted classifiers in terms of margins.

1 Introduction

Boosting algorithms produce highly accurate classifiers by combining several less accurate classifiers and are amongst the most popular learning algorithms, obtaining state-of-the-art performance on several benchmark machine learning tasks [KMF+17, CG16]. The most famous of these boosting algorithm is arguably AdaBoost [FS97]. For binary classification, AdaBoost takes a training set $S = \langle (x_1, y_1), \dots, (x_m, y_m) \rangle$ of m labeled samples as input, with $x_i \in \mathcal{X}$ and labels $y_i \in \{-1, 1\}$. It then produces a classifier f in iterations: in the jth iteration, a base classifier $h_j : \mathcal{X} \to \{-1, 1\}$ is trained on a reweighed version of S that emphasizes data points that f struggles with and this classifier is then added to f. The final classifier is obtained by taking the sign of $f(x) = \sum_j \alpha_j h_j(x)$, where the α_j's are non-negative coefficients carefully chosen by AdaBoost. The base classifiers h_j all come from a *hypothesis set* $\mathcal{H}$, e.g. $\mathcal{H}$ could be a set of small decision trees or similar. As AdaBoost's training progresses, more and more base classifiers are added to f, which in turn causes the training error of f to decrease. If $\mathcal{H}$ is rich enough, AdaBoost will eventually classify all the data points in the training set correctly [FS97].

[*] Computer Science Department. Aarhus University. `jallan@cs.au.dk`.

[†] Computer Science Department. Aarhus University. Supported by a Villum Young Investigator Grant `lior.kamma@cs.au.dk`.

[‡] Computer Science Department. Aarhus University. Supported by a Villum Young Investigator Grant and an AUFF Starting Grant. `larsen@cs.au.dk`.

[§] Computer Science Department. Aarhus University. Supported by an AUFF Starting Grant. `alexmath@cs.au.dk`.

[¶] Department of Electronic Engineering and Computer Science. UC Berkeley. `minilek@berkeley.edu`. Supported by NSF CAREER award CCF-1350670, NSF grant IIS-1447471, ONR grant N00014-18-1-2562, ONR DORECG award N00014-17-1-2127, an Alfred P. Sloan Research Fellowship, and a Google Faculty Research Award.

Early experiments with AdaBoost report a surprising generalization phenomenon [SFBL98]. Even after perfectly classifying the entire training set, further iterations keeps improving the test accuracy. This is contrary to what one would expect, as f gets more complicated with more iterations, and thus prone to overfitting. The most prominent explanation for this phenomena is margin theory, introduced by Schapire *et al.* [SFBL98]. The margin of a training point (x_i, y_i) is a number in $[-1,1]$, which can be interpreted, loosely speaking, as the classifier's confidence on that point. Formally, we say that $f(x) = \sum_j \alpha_j h_j(x)$ is a *voting classifier* if $\alpha_j \geq 0$ for all j. Note that one can additionally assume without loss of generality that $\sum_j \alpha_j = 1$ since normalizing each α_i by $\sum_j \alpha_j$ leaves the sign of $f(x_i)$ unchanged. The margin of a point (x_i, y_i) with respect to a voting classifier f is then defined as

$$\operatorname{margin}(x_i) := y_i f(x_i) \quad = \quad y_i \sum_j \alpha_j h_j(x_i)\,.$$

Thus $\operatorname{margin}(x_i) \in [-1,1]$, and if $\operatorname{margin}(x_i) > 0$, then taking the sign of $f(x_i)$ correctly classifies (x_i, y_i). Informally speaking, margin theory guarantees that voting classifiers with large (positive) margins have a smaller generalization error. Experimentally AdaBoost has been found to continue to improve the margins even when training past the point of perfectly classifying the training set. Margin theory may therefore explain the surprising generalization phenomena of AdaBoost. Indeed, the original paper by Schapire *et al.* [SFBL98] that introduced margin theory, proved the following margin-based generalization bound. Let $\mathcal{D}$ be an unknown distribution over $\mathcal{X} \times \{-1,1\}$ and assume that the training data S is obtained by drawing m i.i.d. samples from $\mathcal{D}$. Then with high probability over S it holds that for every margin $\theta \in (0,1]$, *every* voting classifier f satisfies

$$\Pr_{(x,y)\sim\mathcal{D}}[yf(x) \leq 0] \leq \Pr_{(x,y)\sim S}[yf(x) < \theta] + O\left(\sqrt{\frac{\ln|\mathcal{H}|\ln m}{\theta^2 m}}\right). \tag{1}$$

The left-hand side of the equation is the out-of-sample error of f (since $\operatorname{sign}(f(x)) \neq y$ precisely when $yf(x) < 0$). On the right-hand side, we use $(x,y) \sim S$ to denote a uniform random point from S. Hence $\Pr_{(x,y)\sim S}[yf(x) < \theta]$ is the fraction of training points with margin less than θ. The last term is increasing in $|\mathcal{H}|$ and decreasing in θ and m. Here it is assumed $\mathcal{H}$ is finite. A similar bound can be proved for infinite $\mathcal{H}$ by replacing $|\mathcal{H}|\lg m$ by $d\lg^2 m$, where d is the VC-dimension of $\mathcal{H}$. This holds for all the generalization bounds below as well. The generalization bound thus shows that f has low out-of-sample error if it attains large margins on most training points. This fits well with the observed behaviour of AdaBoost in practice.

The generalization bound above holds for every voting classifier f, i.e. regardless of how f was obtained. Hence a natural goal is to design boosting algorithms that produce voting classifiers with large margins on many points. This has been the focus of a long line of research and has resulted in numerous algorithms with various margin guarantees, see e.g. [GS98, Bre99, BDST00, RW02, RW05, GLM19]. One of the most well-known of these is Breimann's ArcGV [Bre99]. ArcGV produces a voting classifier maximizing the *minimal* margin, i.e. it produces a classifier f for which $\min_{(x,y)\in S} yf(x)$ is as large as possible. Breimann complemented the algorithm with a generalization bound stating that with high probability over the sample S, it holds that every voting classifier f satisfies:

$$\Pr_{(x,y)\sim\mathcal{D}}[yf(x) \leq 0] \leq O\left(\frac{\ln|\mathcal{H}|\ln m}{\hat{\theta}^2 m}\right), \tag{2}$$

where $\hat{\theta} = \min_{(x,y)\in S} yf(x)$ is the minimal margin over all training examples. Notice that if one chooses θ as the minimal margin in the generalization bound (1) of Schapire *et al.* [SFBL98], then the term $\Pr_{(x,y)\sim S}[yf(x) < \theta]$ becomes 0 and one obtains the bound

$$\Pr_{(x,y)\sim\mathcal{D}}[yf(x) \le 0] \le O\left(\sqrt{\frac{\ln|\mathcal{H}|\ln m}{\hat{\theta}^2 m}}\right),$$

which is weaker than Breimann's bound and motivated his focus on maximizing the minimal margin. Minimal margin is however quite sensitive to outliers and work by Gao and Zhou [GZ13] proved a generalization bound which provides an interpolation between (1) and (2). Their bound is known as the kth margin bound, and states that with high probability over the sample S, it holds for every margin $\theta \in (0,1]$ and every voting classifier f that:

$$\Pr_{(x,y)\sim\mathcal{D}}[yf(x) < 0] \le \Pr_{(x,y)\sim S}[yf(x) < \theta] + O\left(\frac{\ln|\mathcal{H}|\ln m}{\theta^2 m} + \sqrt{\Pr_{(x,y)\sim S}[yf(x) < \theta]\frac{\ln|\mathcal{H}|\ln m}{\theta^2 m}}\right).$$

The kth margin bound remains the strongest margin-based generalization bound to date (see Section 1.2 for further details). The kth margin bound recovers Breimann's minimal margin bound by choosing θ as the minimal margin (making $\Pr_{(x,y)\sim S}[yf(x) < \theta] = 0$), and it is always at most the same as the bound (1) by Schapire *et al.* As with previous generalization bounds, it suggests that boosting algorithms should focus on obtaining a large margin on as large a fraction of training points as possible.

Despite the decades of progress on generalization *upper* bounds, we still do not know how tight these bounds are. That is, we do not have any margin-based generalization *lower* bounds. Generalization lower bounds are not only interesting from a theoretical point of view, but also from an algorithmic point of view: If one has a provably tight generalization bound, then a natural goal is to design a boosting algorithm minimizing a loss function that is equal to this generalization bound. This approach makes most sense with a matching lower bound as the algorithm might otherwise minimize a sub-optimal loss function. Furthermore, a lower bound may also inspire researchers to look for other parameters than margins when explaining the generalization performance of voting classifiers. Such new parameters may even prove useful in designing new algorithms, with even better generalization performance in practice.

1.1 Our Results

In this paper we prove the first margin-based generalization lower bounds for voting classifiers. Our lower bounds almost match the kth margin bound and thus essentially settles the generalization performance of voting classifiers in terms of margins.

To present our main theorems, we first introduce some notation. For a set $\mathcal{X}$ and hypothesis set $\mathcal{H}$, let $C(\mathcal{H})$ denote the family of all voting classifiers over $\mathcal{H}$, i.e. $C(\mathcal{H})$ contains all functions $f : \mathcal{X} \to [-1,1]$ that can be written as $f(x) = \sum_{h\in\mathcal{H}} \alpha_h h(x)$ such that $\alpha_h \ge 0$ for all h and $\sum_h \alpha_h = 1$. For a (randomized) learning algorithm $\mathcal{A}$ and a sample S of m points, let $f_{\mathcal{A},S}$ denote the (possibly random) voting classifier produced by $\mathcal{A}$ when given the sample S as input. With this notation, our first main theorem is the following:

Theorem 1. *For every large enough integer N, every $\theta \in (1/N, 1/40)$ and every $\tau \in [0, 49/100]$ there exist a set $\mathcal{X}$ and a hypothesis set $\mathcal{H}$ over $\mathcal{X}$, such that $\ln|\mathcal{H}| = \Theta(\ln N)$ and for every $m = \Omega\left(\theta^{-2}\ln|\mathcal{H}|\right)$ and for every (randomized) learning algorithm $\mathcal{A}$, there exist a distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1,1\}$ and a voting classifier $f \in C(\mathcal{H})$ such that with probability at least $1/100$ over the choice of samples $S \sim \mathcal{D}^m$ and the random choices of $\mathcal{A}$*

1. $\Pr_{(x,y)\sim S}[yf(x) < \theta] \le \tau$*; and*

2. $\Pr_{(x,y)\sim \mathcal{D}}[yf_{\mathcal{A},S}(x) < 0] \ge \tau + \Omega\left(\frac{\ln|\mathcal{H}|}{m\theta^2} + \sqrt{\tau \cdot \frac{\ln|\mathcal{H}|}{m\theta^2}}\right).$

Theorem 1 states that for any algorithm $\mathcal{A}$, there is a distribution $\mathcal{D}$ for which the out-of-sample error of the voting classifier produced by $\mathcal{A}$ is at least that in the second point of the theorem. At the same time, one can find a voting classifier f obtaining a margin of at least θ on at least a $1-\tau$ fraction of the sample points. Our proof of Theorem 1 not only shows that such a classifier exists, but also provides an algorithm that constructs such a classifier. Loosely speaking, the first part of the theorem reflects on the nature of the distribution $\mathcal{D}$ and the hypothesis set $\mathcal{H}$. Intuitively it means that the distribution is not too hard and the hypothesis set is rich enough, so that it is possible to construct a voting classifier with good empirical margins. Clearly, we cannot hope to prove that the algorithm $\mathcal{A}$ constructs a voting classifier that has a margin of at least θ on a $1-\tau$ fraction of the sample set, since we make no assumptions on the algorithm. For example, if the constant hypothesis h_1 that always outputs 1 is in $\mathcal{H}$, then $\mathcal{A}$ could be the algorithm that simply outputs h_1. The interpretation is thus: $\mathcal{D}$ and $\mathcal{H}$ allow for an algorithm $\mathcal{A}$ to produce a voting classifier f with margin at least θ on a $1-\tau$ fraction of samples. The second part of the theorem thus guarantees that regardless of which voting classifier $\mathcal{A}$ produces, it still has large out-of-sample error. This implies that every algorithm that constructs a voting classifier by maximizing margins on the training examples, must have a large error. Formally, Theorem 1 implies that if $\Pr_{(x,y)\sim S}[yf_{\mathcal{A},S}(x) < \theta] \le \tau$ then

$$\Pr_{(x,y)\sim \mathcal{D}}[yf_{\mathcal{A},S}(x) < 0] \ge \Pr_{(x,y)\sim S}[yf_{\mathcal{A},S}(x) < \theta] + \Omega\left(\frac{\ln|\mathcal{H}|}{m\theta^2} + \sqrt{\tau \cdot \frac{\ln|\mathcal{H}|}{m\theta^2}}\right).$$

The first part of the theorem ensures that the condition is not void. That is, there exists an algorithm $\mathcal{A}$ for which $\Pr_{(x,y)\sim S}[yf_{\mathcal{A},S}(x) < \theta] \le \tau$. Comparing Theorem 1 to the kth margin bound, we see that the parameter τ corresponds to $\Pr_{(x,y)\sim S}[yf(x) < \theta]$. The magnitude of the out-of-sample error in the second point in the theorem thus matches that of the kth margin bound, except for a factor $\ln m$ in the first term inside the $\Omega(\cdot)$ and a $\sqrt{\ln m}$ factor in the second term. If we consider the range of parameters $\theta, \tau, \ln|\mathcal{H}|$ and m for which the lower bound applies, then these ranges are almost as tight as possible. For τ, note that the theorem cannot generally be true for $\tau > 1/2$, as the algorithm $\mathcal{A}$ that outputs a uniform random choice of hypothesis among h_1 and h_{-1} (the constant hypothesis outputting -1), gives a (random) voting classifier $f_{\mathcal{A},S}$ with an expected out-of-sample error of $1/2$. This is less than the second point of the theorem would state if it was true for $\tau > 1/2$. For $\ln|\mathcal{H}|$, observe that our theorem holds for arbitrarily large values of $|\mathcal{H}|$. That is, the integer N can be as large as desired, making $\ln|\mathcal{H}| = \Theta(\ln N)$ as large as desired. Finally, for the constraint on m, notice again that the theorem simply cannot be true for smaller values of m as then the term $\ln|\mathcal{H}|/(m\theta^2)$ exceeds 1.

Our second main result gets even closer to the kth margin bound:

Theorem 2. *For every large enough integer N, every $\theta \in (1/N, 1/40)$, $\tau \in [0, 49/100]$ and every $(\theta^{-2}\ln N)^{1+\Omega(1)} \leq m \leq 2^{N^{O(1)}}$, there exist a set $\mathcal{X}$, a hypothesis set $\mathcal{H}$ over $\mathcal{X}$ and a distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1,1\}$ such that $\ln|\mathcal{H}| = \Theta(\ln N)$ and with probability at least $1/100$ over the choice of samples $S \sim \mathcal{D}^m$ there exists a voting classifier $f_S \in C(\mathcal{H})$ such that*

1. $\Pr_{(x,y)\sim S}[yf_S(x) < \theta] \leq \tau$*; and*

2. $\Pr_{(x,y)\sim \mathcal{D}}[yf_S(x) < 0] \geq \tau + \Omega\left(\frac{\ln|\mathcal{H}|\ln m}{m\theta^2} + \sqrt{\tau \cdot \frac{\ln|\mathcal{H}|}{m\theta^2}}\right).$

Observe that the second point of Theorem 2 has an additional $\ln m$ factor on the first term in $\Omega(\cdot)$ compared to Theorem 1. It is thus only off from the kth margin bound by a $\sqrt{\ln m}$ factor in the second term and hence completely matches the kth margin bound for small values of τ. To obtain this strengthening, we replaced the guarantee in Theorem 1 saying that *all* algorithms $\mathcal{A}$ have such a large out-of-sample error. Theorem 2 demonstrates the existence of a voting classifier f_S (that is chosen as a function of the sample S) that simultaneously achieves a margin of at least θ on a $1-\tau$ fraction of the sample points, and yet has out-of-sample error at least that in point 2. Since the kth margin bound holds with high probability *for all* voting classifiers, Theorem 2 rules out any strengthening of the kth margin bound, except for possibly a $\sqrt{\ln m}$ factor on the second additive term. Finally, we mention that both our lower bounds are proved for a finite hypothesis set $\mathcal{H}$. This only makes the lower bounds stronger than if we proved it for an infinite $\mathcal{H}$ with bounded VC-dimension, since the VC-dimension of a finite $\mathcal{H}$, is no more than $\lg|\mathcal{H}|$.

1.2 Related Work

We mentioned above that the kth margin bound is the strongest margin-based generalization bound to date. Technically speaking, it is incomparable to the so-called *emargin* bound by Wang *et al.* [WSJ+11]. The kth margin bound by Gao and Zhou [GZ13], the minimum margin bound by Breimann [Bre99] and the bound by Schapire *et al.* [SFBL98] all have the form $\Pr_{(x,y)\sim\mathcal{D}}[yf(x) < 0] \leq \Pr_{(x,y)\sim S}[yf(x) < \theta] + \Gamma(\theta, m, |\mathcal{H}|, \Pr_{(x,y)\sim S}[yf(x) < \theta])$ for some function Γ. The emargin bound has a different (and quite involved) form, making it harder to interpret and compute. We will not discuss it in further detail here and just remark that our results show that for generalization bounds of the form studied in most previous work [SFBL98, Bre99, GZ13], one cannot hope for much stronger upper bounds than the kth margin bound.

2 Proof Overview

The main argument that lies in the heart of both proofs is a probabilistic method argument. Let $\mathcal{X}$ be a set of size u. With every labeling $\ell \in \{-1,1\}^u$ we associate a distribution $\mathcal{D}_\ell$ over $\mathcal{X} \times \{-1,1\}$. We then show that with some positive probability if we sample $\ell \in \{-1,1\}^u$, $\mathcal{D}_\ell$ satisfies the requirements of Theorem 2. We thus conclude the existence of a suitable distribution. We next give a more detailed high-level description of the proof for Theorem 2.

Constructing a Family of Distributions. We start by first describing the construction of $\mathcal{D}_\ell$ for $\ell \in \{-1,1\}^u$. Our construction combines previously studied distribution patterns in a subtle manner.

Ehrenfeucht *et al.* [EHKV89] observed that if a distribution $\mathcal{D}$ assigns each point in $\mathcal{X}$ a fixed (yet unknown) label, then, loosely speaking, every classifier f, that is constructed using only information supplied by a sample S, cannot do better than random guessing the labels for the points in $\mathcal{X} \setminus S$. Intuitively, consider a uniform distribution $\mathcal{D}_\ell$ over $\mathcal{X}$. If we assume, for example, that $|\mathcal{X}| \geq 10m$, then with very high probability over a sample S of m points, many elements of $\mathcal{X}$ are not in S. Moreover, assume that $\mathcal{D}_\ell$ associates every $x \in \mathcal{X}$ with a unique "correct" label $\ell(x)$. Consider some (perhaps random) learning algorithm $\mathcal{A}$, and let $f_{\mathcal{A},S}$ be the classifier it produces given a sample S as input. If ℓ is chosen randomly, then, loosely speaking, for every point x not in the sample, $f_{\mathcal{A},S}(x)$ and $\ell(x)$ are independent, and thus $\mathcal{A}$ returns the wrong label with probability $1/2$. In turn, this implies that there exists a labeling ℓ such that $\mathcal{A}$ is wrong on a constant fraction of $\mathcal{X}$ when receiving a sample $S \sim \mathcal{D}_\ell^m$. While the argument above can in fact be used to prove an arbitrarily large generalization error, it requires $|\mathcal{X}|$ to be large, and specifically to increase with m. This conflicts with the first point in Theorem 2, that is, we have to argue that a voting classifier f with good margins exist for the sample S. If S consists of m distinct points, and each point in $\mathcal{X}$ can have an arbitrary label, then intuitively $\mathcal{H}$ needs to be very large to ensure the existence of f. In order to overcome this difficulty, we set $\mathcal{D}_\ell$ to assign very high probability to one designated point in $\mathcal{X}$, and the rest of the probability mass is then equally distributed between all other points. The argument above still applies for the subset of small-probability points. More precisely, if $\mathcal{D}_\ell$ assigns all but one point in $\mathcal{X}$ probability $\frac{1}{10m}$, then the expected generalization error (over the choice of ℓ) is still $\Omega\left(\frac{1}{10m}|\mathcal{X}|\right)$. It remains to determine how large can we set $|\mathcal{X}|$. In the notations of the theorem, in order for a hypothesis set $\mathcal{H}$ to satisfy $\ln|\mathcal{H}| = \Theta(\ln N)$, and at the same time, have an $f \in C(\mathcal{H})$ obtaining margins of θ on most points in a sample, our proof (and specifically Lemma 3, described hereafter) requires $\mathcal{X}$ to be not significantly larger than $\frac{\ln N}{\theta^2}$, and therefore the generalization error we get is $\Omega\left(\frac{\ln|\mathcal{H}|}{\theta^2 m}\right)$.

Anthony and Bartlett [AB09, Chapter 5] additionally observed that for a distribution $\mathcal{D}$ that assigns each point in $\mathcal{X}$ a random label, if S does not sample a point x enough times, any classifier f, that is constructed using only information supplied by S, cannot determine with good probability the Bayes label of x, that is, the label of x that minimizes the error probability. Intuitively, consider once more a distribution $\mathcal{D}_\ell$ that is uniform over $\mathcal{X}$. However, instead of associating every point $x \in \mathcal{X}$ with one correct label $\ell(x)$, $\mathcal{D}_\ell$ is now only slightly biased towards ℓ. That is, given that x is sampled, the label in the sample point is $\ell(x)$ with probability that is a little larger than $1/2$, say $(1+\alpha)/2$ for some small $\alpha \in (0,1)$. Note that every classifier f has an error probability of at least $(1-\alpha)/2$ on every given point in $\mathcal{X}$. Consider once again a learning algorithm $\mathcal{A}$ and the voting classifier $f_{\mathcal{A},S}$ it constructs. Loosely speaking, if S does not sample a point x enough times, then with good probability $f_{\mathcal{A},S}(x) \neq \ell(x)$. More formally, in order to correctly assign the Bayes label of x, an algorithm must see $\Omega(\alpha^{-2})$ samples of x. Therefore if we set the bias α to be $\sqrt{|\mathcal{X}|/(10m)}$, then with high probability the algorithm does not see a constant fraction of $\mathcal{X}$ enough times to correctly assign their label. In turn, this implies an expected generalization error of $(1-\alpha)/2 + \Omega(\sqrt{|\mathcal{X}|/m})$, where the expectation is over the choice of ℓ. By once again letting $|\mathcal{X}| = \frac{\ln N}{\theta^2}$ we conclude that there exists a labeling ℓ such that for $S \sim \mathcal{D}_\ell^m$, the expected generalization error of $f_{\mathcal{A},S}$ is $\frac{1-\alpha}{2} + \Omega\left(\sqrt{\frac{\ln|\mathcal{H}|}{\theta^2 m}}\right)$. This expression is almost the second term inside the Ω-notation in the theorem statement, though slightly larger. We note, however, for large values of m, the in-sample error is arbitrarily close to $1/2$. One challenge is therefore to reduce the in-sample-error, and moreover guarantee that we can find a voting classifier f where the $(m\tau)$'th smallest margin for f is at least θ, where τ, θ are the parameters provided by the theorem statement.

To this end, our proof subtly weaves the two ideas described above and constructs a family of distributions $\{\mathcal{D}_\ell\}_{\ell \in \{-1,1\}^u}$. Informally, we partition $\mathcal{X}$ into two disjoint sets, and conditioned on the sample point $x \in \mathcal{X}$ belonging to each of the subsets, $\mathcal{D}_\ell$ is defined similarly to be one of the two distribution patterns defined above. The main difficulty lies in delicately balancing all ingredients and ensuring that we can find an f with margins of at least θ on all but τm of the sample points, while still enforcing a large generalization error. Our proof refines the proof given by Ehrenfeucht *et al.* and Anthony and Bartlett and shows that there exists a labeling ℓ such that $f_{\mathcal{A},S}$ has large generalization error with respect to $\mathcal{D}_\ell$ (with probability at least $1/100$ over the randomness of $\mathcal{A}, S$).

Small yet Rich Hypothesis Sets. The technical crux in our proofs is the construction of an appropriate hypothesis set. Loosely speaking, the size of $\mathcal{H}$ has to be small, and most importantly, independent of the size m of the sample set. On the other hand, the set of voting classifiers $C(\mathcal{H})$ is required to be rich enough to, intuitively, contain a classifier that with good probability has good in-sample margins for a sample $S \sim \mathcal{D}_\ell^m$ with a large fraction of labelings $\ell \in \{-1,1\}^u$. Our main technical lemma presents a distribution μ over small hypothesis sets $\mathcal{H} \subset \mathcal{X} \to \{-1,1\}$ such that for every *sparse* $\ell \in \{-1,1\}^u$, that is $\ell_i = -1$ for a small number of entries $i \in [u]$, with high probability over $\mathcal{H} \sim \mu$, there exists some voting classifier $f \in C(\mathcal{H})$ that has minimum margin θ with ℓ over the entire set $\mathcal{X}$. In fact, the size of the hypothesis set does not depend on the size of $\mathcal{X}$, but only on the sparsity parameter d. More formally, we show the following.

Lemma 3. *For every $\theta \in (0,1/40)$, $\delta \in (0,1)$ and integers $d \le u$, there exists a distribution $\mu = \mu(u,d,\theta,\delta)$ over hypothesis sets $\mathcal{H} \subset \mathcal{X} \to \{-1,1\}$, where $\mathcal{X}$ is a set of size u, such that the following holds.*

1. *For all $\mathcal{H} \in \operatorname{supp}(\mu)$, we have $|\mathcal{H}| = N$; and*
2. *For every labeling $\ell \in \{-1,+1\}^u$, if no more than d points $x \in \mathcal{X}$ satisfy $\ell(x) = -1$, then*

$$\Pr_{\mathcal{H}\sim\mu}[\exists f \in \mathcal{C}(\mathcal{H}) : \forall x \in \mathcal{X}.\ \ell(x)f(x) \ge \theta] \ge 1-\delta\,,$$

where $N = \Theta\left(\theta^{-2}\ln u \ln(\theta^{-2}\ln u\delta^{-1})e^{\Theta(\theta^2 d)}\right)$

To show the existence of a good voting classifier in $C(\mathcal{H})$ our proof actually employs a slight variant of the celebrated AdaBoost algorithm, and shows that with high probability (over the choice of the random hypothesis set $\mathcal{H}$), the voting classifier constructed by this algorithm attains minimum margin at least θ over the entire set $\mathcal{X}$.

Existential Lower Bound. Our proof of Theorem 2 uses many of the same ideas as the proof of Theorem 1. The difference between the generalization lower bound (second point) in Theorem 1 and 2 is an $\ln m$ factor in the first term inside the $\Omega(\cdot)$ notation. That is, Theorem 2 has an $\Omega(\frac{\ln|\mathcal{H}|\ln m}{\theta^2 m})$ where Theorem 1 has an $\Omega(\frac{\ln|\mathcal{H}|}{\theta^2 m})$. This term originated from having $\ln|\mathcal{H}|/\theta^2$ points with a probability mass of $1/10m$ in $\mathcal{D}_\ell$ and one point having the remaining probability mass. In the proof of Theorem 2, we first exploit that we are proving an existential lower bound by assigning all points the same label 1. That is, our hard distribution $\mathcal{D}$ assigns all points the label 1 (ignoring the second half of the distribution with the random and slightly biased labels). Since we are not proving a lower bound for every algorithm, this will not cause problems. We then change $|\mathcal{X}|$ to about $m/\ln m$ and assign each point the same probability mass $\ln m/m$ in distribution $\mathcal{D}$. The key observation is that on a random sample S of m points, by a coupon-collector argument, there will still be $m^{\Omega(1)}$ points from $\mathcal{X}$ that were not sampled. From Lemma 3, we can now find a voting classifier f, such that $\operatorname{sign}(f(x))$ is 1 on all points in $x \in S$, and -1 on a set of $d = \ln|\mathcal{H}|/\theta^2$ points in $\mathcal{X} \setminus S$. This means that f has out-of-sample error $\Omega(d \ln m/m) = \Omega(\frac{\ln|\mathcal{H}|\ln m}{\theta^2 m})$ under distribution $\mathcal{D}$ and obtains a margin of θ on all points in the sample S.

As in the proof Theorem 1, we can combine the above distribution $\mathcal{D}$ with the ideas of Anthony and Bartlett to add the terms depending on τ to the lower bound.

3 Margin-Based Generalization Lower Bounds

In this section we prove 2 assuming Lemma 3, whose proof is deferred to Section 4, and we start by describing the outlines of the proofs. To this end fix some integer N, and fix $\theta \in (1/N, 1/40)$. Let u be an integer, and let $\mathcal{X} = \{\xi_1, \dots, \xi_u\}$ be some set with u elements. With every $\ell \in \{-1,1\}^u$ we associate a distribution $\mathcal{D}_\ell$ over $\mathcal{X} \times \{-1,1\}$, and show that with some constant probability over a random choice of ℓ, a voting classifier of interest has a high generalization probability with respect to $\mathcal{D}_\ell$. By a voting classifier of interest we mean one constructed by a learning algorithm in the proof of Theorem 1 and an adversarial classifier in the proof of Theorem 2. We additionally show existence of a hypothesis set $\hat{\mathcal{H}}$ such that with very high (constant) probability over a random choice of $\ell \in \{-1,1\}^u$, $C(\hat{\mathcal{H}})$ contains a voting classifier that attains high margins with ℓ over the entire set $\mathcal{X}$. Finally, we conclude that with positive probability over a random choice of $\ell \in \{-1,1\}^u$ both properties are satisfied, and therefore there exists at least one labeling ℓ that satisfies both properties.

We start by constructing the family $\{\mathcal{D}_\ell\}_{\ell \in \{-1,1\}^u}$ of distributions over $\mathcal{X} \times \{-1,1\}$. To this end, let $d \le u$ be some constant to be fixed later, and let $\ell \in \{-1,1\}^u$. We define $\mathcal{D}_\ell$ separately for the first $u-d$ points and the last d points of $\mathcal{X}$. Intuitively, every point in $\{\xi_i\}_{i\in[u-d]}$ has a fixed label determined by ℓ, however all points but one have a very small probability of being sampled according to $\mathcal{D}_\ell$. Every point in $\{\xi_i\}_{i\in[u-d,u]}$, on the other hand, has an equal probability of being sampled, however its label is not fixed by ℓ rather than slightly biased towards ℓ. Formally, let $\alpha, \beta, \varepsilon \in [0,1]$ be constants to be fixed later. We construct $\mathcal{D}_\ell$ using the ideas described earlier in Section 2, by sewing them together over two parts of the set $\mathcal{X}$. We assign probability $1-\beta$ to $\{\xi_i\}_{i\in[u-d]}$ and β to $\{\xi_i\}_{i\in[u-d+1,u]}$. That is, for $(x,y) \sim \mathcal{D}_\ell$, the probability that $x \in \{\xi_i\}_{i\in[u-d]}$ is $1-\beta$. Next, conditioned on $x \in \{\xi_i\}_{i\in[u-d]}$, (ξ_1, ℓ_1) is assigned high probability $(1-\varepsilon)$ and the rest of the measure is distributed uniformly over $\{(\xi_i,\ell_i)\}_{i\in[2,u-d]}$. That is

$$\Pr_{\mathcal{D}_\ell}[(\xi_1,\ell_1)] = (1-\beta)(1-\varepsilon)\ , \ and\ \forall j \in [2,u-d].\ \Pr_{\mathcal{D}_\ell}[(\xi_j,\ell_j)] = \frac{(1-\beta)\varepsilon}{u-d-1}\ .$$

Finally, conditioned on $x \in \{\xi_i\}_{i\in[u-d+1,u]}$, x distributes uniformly over $\{\xi_i\}_{i\in[u-d+1,u]}$, and conditioned on $x = \xi_i$, we have $y = \ell_i$ with probability $\frac{1+\alpha}{2}$. That is

$$\forall j \in [u-d+1,u].\ \Pr_{\mathcal{D}_\ell}[(\xi_j,\ell_j)] = \frac{(1+\alpha)\beta}{2d}\ , and\ \Pr_{\mathcal{D}_\ell}[(\xi_j,-\ell_j)] = \frac{(1-\alpha)\beta}{2d}\ .$$

In order to give a lower bound on the generalization error for some classifier f of interest, we define new random variables such that their sum is upper bounded by $\Pr_{(x,y)\sim\mathcal{D}_\ell}[yf(x) < 0]$, and give a lower bound on that sum. To this end, for every $\ell \in \{-1,1\}^u$ and $f : \mathcal{X} \to \mathbb{R}$, denote

$$\Psi_1(\ell,f) = \frac{(1-\beta)\varepsilon}{u-d-1} \sum_{i\in[2,u-d]} \mathbb{1}_{\ell_i f(\xi_i)<0} \quad ; \quad \Psi_2(\ell,f) = \frac{\alpha\beta}{d} \sum_{i\in[u-d+1,u]} \mathbb{1}_{\ell_i f(\xi_i)<0}\ . \qquad (3)$$

When f, ℓ are clear from the context we shall simply denote Ψ_1, Ψ_2. We show next that indeed proving a lower bound on $\Psi_1 + \Psi_2$ implies a lower bound on the generalization error.

Claim 4. *For every ℓ, f we have* $\Pr_{(x,y)\sim\mathcal{D}_\ell}[yf(x) < 0] \geq \frac{\beta(1-\alpha)}{2} + \Psi_1 + \Psi_2$.

Proof. We first observe that

$$\begin{aligned}\Pr_{(x,y)\sim\mathcal{D}_\ell}[yf(x) < 0] &= \mathbb{E}_{(x,y)\sim\mathcal{D}_\ell}[\mathbb{1}_{yf(x)<0}] \\ &= \sum_{i\in[u-d], y\in\{-1,1\}} \mathbb{1}_{yf(\xi_i)<0} \Pr_{\mathcal{D}_\ell}[(\xi_i, y)] + \sum_{i\in[u-d+1,u], y\in\{-1,1\}} \mathbb{1}_{yf(\xi_i)<0} \Pr_{\mathcal{D}_\ell}[(\xi_i, y)]\end{aligned} \tag{4}$$

For every $i \in [u-d]$ and $y \in \{-1,1\}$, if $y \neq \ell_i$ then $\Pr_{\mathcal{D}_\ell}[(\xi_i, y)] = 0$. Moreover, if $i \geq 2$ and $y = \ell_i$ then $\Pr_{\mathcal{D}_\ell}[(\xi_i, y)] = \frac{(1-\beta)\varepsilon}{u-d-1}$. Therefore

$$\sum_{i\in[u-d], y\in\{-1,1\}} \mathbb{1}_{yf(\xi_i)<0} \Pr_{\mathcal{D}_\ell}[(\xi_i, y)] \geq \frac{(1-\beta)\varepsilon}{u-d-1} \sum_{i\in[2,u-d]} \mathbb{1}_{yf(\xi_i)<0} = \Psi_1 \,. \tag{5}$$

Next, for every $i \in [u-d+1, u]$ we have that

$$\begin{aligned}\sum_{y\in\{-1,1\}} \mathbb{1}_{yf(\xi_i)<0} \Pr_{\mathcal{D}_\ell}[(\xi_i, y)] &= \mathbb{1}_{\ell_i f(\xi_i)<0} \Pr_{\mathcal{D}_\ell}[(\xi_i, \ell_i)] + \mathbb{1}_{\ell_i f(\xi_i)>0} \Pr_{\mathcal{D}_\ell}[(\xi_i, -\ell_i)] \\ &= \frac{(1-\alpha)\beta}{2d} + \mathbb{1}_{\ell_i f(\xi_i)<0} \frac{\alpha\beta}{d} \,,\end{aligned}$$

and therefore

$$\sum_{i\in[u-d+1,u], y\in\{-1,1\}} \mathbb{1}_{yf(\xi_i)<0} \Pr_{\mathcal{D}_\ell}[(\xi_i, y)] = \frac{(1-\alpha)\beta}{2} + \frac{\alpha\beta}{d} \sum_{i\in[u-d+1,u]} \mathbb{1}_{\ell_i f(\xi_i)<0} \,. \tag{6}$$

Plugging (5) and (6) into (4) we conclude the claim. □

To prove existence of a "rich" yet small enough hypothesis set $\hat{\mathcal{H}}$ we apply Lemma 3 together with Yao's minimax principle. In order to ensure that the hypothesis sets constructed using Lemma 3 is small enough, and specifically has size $N^{O(1)}$, we need to focus our attention on sparse labelings $\ell \in \{-1,1\}^u$ only. That is, the labelings cannot contain more than $\Theta\left(\frac{\ln N}{\theta^2}\right)$ entries equal to -1. More formally, we define a set of labelings of interest $\mathcal{L}(u,d)$ as the set of all labelings $\ell \in \{-1,1\}^u$ such that the restriction to the first $u-d$ entries is d-sparse. That is

$$\mathcal{L}(u,d) := \{\ell \in \{-1,1\}^u : |\{i \in [u-d] : \ell_i = -1\}| \leq d\} \,. \tag{7}$$

We next show that there exists a small enough (with respect to N) hypothesis set $\hat{\mathcal{H}}$ that is rich enough. That is, with high probability over choosing ℓ uniformly at random from $\ell \in_R \mathcal{L}(u,d)$, there exists a voting classifier $f \in C(\hat{\mathcal{H}})$ that attains high minimum margin with ℓ over the entire set $\mathcal{X}$. Note that the following result, similarly to Lemma 3 does not depend on the size of $\mathcal{X}$, but only on the sparsity of the labelings in question.

Claim 5. *If $u \leq 2^{N^{O(1)}}$ and $d \leq \frac{\ln N}{\theta^2}$ then there exists a hypothesis set $\hat{\mathcal{H}}$ satisfying that* $\ln|\hat{\mathcal{H}}| = \Theta(\ln N)$ *and*

$$\Pr_{\ell\in_R\mathcal{L}(u,d)}[\exists f \in \mathcal{C}(\hat{\mathcal{H}}) : \forall i \in [u].\ \ell_i f(\xi_i) \geq \theta] \geq 1 - 1/N \,.$$

Proof. Let $\mu = \mu(u, d, \theta, 1/N)$, be the distribution whose existence is guaranteed in Lemma 3. Then for every labeling $\ell \in \mathcal{L}(u,d)$, with probability at least $1 - 1/N$ over $\mathcal{H} \sim \mu$, there exists a voting classifier $f \in C(\mathcal{H})$ that has minimal margin of θ. That is, for every $i \in [u]$, $\ell_i f(\xi_i) \geq \theta$. By Yao's minimax principle, there exists a hypothesis set $\hat{\mathcal{H}} \in \text{supp}(\mu)$ such that

$$\Pr_{\ell \in_R \mathcal{L}(u,d)} [\exists f \in \mathcal{C}(\hat{\mathcal{H}}) : \forall i \in [u].\, \ell_i f(x_i) \geq \theta] \geq 1 - 1/N\,.$$

Moreover, since $\hat{\mathcal{H}} \in \text{supp}(\mu)$, then $|\hat{\mathcal{H}}| = \Theta\left(\theta^{-2} \ln u \cdot \ln(N\theta^{-2} \ln u) \cdot e^{\Theta(\theta^2 d)}\right)$. Since $\theta \geq 1/N$, since $u \leq 2^{N^{O(1)}}$ and since $d \leq \frac{\ln N}{\theta^2}$ and thus $e^{\theta^2 d} \leq N$ we get that there exists some universal constant $C > 0$ such that $|\hat{\mathcal{H}}| = \Theta(N^C)$, and thus $\ln |\hat{\mathcal{H}}| = \Theta(\ln N)$. □

3.1 Proof Algorithmic Lower Bound

This section is devoted to the proof of Theorem 1. That is, we show that for every algorithm $\mathcal{A}$, there exist some distribution $\mathcal{D} \in \{\mathcal{D}_\ell\}_{\ell \in \{-1,1\}^u}$ and some classifier $\hat{f} \in C(\hat{\mathcal{H}})$ such that with constant probability over $S \sim \mathcal{D}^m$, $\hat{f}$ has large margins on points in S, yet $f_{\mathcal{A},S}$ has large generalization error. To this end we now fix u to be $\frac{2\ln N}{\theta^2}$ and $d = \frac{u}{2} = \frac{\ln N}{\theta^2}$. For these values of u, d we get that $\mathcal{L}(u,d)$ is, in fact, the set of all possible labelings, i.e. $\mathcal{L}(u,d) = \{-1,1\}^u$. Next, fix $\mathcal{A}$ to be a (perhaps randomized) learning algorithm. For every m-point sample S and let that $f_{\mathcal{A},S}$ denote the classifier returned by $\mathcal{A}$ when running on sample S.

The main challenge is to show that there exists a labeling $\hat{\ell} \in \{-1,1\}^u$ such that $C(\hat{\mathcal{H}})$ contains a good voting classifier for $\hat{\ell}$ and, in addition, $f_{\mathcal{A},S}$ has a large generalization error with respect to $\mathcal{D}_{\hat{\ell}}$.

Lemma 6. *If $\alpha \leq \sqrt{\frac{u}{40\beta m}}$, then there exists $\hat{\ell} \in \{-1,1\}^u$ such that*

1. *There exists $\hat{f} = \hat{f}_{\hat{\ell}} \in \mathcal{C}(\hat{\mathcal{H}})$ such that for every $i \in [u]$, $\hat{\ell}_i \hat{f}(\xi_i) \geq \theta$; and*
2. *with probability at least $1/25$ over $S \sim \mathcal{D}^m_{\hat{\ell}}$ and the randomness of $\mathcal{A}$ we have*

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}} \left[y f_{\mathcal{A},S}(x) < 0 \right] \geq \frac{(1-\alpha)\beta}{2} + \frac{(1-\beta)\varepsilon}{24} + \frac{\alpha\beta}{24}\,.$$

Before proving the lemma, we first show how it implies Theorem 1

Proof of Theorem 1. Fix some $\tau \in [0, 49/100]$. Assume first that $\tau \leq \frac{u}{300m}$, and let $\varepsilon = \frac{u}{10m}$ and $\beta = \alpha = 0$. Let $\hat{\ell}, \hat{f}$ be as in Lemma 6, then for every sample $S \sim \mathcal{D}^m_{\hat{\ell}}$, $\Pr_{(x,y)\sim S}[y\hat{f}(x) < \theta] = 0 \leq \tau$, and moreover with probability at least $1/25$ over S and the randomness of $\mathcal{A}$

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}} [y f_{\mathcal{A},S}(x) < 0] \geq \frac{(1-\beta)\varepsilon}{24} \geq \tau + \Omega\left(\frac{u}{m}\right) = \tau + \Omega\left(\frac{\ln|\hat{\mathcal{H}}|}{m\theta^2} + \sqrt{\frac{\tau \ln|\hat{\mathcal{H}}|}{m\theta^2}}\right).$$

where the last transition is due to the fact that $u = 2\theta^{-2} \ln N = \Theta(\theta^{-2} \ln|\hat{\mathcal{H}}|)$ and $\tau = O(u/m)$.

Otherwise, assume $\tau > \frac{u}{300m}$, and let $\varepsilon = \frac{u}{10m}$, $\alpha = \sqrt{\frac{u}{2560\tau m}}$ and $\beta = \frac{64\tau}{32-31\alpha}$. Since $\tau \geq \frac{u}{300m}$, then $\alpha \in [0,1]$. Moreover, if $m > Cu$ for large enough but universal constant $C > 0$, then $32 - 31\alpha \geq 64 \cdot \frac{49}{100} \geq 64\tau$, and hence $\beta \in [0,1]$. Moreover, since $\alpha \leq 1$ then $\beta \leq 64\tau$, and therefore $\alpha = \sqrt{\frac{u}{2560\tau m}} \leq \sqrt{\frac{u}{40\beta m}}$. Let therefore $\hat{\ell}, \hat{f}$ be a labeling and a classifier in $C(\hat{\mathcal{H}})$ whose existence is guaranteed in Lemma 6.

Let $\langle(x_1,y_1),\dots,(x_m,y_m)\rangle \sim \mathcal{D}_{\hat\ell}^m$ be a sample of m points drawn independently according to $\mathcal{D}_{\hat\ell}$. For every $j\in[m]$, we have $\mathbb{E}[\mathbb{1}_{y_j\hat f(x_j)<\theta}] = \frac{(1-\alpha)\beta}{2}$. Therefore by Chernoff we get that for large enough N,

$$\Pr_{S\sim\mathcal{D}_{\hat\ell}^m}\left[\Pr_{(x,y)\sim S}\left[y\hat f(x)<\theta\right]\geq\tau\right] = \Pr_{S\sim\mathcal{D}_{\hat\ell}^m}\left[\frac{1}{m}\sum_{j\in[m]}\mathbb{1}_{\hat y_j\hat f(x_j)<\theta}\geq\frac{(1-31\alpha/32)\beta}{2}\right]$$
$$\leq e^{-\Theta(\alpha^2\beta m)}\leq e^{-\Theta(u)}\leq 10^{-3}\,,$$

where the inequality before last is due to the fact that $\alpha^2\beta m = \frac{u\beta}{2560\tau} = \Omega(u)$, since $\beta\geq 2\tau$. Moreover, by Lemma 6 we get that with probability at least $1/25$ over S and $\mathcal{A}$ we get that

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat\ell}}[yf_{\mathcal{A},S}(x)<0]\geq\frac{(1-\alpha)\beta}{2}+\frac{\alpha\beta}{32}=\frac{(1-31\alpha/32)\beta}{2}+\frac{\alpha\beta}{64}=\tau+\Omega\left(\sqrt{\frac{\tau u}{m}}\right)$$
$$\geq\tau+\Omega\left(\frac{\ln|\hat{\mathcal{H}}|}{m\theta^2}+\sqrt{\frac{\tau\ln|\hat{\mathcal{H}}|}{m\theta^2}}\right),$$

where the last transition is due to the fact that $\tau=\Omega(u/m)$. This completes the proof of Theorem 1. □

For the rest of the section we therefore prove Lemma 6. We start by lower bounding the expected value of $\Psi_1+\Psi_2$, where the expectation is over the choice of labeling $\ell\in\{-1,1\}^u$, $S\sim\mathcal{D}_\ell^m$ and the random choices made by $\mathcal{A}$. Intuitively, as points in $\{\xi_2,\dots,\xi_u\}$ are sampled with very small probability, it is very likely that the sample S does not contain many of them, and therefore the algorithm cannot do better than randomly guessing many of the labels. Moreover, if α is small enough, and S does not sample a point in $\{\xi_{u/2+1},\dots,\xi_u\}$ enough times, there is a larger probability that $\mathcal{A}$ does not determine the bias correctly.

Claim 7. *If $\alpha\leq\sqrt{\frac{u}{40\beta m}}$, then $\mathbb{E}_{\ell\in\{-1,1\}^u}\left[\mathbb{E}_{\mathcal{A},S}\left[\Psi_1(\ell,f_{\mathcal{A},S})+\Psi_2(\ell,f_{\mathcal{A},S})\right]\right]\geq\frac{(1-\beta)\varepsilon}{6}+\frac{\alpha\beta}{6}$.*

Proof. To lower bound the expectation, we lower bound the expectations of Ψ_1 and Ψ_2 separately. For every $i\in[2,u-d]\setminus\{1\}$, if $\xi_i\notin S$ then ℓ_i and $f_{\mathcal{A},S}(\xi_i)$ are independent, and therefore $\mathbb{E}_\ell[\mathbb{1}_{\ell_i f_{\mathcal{A},S}(\xi_i)<0}]=\frac{1}{2}$. Let $\mathcal{S}$ be the set of all samples for which $|S\cap\{\xi_2,\dots,\xi_{u-d}\}|\leq\frac{u-d-1}{2}$, then for every $S\in\mathcal{S}$,

$$\mathbb{E}_\ell\left[\sum_{i\in[2,u-d-1]}\mathbb{1}_{\ell_i f_{\mathcal{A},S}(\xi_i)<0}\right]\geq\frac{u-d-1-|S\cap\{\xi_2,\dots,\xi_{u-d}\}|}{2}\geq\frac{u-d-1}{4}\,,$$

As this holds for every $S\in\mathcal{S}$, we conclude that

$$\mathbb{E}_{\mathcal{A},S}\left[\mathbb{E}_\ell\left[\Psi_1(\ell,f_{\mathcal{A},S})\right]\,\middle|\,S\in\mathcal{S}\right]\geq\frac{(1-\beta)\varepsilon}{u-d-1}\cdot\frac{u-d-1}{4}=\frac{(1-\beta)\varepsilon}{4}\,.$$

Next, for large enough N a Chernoff bound gives $\Pr_{S\sim\mathcal{D}^m}[\mathcal{S}]\geq 1-e^{-\Theta(u)}\geq 2/3$, and therefore $\mathbb{E}_{\mathcal{A},S}\left[\mathbb{E}_\ell\left[\Psi_1(\ell,f_{\mathcal{A},S})\right]\right]\geq\frac{(1-\beta)\varepsilon}{6}$, and by Fubini's theorem $\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}[\Psi_1(\ell,f_{\mathcal{A},S})]\right]\geq\frac{(1-\beta)\varepsilon}{6}$.

Next, let $i\in[u-d+1,u]$. Denote by $\sigma_i\in[m]$ the number of times ξ_i was sampled into S. Then

$$\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}\left[\mathbb{1}_{\ell_i f_{\mathcal{A},S}(\xi_i)<0}\right]\right]=\sum_{n=0}^{m}\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}\left[\mathbb{1}_{\ell_i f_{\mathcal{A},S}(\xi_i)<0}\,\middle|\,\sigma_i=n\right]\right]\cdot\Pr[\sigma_i=n] \tag{8}$$

For every $x > 0$ and $y \in (0,1)$, let $\Phi(x,y) = \frac{1}{4}\left(1 - \sqrt{1 - \exp\left(\frac{-xy^2}{1-y^2}\right)}\right)$, then a result by Anthony and Bartlett [AB09, Lemma 5.1] shows that

$$\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}\left[\mathbb{1}_{\ell_i f_{\mathcal{A},S}(\xi_i)<0} \middle| \sigma_i = n\right]\right] \geq \Phi(n+2,\alpha)$$

Plugging this into (8), by the convexity of $\Phi(\cdot,\alpha)$ and Jensen's inequality we get that

$$\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}\left[\mathbb{1}_{\ell_i f_{\mathcal{A},S}(\xi_i)<0}\right]\right] \geq \sum_{n=0}^{m} \Phi(n+2,\alpha)\cdot \Pr[\sigma_i = n] \geq \Phi(\mathbb{E}[\sigma_i]+2,\alpha)\,.$$

Since $\mathbb{E}[\sigma_i] = \frac{2\beta m}{u}$, and Since $\Phi(\cdot,\alpha)$ is monotonically decreasing we get that

$$\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}\left[\mathbb{1}_{\ell_i f_{\mathcal{A},S}(\xi_i)<0}\right]\right] \geq \Phi\left(\frac{4\beta m}{u},\alpha\right)\,.$$

Summing over all $i \in [u-d+1,u]$ we get that $\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}[\Psi_2(\ell,f_{\mathcal{A},S})]\right] \geq \alpha\beta\Phi\left(\frac{4\beta m}{u},\alpha\right)$. The claim then follows from the fact that for every $\alpha \leq \sqrt{\frac{u}{40\beta m}}$ we have $\Phi(\frac{8\beta m}{u},\alpha) \geq \frac{1}{6}$. □

We next show that for small values of α, a large fraction of labelings $\ell \in \{-1,1\}^u$ satisfy that $\Psi_1 + \Psi_2$ is large with some positive constant probability over the random choices of $\mathcal{A}$ and the choice of $S \in \mathcal{S}$.

Claim 8. *If $\alpha \leq \sqrt{\frac{u}{40\beta m}}$, then with probability at least $1/11$ over the choice of $\ell \in \{-1,1\}^u$ we have*

$$\Pr_{\mathcal{A},S}\left[\Psi_1(\ell,f_{\mathcal{A},S}) + \Psi_2(\ell,f_{\mathcal{A},S}) \geq \frac{(1-\beta)\varepsilon}{24} + \frac{\alpha\beta}{24}\right] \geq \frac{1}{25}\,.$$

Proof. First note that by substituting every indicator in (3) with 1 we get that with probability 1 over all samples S, labelings ℓ and random choices of $\mathcal{A}$ we have

$$\Psi_1 + \Psi_2 \leq (1-\beta)\varepsilon + \alpha\beta\,, \tag{9}$$

and therefore $\Pr_\ell\left[\mathbb{E}_{\mathcal{A},S}[\Psi_1+\Psi_2] \leq (1-\beta)\varepsilon + \alpha\beta\right] = 1$. Furthermore, for every $\alpha \leq \sqrt{\frac{u}{40\beta m}}$ we get from Claim 7 that $\mathbb{E}_\ell\left[\mathbb{E}_{\mathcal{A},S}[\Psi_1+\Psi_2]\right] \geq \frac{1}{6}\left((1-\beta)\varepsilon+\alpha\beta\right)$. Denote $X = \mathbb{E}_{\mathcal{A},S}[\Psi_1+\Psi_2]$ and $a = (1-\beta)\varepsilon + \alpha\beta$. In these notations we have that (9) states that $\Pr_\ell[X \leq a] = 1$, and Claim 7 states that $\mathbb{E}_\ell[X] \geq a/6$. Therefore $a - X$ is a non-negative random variable, and from Markov's inequality we get that

$$\Pr_\ell[X \leq a/12] = \Pr_\ell[a - X \geq 11a/12] \leq \Pr_\ell[a-X \geq 1.1\mathbb{E}[a-X]] \leq 10/11$$

and therefore $\Pr_\ell[\mathbb{E}_{\mathcal{A},S}[\Psi_1+\Psi_2] \geq \frac{1}{12}((1-\beta)\varepsilon+\alpha\beta)] \geq 1/11$.

Next, fix some $\ell \in \{-1,1\}^u$ for which $\mathbb{E}_{\mathcal{A},S}[\Psi_1+\Psi_2] \geq \frac{1}{12}((1-\beta)\varepsilon+\alpha\beta)$. Once again, as $\Pr_{\mathcal{A},S}[\Psi_1 + \Psi_2 \leq 12\mathbb{E}_{\mathcal{A},S}[\Psi_1+\Psi_2]] = 1$ we get from Markov's inequality that with probability at least $1/25$ we have

$$\Pr_{\mathcal{A},S}\left[\Psi_1+\Psi_2 \geq \frac{(1-\varepsilon)\beta}{24} + \frac{\alpha\beta}{24}\right] \geq \Pr_{\mathcal{A},S}\left[\Psi_1+\Psi_2 \geq \frac{1}{2}\mathbb{E}_{\mathcal{A},S}[\Psi_1+\Psi_2]\right] \geq \frac{1}{25}\,.$$

□

To finish the proof of Lemma 6, observe that from Claims 5 and 8 we get that with positive probability over $\ell \in \{-1,1\}$ there exists a voting classifier $f \in C(\hat{\mathcal{H}})$ such that for every $i \in [u]$, $\ell_i f(x_i) \geq \theta$ and in addition $\Pr_{\mathcal{A},S}\left[\Psi_1 + \Psi_2 \geq \frac{(1-\varepsilon)\beta}{24} + \frac{\alpha\beta}{24}\right] \geq \frac{1}{25}$. As this occurs with positive probability, we conclude that there exists some labeling $\hat{\ell} \in \{-1,1\}^u$ satisfying both properties. Since for every set of random choices of $\mathcal{A}$, and every $S \sim \mathcal{D}_{\hat{\ell}}^m$, Claim 4 guarantees that

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y f_{\mathcal{A},S}(x)] \geq \frac{(1-\alpha)\beta}{2} + \Psi_1(\hat{\ell}, f_{\mathcal{A},S}) + \Psi_2(\hat{\ell}, f_{\mathcal{A},S}) \, ,$$

this concludes the proof of Lemma 6, and thus the proof of Theorem 1 is now complete.

3.2 Proof of Existential Lower Bound

This section is devoted to the proof of Theorem 2. That is, we show the existence of a distribution $\mathcal{D} \in \{\mathcal{D}_\ell\}_{\ell\in\{-1,1\}^u}$ such that with a constant probability over $S \sim \mathcal{D}^m$ there exists some voting classifier $f_S \in C(\hat{\mathcal{H}})$ such that f_S has large margins on points in S, but has large generalization probability with respect to $\mathcal{D}$. To this end, let m be such that $\frac{\ln N}{\theta^2} < \left(\frac{m}{\ln m}\right)^{9/10}$, and note that $m = \left(\frac{\ln N}{\theta^2}\right)^{1+\Omega(1)}$. Let $u = \frac{40m}{\ln m} \leq 2^{N^{O(1)}}$, and let $d = \frac{\ln N}{\theta^2}$.

Similarly to the proof of Theorem 1, the main challenge is to show the existence of a labeling that satisfies all desired properties. We draw the reader's attention to the fact that unlike the previous proof, the distribution over labelings is not uniform over the entire set $\{-1,1\}^u$, but rather a designated subset of sparse labelings.

With every labeling $\ell \in \{-1,1\}^u$ and an m-point sample S, we associate a classifier $h_{\ell,S}$ as follows. Intuitively, $h_{\ell,S}$ "adverserially changes" at most d labels of points in $\{\xi_2,\ldots,\xi_{u-d}\}$ that were not picked by S, and chooses the majority label for points in $\{\xi_{u-d+1},\ldots,\xi_u\}$. Formally, let $\mathcal{I}_S \subseteq \{\xi_2,\ldots,\xi_{u-d}\} \setminus S$ be a set of size at most d chosen uniformly at random, then for every $x \in \{\xi_1,\ldots,\xi_{u-d}\}$, $h_{\ell,S}(x) = -\ell(x)$ if and only if $x \in \mathcal{I}_S$, and for every $x \in \{\xi_{u-d+1},\ldots,\xi_u\}$, $h_{\ell,S}(x)$ is the majority of labels of x in S. That is $h_{\ell,S}(x) = 1$ if and only if $(x,1)$ appears in S more times than $(x,-1)$. Break ties arbitratily.

Lemma 9. *If* $\alpha \leq \sqrt{\frac{d}{40\beta m}}$ *then there exists* $\hat{\ell} \in \{-1,1\}^u$ *such that*

1. *For every* $i \in [u-d]$, $\hat{\ell}_i = 1$;
2. *With probability at least* 99/100 *over the choice of sample* $S \sim \mathcal{D}_{\hat{\ell}}^m$, *there exists a voting classifier* $f_S \in C(\hat{\mathcal{H}})$ *such that* $f_S(\xi_i) h_{\hat{\ell},S}(\xi_i) \geq \theta$ *for all* $i \in [u]$; *and*
3. *with probability at least* 1/25 *over* $S \sim \mathcal{D}_{\hat{\ell}}^m$ *we have*

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y h_{\hat{\ell},S}(x) < 0] \geq \frac{(1-\alpha)\beta}{2} + \frac{(1-\beta)\varepsilon d}{8(u-d-1)} + \frac{\alpha\beta}{24} \, .$$

We first show that the lemma implies Theorem 2.

Proof of Theorem 2. Fix some $\tau \in [0, 49/100]$. Assume first that $\tau \leq \frac{d}{50u}$, and let $\varepsilon = \frac{1}{2}$ and $\beta = \alpha = 0$. With probability 1/25 over S we have

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y h_{\hat{\ell},S}(x) < 0] \geq \frac{(1-\beta)\varepsilon d}{8u} \geq \tau + \Omega\left(\frac{d}{u}\right) = \tau + \Omega\left(\frac{\ln|\hat{\mathcal{H}}|\ln m}{m\theta^2} + \sqrt{\frac{\tau \ln|\hat{\mathcal{H}}|\ln m}{m\theta^2}}\right) \, ,$$

where the last transition is due to the fact that $d = \theta^{-2}\ln N = \Theta(\theta^{-2}\ln|\hat{\mathcal{H}}|)$ and $\tau = O(d/u)$. Moreover, with probability 99/100 over S there exists $f_S \in C(\hat{\mathcal{H}})$ such that $f_S(\xi_i)h_{\hat{\ell},S}(\xi_i) \geq \theta$ for all $i \in [u]$. We get that with probability at least 1/100 over the sample S there exists $f_S \in C(\hat{\mathcal{H}})$ such that

$$\Pr_S[y_j f_S(x_j) < \theta] = \Pr_S[y_j h_{\hat{\ell},S}(x_j) < 0] = 0 \leq \tau\,,$$

and moreover

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y f_S(x) < 0] = \Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y h_{\hat{\ell},S}(x) < 0] \geq \tau + \Omega\left(\frac{\ln|\hat{\mathcal{H}}|\ln m}{m\theta^2} + \sqrt{\frac{\tau\ln|\hat{\mathcal{H}}|\ln m}{m\theta^2}}\right).$$

Otherwise, assume $\tau > \frac{d}{50u}$, and let $\varepsilon = \frac{1}{2}$, $\alpha = \sqrt{\frac{d}{2560\tau m}}$ and $\beta = \frac{64\tau}{32-31\alpha}$. Since $\tau \geq \frac{d}{50u}$, then $\alpha \in [0,1]$. Moreover, for large enough constant $C > 0$, if $m > Cd$, then $32 - 31\alpha \geq 64 \cdot \frac{499}{1000} \geq 64 \cdot \frac{101}{100}\tau$, and therefore $\beta \in [0, 100/101]$.

Next, let $\langle (x_1,y_1),\ldots,(x_m,y_m)\rangle \sim \mathcal{D}_{\hat{\ell}}^m$ be a sample of m points drawn independently according to $\mathcal{D}_{\hat{\ell}}$. For every $j \in [m]$, let $\mathcal{E}_j$ be the event that $(x_j,y_j) \in \{(\xi_i, -\hat{\ell}_i)\}_{i\in[u-d+1,u]}$, then we have $\mathbb{1}_{y_j f_S(x_j)<0} < \mathbb{1}_{\mathcal{E}_j}$. Moreover, $\mathbb{E}[\mathbb{1}_{\mathcal{E}_j}] = \frac{(1-\alpha)\beta}{2}$, and $\{\mathbb{1}_{\mathcal{E}_j}\}_{j\in[m]}$ are independent. Therefore by Chernoff we get that for large enough N,

$$\begin{aligned}\Pr_{S\sim\mathcal{D}_{\hat{\ell}}^m}\left[\Pr_{(x,y)\sim S}\left[y h_{\hat{\ell},S}(x) < 0\right] \geq \tau\right] &\leq \Pr_{S\sim\mathcal{D}_{\hat{\ell}}^m}\left[\frac{1}{m}\sum_{j\in[m]}\mathbb{1}_{\mathcal{E}_j} \geq \frac{(1-31\alpha/32)\beta}{2}\right]\\ &\leq e^{-\Theta(\alpha^2\beta m)} = e^{-\Theta(d)} \leq 10^{-3}\,,\end{aligned}$$

where the inequality before last is due to the fact that $\alpha^2\beta m = \frac{d\beta}{2560\tau} = \Omega(d)$, since $\beta \geq 2\tau$.

Moreover, since $\alpha \leq 1$ then $\beta \leq 64\tau$, and therefore $\alpha = \sqrt{\frac{d}{2560\tau m}} \leq \sqrt{\frac{d}{40\beta m}}$. Thus with probability at least 1/25 over S we get that

$$\begin{aligned}\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y h_{\hat{\ell},S}(x) < 0] &\geq \frac{(1-\alpha)\beta}{2} + \frac{(1-\beta)\varepsilon d}{u-d-1} + \frac{\alpha\beta}{32} = \frac{(1-31\alpha/32)\beta}{2} + \frac{(1-\beta)\varepsilon d}{u-d-1} + \frac{\alpha\beta}{64}\\ &= \tau + \Omega\left(\frac{d}{u} + \sqrt{\frac{\tau d}{m}}\right) \geq \tau + \Omega\left(\frac{\ln|\hat{\mathcal{H}}|\ln m}{m\theta^2} + \sqrt{\frac{\tau\ln|\hat{\mathcal{H}}|}{m\theta^2}}\right),\end{aligned}$$

Therefore with probability at least 1/50 over the sample S we get that $\Pr_{(x,y)\sim S}\left[y h_{\hat{\ell},S}(x) < 0\right] \leq \tau$ and moreover

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y h_{\hat{\ell},S}(x) < 0] \geq \tau + \Omega\left(\frac{\ln|\hat{\mathcal{H}}|\ln m}{m\theta^2} + \sqrt{\frac{\tau\ln|\hat{\mathcal{H}}|}{m\theta^2}}\right).$$

Finally, from Lemma 9 and similarly to the first part of the proof, we get that with probability 1/100 over the choice of S there exists $f_S \in C(\hat{\mathcal{H}})$ such that $h_{\hat{\ell},S}(\xi_i)f_S(\xi_i) \geq \theta$ for all $i \in [u]$. For all these samples S we get that $\Pr_{(x,y)\sim S}\left[y f_S(x) < \theta\right] = \Pr_{(x,y)\sim S}\left[y h_{\hat{\ell},S}(x) < 0\right] \leq \tau$ and moreover

$$\Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y f_S(x) < 0] = \Pr_{(x,y)\sim\mathcal{D}_{\hat{\ell}}}[y h_{\hat{\ell},S}(x) < 0] \geq \tau + \Omega\left(\frac{\ln|\hat{\mathcal{H}}|\ln m}{m\theta^2} + \sqrt{\frac{\tau\ln|\hat{\mathcal{H}}|}{m\theta^2}}\right).$$

□

For the rest of the section we therefore prove Lemma 9. We start by lower bounding the expected value of $\Psi_1(\ell, h_{\ell,S}) + \Psi_2(\ell, h_{\ell,S})$ over a choice of a labeling ℓ and samples $S \in \mathcal{D}_\ell$. We consider next the subset $\mathcal{L}'$ of $\mathcal{L}(u,d)$ containing all labelings ℓ satisfying $\ell_i = 1$ for all $i \in [u-d]$. Intuitively, by a coupon-collector like argument we show that with very high probability over the sample S, there are at least d points in $\{\xi_i\}_{i\in[u-d]}$ not sampled into S.

Claim 10. *If* $\alpha \le \sqrt{\frac{d}{40\beta m}}$ *then*

$$\mathbb{E}_{\ell\in\mathcal{L}'}\left[\mathbb{E}_S\left[\Psi_1(\ell, h_{\ell,S}) + \Psi_2(\ell, h_{\ell,S})\right]\right] \ge \frac{(1-\varepsilon)\beta d}{2(u-d-1)} + \frac{\alpha\beta}{6}\,.$$

Proof. Let $\mathcal{S}$ be the set of all m-point samples S for which $|\{\xi_2,\dots,\xi_{u-d}\}\setminus S| \ge d$. For every $S\in\mathcal{S}$ we have $|\mathcal{I}_S| = d$, and therefore

$$\sum_{i\in[2,u-d]} \mathbb{1}_{\ell_i h_{\ell,S}(\xi_i)<0} = \sum_{i\in[2,u-d]} \mathbb{1}_{h_{\ell,S}(\xi_i)<0} = |\mathcal{I}_S| = d\,.$$

Therefore $\mathbb{E}_\ell[\mathbb{E}_S[\Psi_1(\ell,h_{\ell,S})|S\in\mathcal{S}]] = \frac{(1-\varepsilon)\beta d}{u-d-1}$. We will show next that $\Pr_S[\mathcal{S}] \ge 1/2$, and conclude that $\mathbb{E}_\ell[\mathbb{E}_S[\Psi_1(\ell,h_{\ell,S})]] \ge \frac{(1-\varepsilon)\beta d}{2(u-d-1)}$. To see this, consider a random sampling $S\sim\mathcal{D}_\ell^m$. We will show by a coupon-collector argument that with high probability, no more than $(u-d-1)-d$ elements of $\{\xi_2,\dots,\xi_{u-d}\}$ are sampled to S, and therefore $S\in\mathcal{S}$. Consider the set of elements of $\{\xi_2,\dots,\xi_{u-d}\}$ sampled by S. For every $k\in[u-2d-1]$, let X_k be the number of samples between the time $(k-1)$th distinct element was sampled from $\{\xi_2,\dots,\xi_{u-d}\}$ and the time the kth distinct element was sampled from $\{\xi_2,\dots,\xi_{u-d}\}$. Then $X_k \sim Geom\left(p_k\right)$, where $p_k = (1-\beta)\varepsilon\cdot\frac{u-d-k}{u-d-1}$. Denote $X := \sum_{k\in[u-2d-1]} X_k$, then

$$\begin{aligned}\mathbb{E}[X] &= \sum_{k\in[u-2d-1]}\frac{1}{p_k} = \sum_{k\in[u-2d-1]}\frac{u-d}{(1-\beta)\varepsilon(u-d-k)} = \frac{u-d-1}{(1-\beta)\varepsilon}\sum_{k=d+1}^{u-d-1}\frac{1}{k}\\ &\ge (u-d-1)[\ln(u-d-1)-\ln(d+1)-1] \ge \frac{1}{2}u\ln\frac{u}{d} \ge \frac{1}{20}u\ln u \ge \frac{4}{3}m\end{aligned}$$

Therefore by letting $\lambda = \frac{3}{4}$, and $p_* = \min_{k\in[u-2d-1]} p_k = (1-\beta)\varepsilon\cdot\frac{u-d-(u-2d-1)}{u-d-1} \ge \frac{d}{u}$ then known tail bounds on the sum of geometrically-distributed random variable (e.g. [Jan18, Theorem 3.1]) we get that for large enough values of m,

$$\Pr_{S\sim\mathcal{D}^m}[S\notin\mathcal{S}] = \Pr[X\le m] \le \Pr[X\le\lambda\mathbb{E}[X]] \le e^{-p_*\mathbb{E}[X](\lambda-1-\ln\lambda)} \le e^{-\Omega(\ln u)} \le 1/2\,. \tag{10}$$

The lower bound on the expectation of Ψ_2 is proved identically to the proof in Claim 7. □

Similarly to Claim 8, we conclude the following.

Claim 11. *For* $\alpha \le \sqrt{\frac{d}{40\beta m}}$, *then with probability at least* $1/11$ *over the choice of* $\ell\in\mathcal{L}'$ *we have*

$$\Pr_{S\sim\mathcal{D}_\ell^m}\left[\Psi_1(\ell,h_{\ell,S}) + \Psi_2(\ell,h_{\ell,S}) \ge \frac{(1-\beta)\varepsilon d}{4(u-d-1)} + \frac{\alpha\beta}{12}\right] \ge \frac{1}{25}\,.$$

We next want to show that there exists a labeling $\ell\in\mathcal{L}'$ such that with high probability over $S\sim\mathcal{D}_\ell^m$, there exists a voting classifier $f_S\in C(\hat{\mathcal{H}})$ attaining high margins with $h_{\ell,S}$. Since the distribution induced on $\{\xi_i\}_{i\in[u-d+1,u]}$ by $\mathcal{D}_\ell$ is uniform, we conclude the following for a large enough value of N.

Claim 12. *With probability at least* 99/100 *over the choice of a labeling* $\ell \in \mathcal{L}'$,

$$\Pr_{S\sim\mathcal{D}_\ell}\left[\exists f_S \in \mathcal{C}(\hat{\mathcal{H}}) : \forall i \in [u].\, h_{\ell,S}(\xi_i) f_S(\xi_i) \geq \theta\right] \geq \frac{99}{100}\,.$$

Proof. For two labelings $\ell \in \mathcal{L}(u,d)$ and $\ell' \in \mathcal{L}'$ we say that ℓ and ℓ' are similar, and denote $\ell \equiv \ell'$ if for all $i \in [u-d+1,u]$, $\ell_i = \ell'_i$. From Claim 5 we know that

$$\begin{aligned}1 - 1/N &\leq \Pr_{\ell\in_R\mathcal{L}(u,d)}[\exists f \in \mathcal{C}(\hat{\mathcal{H}}) : \forall i \in [u].\, \ell_i f(\xi_i) \geq \theta] = \\ &= \sum_{\ell'\in\mathcal{L}} \Pr_{\ell\in_R\mathcal{L}(u,d)}[\exists f \in \mathcal{C}(\hat{\mathcal{H}}) : \forall i \in [u].\, \ell_i f(\xi_i) \geq \theta | \ell \equiv \ell'] \cdot \Pr_{\ell\in_R\mathcal{L}(u,d)}[\ell \equiv \ell'] \\ &= \sum_{\ell'\in\mathcal{L}} \Pr_{S\sim\mathcal{D}_{\ell'}^m}[\exists f_S \in \mathcal{C}(\hat{\mathcal{H}}) : \forall i \in [u].\, h_{\ell',S}(\xi_i) f(\xi_i) \geq \theta | \ell \equiv \ell'] \cdot \Pr_{\ell\in_R\mathcal{L}(u,d)}[\ell \equiv \ell']\end{aligned}$$

For a large enough value of N we conclude that with probability at least 99/100 over a choice of $\ell' \in \mathcal{L}'$, for at least a 99/100 fraction of samples $S \sim \mathcal{D}_{\ell'}^m$ there exists a voting classifier $f_S \in C(\hat{\mathcal{H}})$ attaining high margins with $h_{\ell',S}$. □

Combining Claims 12 and 11 we conclude that if $\alpha \leq \sqrt{\frac{d}{40\beta m}}$ then there exists $\hat{\ell} \in \mathcal{L}'$ satisfying the guarantees in Lemma 9. The proof of the lemma, and therefore of Theorem 2 is now complete.

4 Existence of a Small Hypotheses Set

This section is devoted to the proof of Lemma 3. That is, we present a distribution μ over fixed-size hypothesis sets and show that for every fixed labeling ℓ with not too many negative labels, with high probability over $\mathcal{H} \sim \mu$, $C(\mathcal{H})$ contains a voting classifier f that attains good margins with respect to ℓ. In fact, our proof not only shows existence of such a voting classifier, but also presents a procedure for constructing one. The presented algorithm is an adaptation of the AdaBoost algorithm.

More formally, fix some $\theta \in (0,1/40)$, $\delta \in (0,1)$ and an integer $d \leq u$. Let $\gamma = 4\theta \in (0,1/10)$ and let $N = 2\gamma^{-2}\ln u \cdot \ln\frac{\gamma^{-2}\ln u}{\delta} \cdot e^{O(\theta^2 d)}$.We define the distribution μ via the following procedure, that samples a hypothesis set $\mathcal{H} \sim \mu$. Let $\hat{h} : \mathcal{X} \to \{-1,1\}$ be defined by $\hat{h}(x) = 1$ for all $x \in \mathcal{X}$. Sample independently and uniformly at random N hypotheses $h_1,\ldots,h_N \in_R \mathcal{X} \to \{-1,1\}$, and define $\mathcal{H} := \{\hat{h}\} \cup \{h_j\}_{j\in[N]}$.

Clearly every $\mathcal{H} \in \operatorname{supp}(\mu)$ satisfies $|\mathcal{H}| = N+1$. We therefore turn to prove the second property. To this end, let $k = \gamma^{-2}\ln u$. In order to show existence of a voting classifier, we conceptually change the procedure defining μ, and think of the random hypotheses as being sampled in k equally sized "batches", each of size N/k, and adding $\hat{h}$ to each of them. Denote the batches by $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k$. We consider next the following procedure to construct a voting classifier $f \in C(\mathcal{H})$ given $\mathcal{H} \sim \mu$. We will use the main ideas from the AdaBoost algorithm. Recall that AdaBoost creates a voting classifier using a sample $S = \langle(x_1,y_1),\ldots,(x_m,y_m)\rangle$ in iterations. Staring with $f_0 = 0$, in iteration j, it computes a new voting classifier $f_j = f_{j-1} + \alpha_j h_j$ for some hypothesis $h_j \in \mathcal{H}$ and weight α_j. The heart of the algorithm lies in choosing h_j. In each iteration, AdaBoost computes a distribution D_j over S and chooses a hypothesis h_j minimizing

$$\varepsilon_j = \Pr_{i\sim D_j}[h_j(x_i) \neq y_i].$$

The weight it then assigns is $\alpha_j = (1/2)\ln((1-\varepsilon_j)/\varepsilon_j)$ and the next distribution D_{j+1} is

$$D_{j+1}(i) = \frac{D_j(i)\exp(-\alpha_j y_i h_j(x_i))}{Z_j}$$

where Z_j is a normalization factor, namely

$$Z_j = \sum_{i=1}^{m} D_j(i)\exp(-\alpha_j y_i h_j(x_i)).$$

The first distribution D_1 is the uniform distribution.

We alter the above slightly assigning uniform weights on the hypotheses, and setting $\alpha_j = \frac{1}{2}\ln\frac{1+2\gamma}{1-2\gamma}$ for all iterations j. The algorithm is formally described as Algorithm 1.

```
Input: $(\mathcal{H}_1,\dots,\mathcal{H}_k) \sim \mu$
Output: $f \in C\left(\bigcup_{j\in[k]}\mathcal{H}_j\right)$
1: let $\alpha = \frac{1}{2}\ln\frac{1+2\gamma}{1-2\gamma}$
2: let $f(x) = 0$ for all $x \in \mathcal{X}$
3: let $D_1(i) = \frac{1}{u}$ for all $i \in [u]$.
4: for $j = 1$ to $k$ do
5:     Find a hypothesis $h_j \in \mathcal{H}_j$ satisfying $\sum_{i\in[u]} D_j(i)\mathbb{1}_{y_i\neq h_j(x_i)} \le \frac{1}{2}-\gamma$.
       If there is no such hypothesis, return fail.
6:     $f_j \leftarrow f_{j-1} + h_j$.
7:     $Z_j \leftarrow \sum_{i\in[u]} D_j(i)\exp(-\alpha y_i h_j(x_i))$.
8:     For every $i \in [u]$ let $D_{j+1}(i) = \frac{1}{Z_j}D_j(i)\exp(-\alpha y_i h_j(x_i))$.
9: return $\frac{1}{k}f_k$.
```

Algorithm 1: Construct a Voting Classifier

We will prove that the algorithm fails with probability at most δ (over the choice of $\mathcal{H}$), and that if the algorithm does not fail, then it returns a voting classifier with minimum margin at least θ. First note that if f is the classifier returned by the algorithm, then clearly $f = \frac{1}{k}\sum_{j\in[k]} h_j \in C(\mathcal{H})$ is a voting classifier.

Claim 13. *Algorithm 1 fails with probability at most δ.*

Proof. Since $\mathcal{H}_1,\dots,\mathcal{H}_k$ are independent, it is enough to show that for every $j \in [k]$, for every $w \in \Delta_u$ with probability at least $1-\delta/k$ there exists $h_j \in \mathcal{H}_j$ such that

$$\sum_{i\in[u]} w_i \mathbb{1}_{y_i\neq h_j(x_i)} \le \frac{1}{2}-\gamma\,, \tag{11}$$

where Δ_u is the u-dimensional simplex. First note that if $\sum_{i\in[u]:y_i=-1} w_i \le \frac{1}{2}-\gamma$, then $\hat{h} \in \mathcal{H}_j$ satisfies (11). We can therefore assume $\sum_{i\in[u]:y_i=-1} w_i > \frac{1}{2}-\gamma$. Next, note that for every $h : \mathcal{X} \to \{-1,1\}$ we have

$$\sum_{i\in[u]} w_i \mathbb{1}_{y_i\neq h(x_i)} = \sum_{i\in[u]} \frac{1}{2}(w_i - w_i y_i h(x_i)) = \frac{1}{2}\left(\sum_{i\in[u]} w_i - \sum_{i\in[u]} w_i y_i h(x_i)\right) = \frac{1}{2} - \frac{1}{2}\sum_{i\in[u]} w_i y_i h(x_i)$$

Therefore $\sum_{i\in[u]} w_i \mathbb{1}_{y_i\neq h(x_i)} > \frac{1}{2}-\gamma$ if and only if $\sum_{i\in[u]} w_i y_i h(x_i) < 2\gamma$. We want to show that with probability at most $\frac{\delta}{k}$ every $h \in \mathcal{H}_j$ satisfies $\sum_{i\in[u]} w_i y_i h_j(x_i) < 2\gamma$. We claim that it is enough to show that

$$\Pr_{h\in_R \mathcal{X}\to\{-1,1\}}\left[\sum_{i\in[u]} w_i y_i h(x_i) \ge 2\gamma\right] \ge \frac{k\ln\frac{k}{\delta}}{N} = \frac{1}{2}e^{-\Theta(\gamma^2 d)} \tag{12}$$

To see why this is enough assume that (12) is true, then since sampling $\mathcal{H}_j$ means independently and uniformly sampling N/k hypotheses $h \in_R \mathcal{X} \to \{-1,1\}$, the probability that there exists $h \in \mathcal{H}_j$ such that (11) holds is at least

$$1-(1-\frac{k\ln\frac{k}{\delta}}{N})^{N/k} \ge 1-\exp\left(-\frac{k\ln\frac{k}{\delta}}{N}\cdot\frac{N}{k}\right) = 1-\frac{\delta}{k}\,.$$

We thus turn to prove that (12) holds. To this end, let $M := \{i \in [u] : \beta_i < 0\}$. Recall that $|M| \le d$ and that we assumed $\sum_{i\in M} w_i = \sum_{i\in M} |y_i w_i| \ge \frac{1}{2}-\gamma$. From a known tail bound by Montgomery-Smith [MS90] on the sum of Rademacher random variables we have that since $\gamma \in (0, 1/10)$,

$$\Pr\left[\sum_{i\in[u]} w_i y_i h(x_i) \ge 2\gamma\right] \ge \Pr\left[\sum_{i\in M} w_i y_i h(x_i) \ge 2\gamma \; and \sum_{i\in[u]\setminus M} w_i y_i h(x_i) \ge 0\right] \ge \frac{1}{2}e^{-\Theta(\gamma^2 d)}$$

□

Claim 14. *If Algorithm 1 does not fail, then for every $i \in [u]$, $y_i f(x_i) \ge \theta$.*

Proof. We first show by induction that for all $j \in [k]$ we have that for all $i \in [u]$

$$\exp(-\alpha y_i f_j(x_i)) = u \cdot D_{j+1}(i) \prod_{\ell\in[j]} Z_\ell\,.$$

To see this observe that for all $i \in [u]$, $D_2(i) = \frac{D_1(i)}{Z_1}\exp(-\alpha y_i h_1(x_i))$. Since $h_1 = f_1$ and by rearranging we get that $\exp(-\alpha y_i f_1(x_i)) = \frac{D_2(i)Z_1}{D_1(i)} = u\cdot D_2(i) Z_1$. For the induction step we have that

$$\begin{aligned}\exp(-\alpha y_i f_j(x_i)) &= \exp(-\alpha y_i(f_{j-1}(x_i)+h_j(x_i))) = \exp(-\alpha y_i f_{j-1}(x_i))\cdot\exp(-\alpha y_i h_j(x_i))\\ &= u\cdot D_j(i)\prod_{\ell\in[j-1]} Z_\ell \cdot \frac{Z_j D_{j+1}(i)}{D_j(i)}\\ &= u\cdot D_{j+1}(i)\prod_{\ell\in[j]} Z_\ell\end{aligned}$$

Since $\sum_{i\in[u]} D_{k+1}(i) = 1$, we get that

$$\sum_{i\in[u]} \exp(-\alpha y_i f_k(x_i)) = u\prod_{\ell\in[k]} Z_\ell\,. \tag{13}$$

We turn therefore to bound Z_ℓ for $\ell \in [k]$. Denote $\varepsilon_\ell = \sum_{i\in[u]} D_\ell(i)\cdot \mathbb{1}_{h_\ell(x_i)\neq y_i}$. Then

$$\begin{aligned}Z_\ell &= \sum_{i\in[u]} D_\ell(i)\exp(-\alpha y_i h_\ell(x_i)) = \sum_{i\in[u]} D_\ell(i)\exp\left(-\frac{1}{2}\ln\left(\frac{1+2\gamma}{1-2\gamma}\right)y_i h_\ell(x_i)\right)\\ &= \sum_{i\in[u]} D_\ell(i)\left(\frac{1+2\gamma}{1-2\gamma}\right)^{-\frac{1}{2}y_i h_\ell(x_i)} = \varepsilon_\ell\left(\frac{1+2\gamma}{1-2\gamma}\right)^{\frac{1}{2}} + (1-\varepsilon_\ell)\left(\frac{1+2\gamma}{1-2\gamma}\right)^{-\frac{1}{2}}\\ &= \left(\frac{\varepsilon_\ell}{1-2\gamma}+\frac{1-\varepsilon_\ell}{1+2\gamma}\right)\sqrt{(1+2\gamma)(1-2\gamma)}\end{aligned}$$

By the condition in line 5 we know that $\varepsilon_\ell \le \frac{1}{2}-\gamma$. Since $\left(\frac{\varepsilon_\ell}{1-2\gamma}+\frac{1-\varepsilon_\ell}{1+2\gamma}\right)$ is increasing as a function of ε_ℓ we therefore get that

$$Z_\ell \le \left(\frac{\frac{1}{2}-\gamma}{1-2\gamma}+\frac{\frac{1}{2}+\gamma}{1+2\gamma}\right)\sqrt{(1+2\gamma)(1-2\gamma)} = \sqrt{(1+2\gamma)(1-2\gamma)} \le 1-2\gamma^2\,,$$

where the last inequality follows from the fact that $1-4\gamma^2 \le (1-2\gamma^2)^2$. Substituting in (13) we get that for every $i \in [u]$,

$$\exp(-\alpha y_i f_k(x_i) \le \sum_{i\in[u]} \exp(-\alpha y_i f_k(x_i)) = u \prod_{\ell\in[k]} Z_\ell \le u \cdot \left(1-2\gamma^2\right)^k \le \exp(\ln u - 2k\gamma^2)\,,$$

and therefore

$$y_i f(x_i) = \frac{1}{k} y_i f_k(x_i) \ge \frac{1}{k\alpha}(2k\gamma^2 - \ln u)\,. \tag{14}$$

Since $\ln(1+x) \le x$ for all $x \ge 0$ we get that

$$\alpha = \frac{1}{2}\ln\left(\frac{1+2\gamma}{1-2\gamma}\right) = \frac{1}{2}\ln\left(1+\frac{4\gamma}{1-2\gamma}\right) \le \frac{2\gamma}{1-2\gamma} \le 4\gamma\,,$$

where the last inequality follows from the fact that $\gamma \in (0, 1/4)$. Substituting in (14) we get that

$$y_i f(x_i) \ge \frac{1}{4k\gamma}(2k\gamma^2 - \ln u) = \frac{\gamma}{2} - \frac{\ln u}{4k\gamma}\,.$$

Recall that $k = \gamma^{-2}\ln u$, and therefore $y_i f(x_i) \ge \gamma/4 = \theta$. □

5 Conclusions

In this work, we showed almost tight margin-based generalization lower bounds for voting classifiers. These new bounds essentially complete the theory of generalization for voting classifers based on margins alone. Closing the remaining gap between the upper and lower bounds is an intriguing open problem and we hope our techniques might inspire further improvements. Our results come in the form of two theorems, one showing generalization lower bounds for *any* algorithm producing a voting classifier, and a slightly stronger lower bound showing the *existence* of a voting classifier with poor generalization. This raises the important question of whether specific boosting algorithms can produce voting classifiers that avoid the $\ln m$ factor in the second lower bound via a careful analysis tailored to the algorithm. As a final important direction for future work, we suggest investigating whether natural parameters other than margins may be used to better explain the practical generalization error of voting classifiers. At least, we now have an almost tight understanding, if no further parameters are taken into consideration.

References

[AB09] M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[BDST00] K. P. Bennett, A. Demiriz, and J. Shawe-Taylor. A column generation algorithm for boosting. In *ICML*, pages 65–72, 2000.

[Bre99] L. Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517, 1999.

[CG16] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[EHKV89] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247 – 261, 1989.

[FS97] Y. Freund and R. 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, 1997.

[GLM19] A. Grønlund, K. G. Larsen, and A. Mathiasen. Optimal minimal margin maximization with boosting. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4392–4401. PMLR, 2019.

[GS98] A. J. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *AAAI/IAAI*, pages 692–699, 1998.

[GZ13] W. Gao and Z.-H. Zhou. On the doubt about margin explanation of boosting. *Artificial Intelligence*, 203:1–18, 2013.

[Jan18] S. Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1 – 6, 2018. `doi:https://doi.org/10.1016/j.spl.2017.11.017`.

[KMF$^+$17] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[MS90] S. J. Montgomery-Smith. The distribution of Rademacher sums. *Proceedings of the American Mathematical Society*, 109(2):517–522, 1990. Available from: `http://www.jstor.org/stable/2048015`.

[RW02] G. Rätsch and M. K. Warmuth. Maximizing the margin with boosting. In *COLT*, volume 2375, pages 334–350. Springer, 2002.

[RW05] G. Rätsch and M. K. Warmuth. Efficient margin maximizing with boosting. *Journal of Machine Learning Research*, 6(Dec):2131–2152, 2005.

[SFBL98] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.

[WSJ$^+$11] L. Wang, M. Sugiyama, Z. Jing, C. Yang, Z.-H. Zhou, and J. Feng. A refined margin analysis for boosting algorithms via equilibrium margin. *Journal of Machine Learning Research*, 12(Jun):1835–1863, 2011.

4.3 [3] What if Neural Networks had SVDs?

What if Neural Networks had SVDs?

Alexander Mathiasen[*] **Frederik Hvilshøj**[*] **Jakob Rødsgaard Jørgensen**[*]

Anshul Nasery[*] [†] **Davide Mottin**[*]

Abstract

Various Neural Networks employ time-consuming matrix operations like matrix inversion. Many such matrix operations are faster to compute given the Singular Value Decomposition (SVD). Techniques from [10, 17] allow using the SVD in Neural Networks without computing it. In theory, the techniques can speed up matrix operations, however, in practice, they are not fast enough. We present an algorithm that is fast enough to speed up several matrix operations. The algorithm increases the degree of parallelism of an underlying matrix multiplication $H \cdot X$ where H is an orthogonal matrix represented by a product of Householder matrices.

1 Introduction

What could be done if the Singular Value Decomposition (SVD) of the weights in a Neural Network was given? Time-consuming matrix operations, such as matrix inversion [6], could be computed faster, reducing training time. However, on $d \times d$ weight matrices it takes $O(d^3)$ time to compute the SVD, which is not faster than computing the matrix inverse in $O(d^3)$ time. In Neural Networks, one can circumvent the SVD computation by using the SVD reparameterization from [17], which, in theory, reduces the time complexity of matrix inversion from $O(d^3)$ to $O(d^2)$. However, in practice, the SVD reparameterization attains no speed-up for matrix inversion on GPUs.

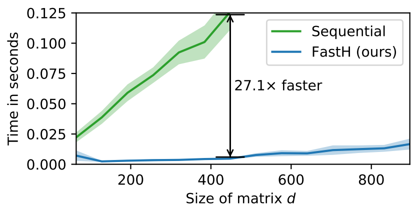


Figure 1: Time consumption of matrix inversion in Neural Networks. The plot compares FastH against the sequential algorithm from [17] (see Section 4).

The difference between theory and practice occurs because the previous technique increases sequential work, which is not taken into account by the time complexity analysis. On a $d \times d$ weight matrix, the previous technique entails the computation of $O(d)$ sequential inner products, which is ill-fit for parallel hardware like a GPU because the GPU cannot utilize all its cores. For example, if a GPU has 4000 cores and computes sequential inner products on 100-dimensional vectors, it can only utilize 100 cores simultaneously, leaving the remaining 3900 cores to run idle.

We introduce a novel algorithm, FastH, which increases core utilization, leaving less cores to run idle. This is accomplished by increasing the degree of parallelization of an underlying matrix multiplication $H \cdot X$ where H is an orthogonal matrix represented by a product of Householder matrices. FastH retains the same desirable time complexity as the sequential algorithm from [17] while reducing the number of sequential operations. On a mini-batch of size $m > 1$, FastH performs $O(d/m + m)$ sequential matrix-matrix operations instead of $O(d)$ sequential vector-vector operations.

In practice, FastH is faster than all algorithms from [17], e.g., FastH is 27 times faster than their sequential algorithm, see Figure 1. Code `www.github.com/AlexanderMath/fasth`.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

[*]Aarhus University, {alexander.mathiasen, fhvilshoj, mrjakobdk}@gmail.com, davide@cs.au.dk
[†]Indian Institute of Technology, Bombay, anshulnasery@gmail.com

2 Background

2.1 Fast Matrix Operations Using SVD

The SVD allows faster computation of many matrix operations commonly used by Neural Networks. A few examples include matrix determinant [3], matrix inverse [7], Spectral Normalization [11], the matrix exponential [8], the Cayley transform [4], weight decay, condition number and compression by low-rank approximation [16]. Proofs can be found in most linear algebra textbooks, see, e.g., [12].

2.2 The SVD Reparameterization

This subsection describes how [17] allows for using the SVD of the weight matrices in Neural Networks without computing them, and in particular, how this approach is limited by the computation of sequential inner products. Let $W = U\Sigma V^T$ be the SVD of a weight matrix W where Σ is a diagonal matrix and U, V are orthogonal matrices, i.e, $U^T = U^{-1}$ and $V^T = V^{-1}$. The goal is to perform gradient descent updates to W while preserving the SVD. Consider updating U, Σ, V a small step $\eta \in \mathbb{R}$ in the direction of gradients $\nabla_U, \nabla_\Sigma, \nabla_V$.

$$\Sigma' = \Sigma - \eta\nabla_\Sigma, \quad U' = U - \eta\nabla_U, \quad V' = V - \eta\nabla_V.$$

While Σ' remains diagonal, both U' and V' are in general not orthogonal, which is needed to preserve the SVD. To this end, [17] suggested using a technique from [10] which decomposes an orthogonal matrix as a product of d Householder matrices $H_1, \ldots, H_d$:

$$U = \prod_{i=1}^{d} H_i \qquad H_i = I - 2\frac{v_i v_i^T}{||v_i||_2^2} \qquad v_i \in \mathbb{R}^d. \tag{1}$$

Householder matrices satisfy several useful properties. In particular, the matrix U remains orthogonal under gradient descent updates $v_i = v_i - \eta\nabla_{v_i}$ [10]. Furthermore, all products of Householder matrices are orthogonal, and any $d \times d$ orthogonal matrix can be decomposed as a product of d Householder matrices [14]. Householder matrices thus allow us to perform gradient descent over orthogonal matrices, which allows us to preserve the SVD of W during gradient descent updates.

Multiplication. One potential issue remains. The Householder decomposition might increase the time it takes to multiply UX for a mini-batch $X \in \mathbb{R}^{d\times m}$ during the forward pass. Computing $UX = H_1 \cdots (H_{d-1}(H_d \cdot X))$ takes d Householder multiplications. If done sequentially, as indicated by the parenthesis, each Householder multiplication can be computed in $O(dm)$ time [17]. All d multiplications can thus be done in $O(d^2m)$ time. Therefore, the Householder decomposition does not increase the time complexity of computing UX.

Unfortunately, the $O(d^2m)$ time complexity comes at the cost of multiplying each Householder matrix sequentially, and each Householder multiplication entails computing an inner product, see Equation (1). The multiplication UX then requires the computation of $O(d)$ inner products sequentially. Such sequential computation is slow on parallel hardware like GPUs, much slower than normal matrix multiplication. To exploit GPUs, [17] suggested using a parallel algorithm that takes $O(d^3)$ time, but this is no faster than computing the SVD.

We are thus left with two options: (1) an $O(d^2m)$ sequential algorithm and (2) an $O(d^3)$ parallel algorithm. The first option is undesirable since it entails the sequential computation of $O(d)$ inner products. The second option is also undesirable since it takes $O(d^3)$ which is the same as computing the SVD, i.e., we might as-well just compute the SVD. In practice, both algorithms usually achieve no speed-up for the matrix operations like matrix inversion as we show in Section 4.2.

Our main contribution is a novel parallel algorithm, FastH, which resolves the issue with sequential inner products without increasing the time complexity. FastH takes $O(d^2m)$ time but performs $O(d/m+m)$ sequential matrix-matrix operations instead of $O(d)$ sequential vector-vector operations (inner products). In practice, FastH is up to $6.2\times$ faster than the parallel algorithm and up to $27.1\times$ faster than the sequential algorithm, see Section 4.1.

Mathematical Setting. We compare the different methods by counting the number of sequential matrix-matrix and vector-vector operations. We count only once when other sequential operations can be done in parallel. For example, processing $v_1, ..., v_{d/2}$ sequentially while, in parallel, processing $v_{d/2+1}, ..., v_d$ sequentially, we count $d/2$ sequential vector-vector operations.

Orthogonal Gradient Descent. The SVD reparameterization performs gradient descent over orthogonal matrices. This is possible with Householder matrices, however, other techniques, such as [2, 9], rely on the matrix exponential and the Cayley map, respectively. For $d \times d$ matrices both techniques spend $O(d^3)$ time, which is no faster than computing the SVD.

3 Fast Householder Multiplication (FastH)

3.1 Forward Pass

Our goal is to create an $O(d^2m)$ algorithm with few sequential operations that solves the following problem: Given an input $X \in \mathbb{R}^{d\times m}$ with $d > m > 1$ and Householder matrices $H_1, ..., H_d$, compute the output $A = H_1 \cdots H_d X$. For simplicity, we assume m divides d.

Since each H_i is a $d \times d$ matrix, it would take $O(d^3)$ time to read the input $H_1, ..., H_d$. Therefore, we represent each Householder matrix H_i by its Householder vector v_i such that $H_i = I - 2v_iv_i^T/||v_i||_2^2$. A simplified version of the forward pass of FastH proceeds as follows: divide the Householder product $H_1 \cdots H_d$ into smaller products $P_1 \cdots P_{d/m}$ so each P_i is a product of m Householder matrices:

$$P_i = H_{(i-1)\cdot m+1} \cdots H_{i\cdot m} \qquad i = 1, ..., d/m. \tag{2}$$

All d/m products P_i can be computed in parallel. The output can then be computed by $A = P_1 \cdots P_{d/m} X$ instead of $A = H_1 \cdots H_d X$, which reduces the number of sequential matrix multiplications from d to d/m.

This algorithm computes the correct A. However, the time complexity increases due to two issues. First, multiplying each product P_i with X takes $O(d^2m)$ time, a total of $O(d^3)$ time for all d/m products. Second, we need to compute all d/m products $P_1, ..., P_{d/m}$ in $O(d^2m)$ time, so each product P_i must be computed in $O(d^2m/(d/m)) = O(dm^2)$ time. If we only use the Householder structure, it takes $O(d^2m)$ time to compute each P_i, which is not fast enough.

Both issues can be resolved, yielding an $O(d^2m)$ algorithm. The key ingredient is a linear algebra result [1] that dates back to 1987. The result is restated in Lemma 1.

Lemma 1. *For any m Householder matrices $H_1, ..., H_m$ there exists $W, Y \in \mathbb{R}^{d\times m}$ st. $I - 2WY^T = H_1 \cdots H_m$. Computing W and Y takes $O(dm^2)$ time and m sequential Householder multiplications.*

For completeness, we provide pseudo-code in Algorithm 1. Theorem 1 states properties of Algorithm 1 and its proof clarify how Lemma 1 solves both issues outlined above.

Theorem 1. *Algorithm 1 computes $H_1 \cdots H_d X$ in $O(d^2m)$ time with $O(d/m + m)$ sequential matrix multiplications.*

Proof. **Correctness.** Each iteration of Step 2 in Algorithm 1 utilizes Lemma 1 to compute $A_i = A_{i+1} - 2W_i(Y_i^T A_{i+1}) = P_i A_{i+1}$. Therefore, at termination, $A_1 = P_1 \cdots P_{d/m} X$. In Step 1, we used Lemma 1 to compute the P_i's such that $A = H_1 \cdots H_d X$ as wanted.

Time Complexity. Consider the for loop in Step 1. By Lemma 1, each iteration takes $O(dm^2)$ time. Therefore, the total time of the d/m iterations is $O(dm^2 d/m) = O(d^2m)$. Consider iteration i of the loop in Step 2. The time of the iteration is asymptotically dominated by both matrix multiplications. Since A_{i+1}, X_i and Y_i are $d \times m$ matrices, it takes $O(dm^2)$ time to compute both matrix multiplications. There are d/m iterations so the total time becomes $O(dm^2 d/m) = O(d^2m)$.

Number of Sequential Operations. Each iteration in Step 2 performs two sequential matrix multiplications. There are d/m sequential iterations which gives a total of $O(d/m)$ sequential matrix multiplications. Each iteration in Step 1 performs m sequential Householder multiplications to construct P_i, see Lemma 1. Since each iteration is run in parallel, the algorithm performs no more than $O(d/m + m)$ sequential matrix multiplications. □

Remark. Section 3.2 extends the techniques from this section to handle gradient computations. For simplicity, this section had Algorithm 1 compute only A_1, however, in Section 3.2 it will be convenient to assume $A_1, ..., A_{d/m}$ are precomputed. Each $A_i = P_i \cdots P_{d/m} X$ can be saved during Step 2 of Algorithm 1 without increasing asymptotic memory consumption.

3.2 Backwards Propagation

This subsection extends the techniques from Section 3.1 to handle gradient computations. Our goal is to create an $O(d^2m)$ algorithm with few sequential operations that solves the following problem: Given $A_1, \dots, A_{d/m+1}$, $P_1, ..., P_{d/m}$ and $\frac{\partial L}{\partial A_1}$ for some loss function L, compute $\frac{\partial L}{\partial X}$ and $\frac{\partial L}{\partial v_1}, ..., \frac{\partial L}{\partial v_d}$, where v_j is a Householder vector st. $H_j = I - 2v_j v_j^T/||v_j||_2^2$.

Since each P_i is a $d \times d$ matrix, it would take $O(d^3/m)$ time to read the input $P_1, ..., P_{d/m}$. Therefore, we represent each P_i by its WY decomposition $P_i = I - 2WY^T$.

On a high-level the backward pass of FastH has two steps.

Step 1. Sequentially compute $\frac{\partial L}{\partial A_2}, \frac{\partial L}{\partial A_3}, ..., \frac{\partial L}{\partial A_{d/m+1}}$ by

$$\frac{\partial L}{\partial A_{i+1}} = \left[\frac{\partial A_i}{\partial A_{i+1}}\right]^T \frac{\partial L}{\partial A_i} = P_i^T \frac{\partial L}{\partial A_i} \tag{3}$$

This also gives the gradient wrt. X since $X = A_{d/m+1}$.

Step 2. Use $\frac{\partial L}{\partial A_1}, ..., \frac{\partial L}{\partial A_{d/m}}$ from Step 1 to compute the gradient $\frac{\partial L}{\partial v_j}$ for all j. This problem can be split into d/m subproblems, which can be solved in parallel, one subproblem for each $\frac{\partial L}{\partial A_i}$.

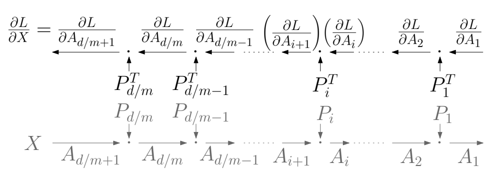
Details. For completeness, we state pseudo-code in Algorithm 2, which we now explain with the help of Figure 2. Figure 2a depicts a computational graph of Step 1 in Algorithm 2. In the figure, consider $\frac{\partial L}{\partial A_1}$ and P_1^T, which both have directed edges to a multiplication node (denoted by $\cdot$). The output of this multiplication is $\frac{\partial L}{\partial A_2}$ by Equation (3). This can be repeated to obtain $\frac{\partial L}{\partial A_2}, ..., \frac{\partial L}{\partial A_{d/m+1}}$.

Step 2 computes the gradient of all Householder vectors $\frac{\partial L}{\partial v_j}$. This computation is split into d/m distinct subproblems that can be solved in parallel. Each subproblem concerns $\frac{\partial L}{\partial A_i}$ and the product P_i, see line 8-10 in Algorithm 2.

To ease notation, we index the Householder matrices of P_i by $P_i = \widehat{H}_1 \cdots \widehat{H}_m$. Furthermore, we let $\widehat{A}_{m+1} := A_{i+1}$ and $\widehat{A}_j := \widehat{H}_j \widehat{A}_{j+1}$. The notation implies that $\widehat{A}_1 = \widehat{H}_1 \cdots \widehat{H}_m \widehat{A}_{m+1} = P_i A_{i+1} = A_i$. The goal of each subproblem is to compute gradients wrt. the Householder vectors $\widehat{v}_m, ..., \widehat{v}_1$ of $\widehat{H}_m, ..., \widehat{H}_1$. To compute the gradient of $\widehat{v}_j$, we need $\widehat{A}_{j+1}$ and $\frac{\partial L}{\partial \widehat{A}_j}$, which can be computed by:

$$\widehat{A}_{j+1} = \widehat{H}_j^{-1} \widehat{A}_j = \widehat{H}_j^T \widehat{A}_j \qquad \frac{\partial L}{\partial \widehat{A}_{j+1}} = \left[\frac{\partial \widehat{A}_j}{\partial \widehat{A}_{j+1}}\right]^T \frac{\partial L}{\partial \widehat{A}_j} = \widehat{H}_j^T \frac{\partial L}{\partial \widehat{A}_j} \tag{4}$$

Figure 2b depicts how $\widehat{A}_2, ..., \widehat{A}_{m+1}$ and $\frac{\partial L}{\partial \widehat{A}_2}, ..., \frac{\partial L}{\partial \widehat{A}_{m+1}}$ can be computed given $\widehat{A}_1$ and $\frac{\partial L}{\partial \widehat{A}_1}$. Given $\widehat{A}_{j+1}$ and $\frac{\partial L}{\partial \widehat{A}_j}$, we can compute $\frac{\partial L}{\partial \widehat{v}_j}$ as done in [10, 17]. For completeness, we restate the needed equation in our notation, see Equation (5).


$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial A_{d/m+1}}$
$\frac{\partial L}{\partial A_{d/m}}$
$\frac{\partial L}{\partial A_{d/m-1}}$
$\left(\frac{\partial L}{\partial A_{i+1}}\right)\left(\frac{\partial L}{\partial A_i}\right)$
$\frac{\partial L}{\partial A_2}$
$\frac{\partial L}{\partial A_1}$
$P_{d/m}^T$
$P_{d/m-1}^T$
P_i^T
P_1^T
$P_{d/m}$
$P_{d/m-1}$
P_i
P_1
X
$A_{d/m+1}$
$A_{d/m}$
$A_{d/m-1}$
A_{i+1}
A_i
A_2
A_1
(a) Step 1: Sequential part of Algorithm 2.

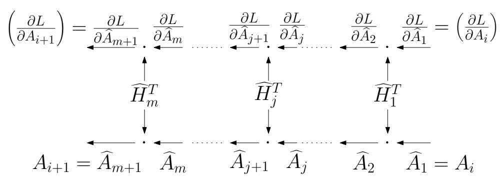

$\left(\frac{\partial L}{\partial A_{i+1}}\right) = \frac{\partial L}{\partial \widehat{A}_{m+1}}$
$\frac{\partial L}{\partial \widehat{A}_m}$
$\frac{\partial L}{\partial \widehat{A}_{j+1}}$
$\frac{\partial L}{\partial \widehat{A}_j}$
$\frac{\partial L}{\partial \widehat{A}_2}$
$\frac{\partial L}{\partial \widehat{A}_1} = \left(\frac{\partial L}{\partial A_i}\right)$
$\widehat{H}_m^T$
$\widehat{H}_j^T$
$\widehat{H}_1^T$
$A_{i+1} = \widehat{A}_{m+1}$
$\widehat{A}_m$
$\widehat{A}_{j+1}$
$\widehat{A}_j$
$\widehat{A}_2$
$\widehat{A}_1 = A_i$
(b) Step 2: The i'th subproblem in Algorithm 2.

Figure 2: Computational graph of Step 1 and the i'th subproblem in Step 2 from Algorithm 2.

Let $a^{(l)}$ be the l'th column of $\widehat{A}_{j+1}$ and let $g^{(l)}$ be the l'th column of $\frac{\partial L}{\partial \widehat{A}_j}$. The sum of the gradient over a mini-batch of size m is then:

$$-\frac{2}{||\widehat{v}_j||_2^2}\sum_{l=1}^{m}(\widehat{v}_j^T a^{(l)})g^{(l)} + (\widehat{v}_j^T g^{(l)})a^{(l)} - \frac{2}{||\widehat{v}_j||_2^2}(\widehat{v}_j^T a^{(l)})(\widehat{v}_j^T g^{(l)})\widehat{v}_j \tag{5}$$

Theorem 2 states properties of Algorithm 2.

Theorem 2. *Algorithm 2 computes $\frac{\partial L}{\partial X}$ and $\frac{\partial L}{\partial v_1}, ..., \frac{\partial L}{\partial v_d}$ in $O(d^2m)$ time with $O(d/m+m)$ sequential matrix multiplications.*

Proof. See the Supplementary Material 8.1. □

Algorithm 1 FastH Forward

```
 1: Input: $X \in \mathbb{R}^{d\times m}$ and $H_1, ..., H_d \in \mathbb{R}^{d\times d}$.
 2: Output: $A_1 = H_1 \cdots H_d X$.
 3: // Step 1
 4: for $i = d/m$ to 1 do in parallel
 5:     Compute $Y_i, W_i \in \mathbb{R}^{d\times m}$ st.
            $P_i = I - 2W_iY_i^T$                    ▷ $O(dm^2)$
        by using Lemma 1.
 6: end for
 7: // Step 2
 8: $A_{d/m+1} = X$.
 9: for $i = d/m$ to 1 do sequentially
10:     $A_i = A_{i+1} - 2W_i(Y_i^T A_{i+1})$. ▷ $O(dm^2)$
11: end for
12: return $A_1$.
```

Algorithm 2 FastH Backward

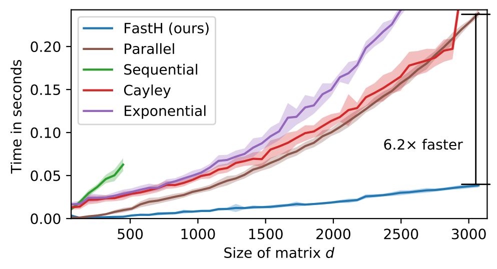
```
 1: Input: $A_1, ..., A_{d/m+1}$, $P_1, ..., P_{d/m}$ and $\frac{\partial L}{\partial A_1}$.
 2: Output: $\frac{\partial L}{\partial X}$ and $\frac{\partial L}{\partial v_k}$ for all $k$ where $H_k = I - 2\frac{v_k v_k^T}{||v_k||_2^2}$.
 3: // Step 1
 4: for $i = 1$ to $d/m$ do sequentially
 5:     $\frac{\partial L}{\partial A_{i+1}} = P_i^T \frac{\partial L}{\partial A_i}$ eq. (3).        ▷ $O(dm^2)$
 6: end for
 7: // Step 2
 8: for $i = 1$ to $d/m$ do in parallel
 9:     Let $\frac{\partial L}{\partial \widehat{A}_1} = \left(\frac{\partial L}{\partial A_i}\right)$.
10:     To ease notation, let $P_i = \widehat{H}_1 \cdots \widehat{H}_m$.
11:     for $j = 1$ to $m$ do
12:         Compute $\widehat{A}_{j+1}$, $\frac{\partial L}{\partial \widehat{A}_j}$ see eq. (4).        ▷ $O(dm)$
13:         Compute $\frac{\partial L}{\partial \widehat{v}_j}$ using $\widehat{A}_{j+1}$, $\frac{\partial L}{\partial \widehat{A}_j}$, eq. (5). ▷ $O(dm)$
14:     end for
15: end for
16: return $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial A_{d/m+1}}$ and $\frac{\partial L}{\partial v_k}$ for all $k = 1, ..., d$.
```

3.3 Extensions

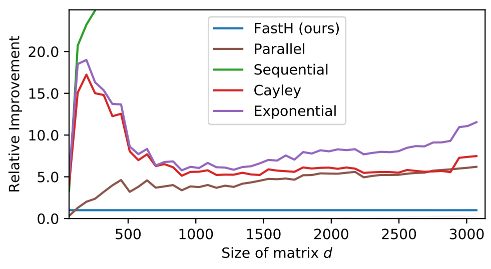
Trade-off. Both Algorithm 1 and Algorithm 2 can be extended to take a parameter k that controls a trade-off between *total time complexity* and *the amount of parallelism*. This can be achieved by changing the number of Householder matrices in each product P_i from the mini-batch size m to an integer k. The resulting algorithms take $O(d^2k + d^2m)$ time, $O(d^2m/k)$ space and has $O(d/k + k)$ sequential matrix multiplications. This extension has the practical benefit that one can try different values of k and choose the one that yields superior performance on a particular hardware setup. Note that we never need to search for k more than once. The number of sequential matrix multiplications $O(d/k + k)$ is minimized when $k = O(\sqrt{d})$. For a constant $c > 1$, we can find the best $k \in \{2, 3, ..., c\lceil\sqrt{d}\rceil\}$ by trying all $O(\sqrt{d})$ values. The search needs to be done only once and takes $O(\sqrt{d}(d^2k + d^2m)) = O(d^3 + d^{2.5}m)$ time. In practice, the time it took to find k was negligable, e.g., on the hardware we describe in Section 4 we found k in less than $1s$ for $d = 784$.

Rectangular Matrices. We can use the SVD reparametrization for rectangular $W \in \mathbb{R}^{n\times m}$. Use orthogonal $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{m\times m}$ and diagonal $\Sigma \in \mathbb{R}^{n\times m}$ and let $W = U\Sigma V^T$.

Convolutional Layers. So far, we have considered the SVD reparameterization for matrices which corresponds to fully connected layers. The matrix case extends to convolutions by, e.g., 1×1 convolutions [7]. The SVD reparameterization can be used for such convolutions without increasing the time complexity. On an input with height h and width w FastH performs $O(d/m + mhw)$ sequential matrix multiplications instead of the $O(d)$ sequential inner products.



(a) Running time.



(b) Relative improvement.

Figure 3: Comparisons of the running times for FastH against previous algorithms. The sequential algorithm from [17] crashed when $d > 448$. (a) Running times of different algorithms for $d \times d$ matrices. (b) Running times of FastH relative to previous algorithms, i.e., the mean time of a previous algorithm divided by the mean time of FastH.

Recurrent Layers. The SVD reparameterization was developed for Recurrent Neural Networks (RNNs) [17]. Let r be the number of recurrent applications of the RNN. FastH performs $O(d/m+rm)$ sequential matrix operations instead of the $O(d)$ sequential inner products.

4 Experiments

This section contains two experiments. Section 4.1 compares the running time of FastH against alternatives. Section 4.2 shows that FastH speeds up matrix operations. To simulate a realistic machine learning environment, we performed all experiments on a standard machine learning server using a single NVIDIA RTX 2080 Ti.

4.1 Comparing Running Time

This subsection compares the running time of FastH against four alternative algorithms. We compare the time all algorithms spend on gradient descent with a single orthogonal matrix, since such constrained gradient descent dominates the running time of the SVD reparameterization.

We first compare FastH against the parallel and sequential algorithm from [17], all three algorithms rely on the Householder decomposition. For completeness, we also compare against approaches that does not rely on the Householder decomposition, in particular, the matrix exponential and the Cayley map [2][3]. See Supplementary Material 8.2 for further details.

We measure the time of a gradient descent step with a weight matrix $W \in \mathbb{R}^{d\times d}$ and a mini-batch $X \in \mathbb{R}^{d\times m}$, where $m = 32$ and $d = 1\cdot 64, 2\cdot 64, ..., 48\cdot 64$. We ran each algorithm 100 times, and we report mean time μ with error bars $[\mu-\sigma, \mu+\sigma]$ where σ is the standard deviation of running time over the 100 repetitions.

Figure 3a depicts the running time on the y-axis, as the size of the $d \times d$ matrices increases on the x-axis. For $d > 64$, FastH is faster than all previous approaches. At $d = 64$ FastH is faster than all previous approaches, except the parallel algorithm. Previous work employ sizes $d = 192$ in [7] and $d = 784$ in [17].

Figure 3b depicts how much faster FastH is relative to the previous algorithms, i.e., the mean time of a previous algorithm divided by the time of FastH, which we refer to as relative improvement. For $d > 500$, the relative improvement of FastH increases with d.

At $d = 448$ FastH is roughly $25\times$ faster than the sequential algorithm. FastH is even faster with $d = 3072$ than the sequential algorithm with $d = 448$. Previous work like [6, 15] use the Householder decomposition with the sequential algorithm. Since FastH computes the same thing as the sequential algorithm, it can be used to reduce computation time with no downside.

[3]For the matrix exponential and the Cayley map we used the open-source implementation https://github.com/Lezcano/expRNN from [2]. For the Householder decomposition, we used the open-source implementation https://github.com/zhangjiong724/spectral-RNN of the sequential and parallel algorithm from [17].

Table 1: Relating standard method to matrix decompositions for computing matrix operations.

Matrix Operation	Standard Method	SVD or Eigendecomposition
Determinant	TORCH.SLOGDET(W)	$\sum_{i=1}^{d} \lg \lvert\Sigma_{ii}\rvert$
Inverse	TORCH.INVERSE(W)	$V\Sigma^{-1}U^T$
Matrix Exponential	Padé Approximation [2]	$Ue^{\Sigma}U^T$
Cayley map	TORCH.SOLVE(I-W, I+W)	$U(I-\Sigma)(I+\Sigma)^{-1}U^T$

4.2 Using the SVD to Compute Matrix Operations

This subsection investigates whether the SVD reparameterization achieves practical speed-ups for matrix operations like matrix inversion. We consider four different matrix operations. For each operation, we compare the SVD reparameterization against the standard method for computing the specific matrix operation, see Table 1.

Timing the Operations. The matrix operations are usually used during the forward pass of a Neural Network, which change the subsequent gradient computations. We therefore measure the sum of the time it takes to compute the matrix operation, the forward pass and the subsequent gradient computations.

For example, with matrix inversion, we measure the time it takes to compute the matrix operation Σ^{-1}, the forward pass $W^{-1}X = V\Sigma^{-1}U^TX$ and the subsequent gradient computation wrt. U, Σ, V and X. The measured time is then compared with TORCH.INVERSE, i.e, we compare against the total time it takes to compute TORCH.INVERSE(W), the forward pass $W^{-1}X$, and the subsequent gradient computation wrt. W and X.

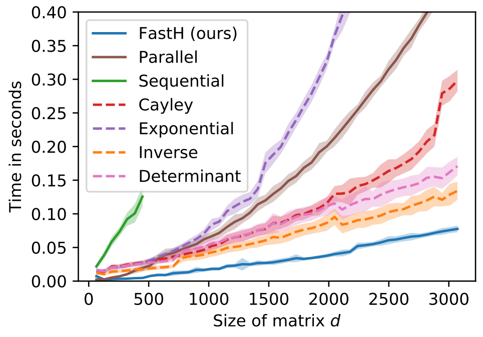


Figure 4: Running time of matrix operations. Solid lines depict approaches which use the SVD reparameterization and dashed lines depict standard methods like TORCH.INVERSE.

Setup. We run the SVD reparameterization with three different algorithms: FastH, the sequential and the parallel algorithm from [17]. For each matrix operation, we consider matrices $V, \Sigma, U, W \in \mathbb{R}^{d\times d}$ and $X \in \mathbb{R}^{d\times m}$, where $m = 32$ and $d = 1 \cdot 64, 2 \cdot 64, ..., 48 \cdot 64$. We repeat the experiment 100 times, and report the mean time μ with error bars $[\mu - \sigma, \mu + \sigma]$ where σ is the standard deviation of the running times over the 100 repetitions. To avoid clutter, we plot only the time of FastH for the matrix operation it is slowest to compute, and the time of the sequential and parallel algorithms for the matrix operation they were fastest to compute.

Figure 4 depicts the measured running time on the y-axis with the size of the $d \times d$ matrices increasing on the x-axis. Each matrix operation computed by a standard method is plotted as a dashed line, and the different algorithms for the SVD reparameterization are plotted as solid lines. In all cases, FastH is faster than the standard methods. For example, with $d = 768$, FastH is $3.1\times$ faster than the Cayley map, $4.1\times$ faster than the matrix exponential, $2.7\times$ faster than inverse and $3.5\times$ faster than matrix determinant. The sequential algorithm is not fast enough to speed up any matrix operation.

5 Related Work

The Householder Decomposition. The Householder decomposition of orthogonal matrices has been used in much previous works, e.g., [6, 10, 13, 15, 17]. Previous work typically use a type of sequential algorithm that performs $O(d)$ sequential inner products. To circumvent the resulting long computation time on GPUs, previous work often suggest limiting the number of Householder matrices, which limits the expressiveness of the orthogonal matrix, introducing a trade-off between computation time and expressiveness.

FastH takes the same asymptotic time as the sequential algorithm, however, it performs less sequential matrix operations, making it up to $27\times$ faster in practice. Since FastH computes the same output as the previous sequential algorithms, it can be used in previous work without degrading the performance of their model. In particular, FastH can be used to either (1) increase expressiveness at no additional computational cost or (2) retain the same level of expresiveness at lower computational cost.

SVDs in Neural Networks. The authors of [17] introduced a technique that provides access to the SVD of the weights in a Neural Network without computing the SVD. Their motivation for developing this technique was the exploding/vanishing gradient issue in RNNs. In particular, they use the SVD reparameterization to force all singular values to be within the range $[1 \pm \epsilon]$ for some small ϵ.

We point out that although their technique, in theory, can be used to speed up matrix operations, their algorithms are too slow to speed-up most matrix operations in practice. To mitigate this problem, we introduce a new algorithm that is more suitable for GPUs, which allows us to speed up several matrix operations in practice.

Different Orthogonal Parameterizations. The SVD reparameterization by [17] uses the Householder decomposition to perform gradient descent with orthogonal matrices. Their work was followed by [4] that raises a theoretical concern about the use of Householder decomposition. Alternative approaches based on the matrix exponential and the Cayley map have desirable provable guarantees, which currently, it is not known whether the Householder decomposition possesses. This might make it desirable to use the matrix exponential or the Cayley map together with the SVD reparameterization from [17]. However, previous work spend $O(d^3)$ time to compute or approximate the matrix exponential and the Cayley map. These approaches are therefore undesirable, because they share the $O(d^3)$ time complexity with SVD and thus cannot speed up SVD computations.

Normalizing Flows. Normalizing Flows [3] is a type of generative model that, in some cases [6, 7], entails the computation of matrix determinant and matrix inversion. [7] propose to use the PLU decomposition $W = PLU$ where P is a permutation matrix and L, U are lower and upper triangular. The decomposition allows the determinant computation in $O(d)$ time instead of $O(d^3)$. [6] point out that a fixed permutation matrix P limits flexibility. To fix this issue, they suggest using the QR decomposition where R is a rectangular matrix and Q is orthogonal. They suggest making Q orthogonal by using the Householder decomposition which FastH can speed up. Alternatively, one could use the SVD decomposition instead of the QR or PLU decomposition.

6 Code

To make FastH widely accessible, we wrote a PyTorch implementation of the SVD reparameterization which uses the FastH algorithm. The implementation can be used by changing just a single line of code, i.e, change NN.LINEAR to LINEARSVD. While implementing FastH, we found that Python did not provide an adequate level of parallelization. We therefore implemented FastH in CUDA to fully utilize the parallel capabilities of GPUs. Code: `github.com/AlexanderMath/fasth/`.

7 Conclusion

We pointed out that, in theory, the techniques from [10, 17] allow decreasing the time complexity of several matrix operations used in Neural Networks. However, in practice, we demonstrated that the techniques are not fast enough on GPUs for moderately sized use-cases. We proposed a novel algorithm, FastH, that remedies the issues with both algorithms from [17], which is up to $27\times$ faster than the previous sequential algorithm. FastH introduces no loss of quality, it computes the same result as the previous algorithms, just faster. FastH brings two immediate benefits: (1) improves upon the techniques from [17] in such a way that it is possible to speed up matrix operations, and (2) speeds up previous work that employ the Householder decomposition as done in, e.g., [6, 13, 15].

Broader Impact

Our algorithm speeds up the use of Householder decompositions in Neural Networks. This can positively impact researchers who use Householder decompositions, since they will be able to execute experiments faster. This is particularly beneficial for researchers with a constraint on their computational budget, in other words, a PhD student with one GPU stands to benefit more than a lab with state-of-the-art GPU computing infrastructure. The reduction in computing time also decrease power consumption and thus carbon emissions. However, as a potential negative impact, it is possible that the decrease in computation time will increase the usage of Neural Networks and thus increase overall carbon emission.

Acknowledgements Alexander Mathiasen was supported by Kasper Green Larsen's AUFF Starting Grant. Frederik Hvilshøj was supported by DIGIT Aarhus University Centre for Digitalisation, Big Data and Data Analytics. Jakob Rødsgaard Jørgensen was supported by the Independent Research Fund Denmark. Finally, we thank DIGIT Aarhus University Centre for Digitalisation for providing us with GPU capabilities.

References

[1] Christian Bischof and Charles Van Loan. The WY Representation for Products of Householder Matrices. *SIAM Journal on Scientific and Statistical Computing*, 1987.

[2] Mario Lezcano Casado. Trivializations for Gradient-Based Optimization on Manifolds. In *NeurIPS*, 2019.

[3] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-Linear Independent Components Estimation. In *ICLR (Workshop)*, 2015.

[4] Adam Golinski, Mario Lezcano-Casado, and Tom Rainforth. Improving Normalizing Flows via Better Orthogonal Parameterizations. In *ICML Workshop on Invertible Neural Networks and Normalizing Flows*, 2019.

[5] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The Reversible Residual Network: Backpropagation Without Storing Activations. In *NIPS*, 2017.

[6] Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging Convolutions for Generative Normalizing Flows. In *ICML*, 2019.

[7] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In *NeurIPS*. 2018.

[8] Mario Lezcano-Casado and David Martínez-Rubio. Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. In *ICML*, 2019.

[9] Jun Li, Fuxin Li, and Sinisa Todorovic. Efficient Riemannian Optimization on the Stiefel Manifold via the Cayley Transform. In *ICLR*, 2020.

[10] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections. In *ICML*, 2017.

[11] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *ICLR*, 2018.

[12] Gilbert Strang. *Linear Algebra and its Applications*. 2006.

[13] Jakub M Tomczak and Max Welling. Improving Variational Auto-Encoders using Householder Flow. *arXiv preprint*, 2016.

[14] Frank Uhlig. Constructive Ways for Generating (Generalized) Real Orthogonal Matrices as Products of (Generalized) Symmetries. *Linear Algebra and its Applications*, 2001.

[15] Rianne van den Berg, Leonard Hasenclever, Jakub Tomczak, and Max Welling. Sylvester Normalizing Flows for Variational Inference. In *UAI*, 2018.

[16] Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition. 2013.

[17] Jiong Zhang, Qi Lei, and Inderjit Dhillon. Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization. In *ICML*, 2018.

8 Supplementary Material

8.1 Proof of Theorem 2.

Theorem. *Algorithm 2 computes $\frac{\partial L}{\partial X}$ and $\frac{\partial L}{\partial v_1}, ..., \frac{\partial L}{\partial v_d}$ in $O(d^2m)$ time with $O(d/m+m)$ sequential matrix multiplications.*

Proof. **Correctness.** FastH computes gradients by the same equations as [17], so in most cases, we show correctness by clarifying how FastH computes the same thing, albeit faster.

Consider $\frac{\partial L}{\partial X}$ computed in Step 1:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial A_{d/m+1}} = P_{d/m}^T \cdots P_1^T \frac{\partial L}{\partial A_1}$$
$$= H_d^T \cdots H_1^T \frac{\partial L}{\partial A_1}. \qquad eq. (2)$$

This is the same as that computed in [17].

Consider Step 2. Both $\frac{\partial L}{\partial \hat{v}_j}$ and $\frac{\partial L}{\partial \widehat{A}_j}$ are computed as done in [17]. $\widehat{A}_{j+1}$ is computed using Equation (4) similar to backpropagation without storing activations [5], but using the fact that $\widehat{H}_j^T = \widehat{H}_j^{-1}$.

Time Complexity. In Step 1, the for loop performs d/m matrix multiplications. Due to the WY decomposition $P_i^T = (I - 2WY^T)^T = I - 2YW^T$ which can be multiplied on $\frac{\partial L}{\partial A_i} \in \mathbb{R}^{d\times m}$ in $O(dm^2)$ time since $W, Y \in \mathbb{R}^{d\times m}$. The computation is repeated d/m times, and take a total of $O(d^2m)$ time.

Step 2 line 12 in Algorithm 3 performs two Householder matrix multiplications which take $O(dm)$ time, see Equation (4). In line 13, the gradient is computed by Equation (5), this sum also takes $O(dm)$ time to compute. Both computations on line 12 and 13 are repeated $d/m \cdot m$ times, see line 8 and line 11. Therefore, the total time is $O(d^2m)$.

Number of Sequential Operations. Step 1 performs $O(d/m)$ sequential matrix operations. Lines 11-14 of Step 2 perform $O(m)$ sequential matrix multiplications. Since each iteration of line 8-15 is run in parallel, the algorithm performs no more than $O(d/m + m)$ sequential matrix multiplications. □

Algorithm 3 FastH Backward

```
1: Input: $A_1, ..., A_{d/m+1}, P_1, ..., P_{d/m}$ and $\frac{\partial L}{\partial A_1}$.
2: Output: $\frac{\partial L}{\partial X}$ and $\frac{\partial L}{\partial v_k}$ for all $k$ where $H_k = I - 2\frac{v_k v_k^T}{||v_k||_2^2}$.
3: // Step 1
4: for $i = 1$ to $d/m$ do sequentially
5:     $\frac{\partial L}{\partial A_{i+1}} = P_i^T \frac{\partial L}{\partial A_i}$ eq. (3).                ▷ $O(dm^2)$
6: end for
7: // Step 2
8: for $i = 1$ to $d/m$ do in parallel
9:     Let $\frac{\partial L}{\partial \widehat{A}_1} = \left(\frac{\partial L}{\partial A_i}\right)$.
10:    To ease notation, let $P_i = \widehat{H}_1 \cdots \widehat{H}_m$.
11:    for $j = 1$ to $m$ do
12:        Compute $\widehat{A}_{j+1}, \frac{\partial L}{\partial \widehat{A}_j}$ see eq. (4).                ▷ $O(dm)$
13:        Compute $\frac{\partial L}{\partial \hat{v}_j}$ using $\widehat{A}_{j+1}, \frac{\partial L}{\partial \widehat{A}_j}$, eq. (5).       ▷ $O(dm)$
14:    end for
15: end for
16: return $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial A_{d/m+1}}$ and $\frac{\partial L}{\partial v_k}$ for all $k = 1, ..., d$.
```

8.2 Comparing Running Time

This subsection clarifies how the matrix exponential and the Cayley map was used in combination with the SVD reparameterization from [17]. It also provides further details on the exact computations we timed in the experiment. These details were left out of the main article as they require the introduction of some notation regarding a *reparameterization function*.

Let $V \in \mathbb{R}^{d\times d}$ be a weight matrix and let ϕ be a function that reparameterizes V so $\phi(V)$ is orthogonal, and we can perform gradient descent wrt. V. The Householder decomposition can be used to construct such a function ϕ, by letting the columns of V be Householder vectors and $\phi(V)$ be the product of the resulting Householder matrices.

There exist alternative ways of constructing ϕ which does not rely on the Householder decomposition. For example, the matrix exponential approach where $\phi_{exp}(V) = e^V$ and the Cayley map approach where $\phi_C(V) = (I - V)(I + V)^{-1}$ [2].

We record the joint time it takes to compute $\phi(V)X$ and the gradients wrt. V and X for a dummy input $X \in \mathbb{R}^{d\times M}$. To simplify the gradient computation of V, we use a dummy gradient $G \in \mathbb{R}^{d\times M}$ st. the gradient wrt. V is $[\frac{\partial\phi(V)\cdot X}{\partial V}]^T G$. It might be useful to think of G as the gradient that arises by back-propagating through a Neural Network.

Both the dummy input and the dummy gradient have normally distributed entries $X_{ij}, G_{ij} \sim N(0, 1)$.

Implementation Details. The parallel algorithm from [17] halted for larger values of d. The failing code was not part of the main computation. This allowed us to remove the failing code and still get a good estimate of the running time of the parallel algorithm. We emphasize that removing the corresponding code makes the parallel algorithm faster. The experiments thus demonstrated that FastH is faster than a lower bound on the running time of the parallel algorithm.

8.3 Using the SVD to Compute Matrix Operations

This section requires first reading Section 4.1 and Section 4.2. Recall that we, in Section 4.2, want to measure the total time it takes to compute both the matrix operation, the forward pass and the gradient computations. For example, with matrix inversion, we want to compute the matrix operation Σ^{-1}, the forward pass $V\Sigma^{-1}U^T X$ and the gradient computations wrt V, Σ, U, X.

The time of the forward pass and gradient computations is no more than two multiplications and two gradient computations, which is exactly two times what we measured in Section 4.1. We re-used those measurements, and add the time it takes to compute the matrix operation, e.g., Σ^{-1}.

Over Estimating the Time of FastH. The matrix exponential and the Cayley map require one orthogonal matrix instead of two, i.e., $U\Sigma U^T$ instead of $U\Sigma V^T$. The WY decomposition then only needs to be computed for U and not both U and V. By re-using the data, we measure the time of two orthogonal matrices, this thus estimates an upper-bound of the real running time of FastH.

4.4 [4] One Reflection Suffice

ONE REFLECTION SUFFICE

Alexander Mathiasen **Frederik Hvilshøj**

ABSTRACT

Orthogonal weight matrices are used in many areas of deep learning. Much previous work attempt to alleviate the additional computational resources it requires to constrain weight matrices to be orthogonal. One popular approach utilizes *many* Householder reflections. The only practical drawback is that many reflections cause low GPU utilization. We mitigate this final drawback by proving that *one* reflection is sufficient if the reflection is computed by an auxiliary neural network.

1 INTRODUCTION

Orthogonal matrices have shown several benefits in deep learning, with successful applications in Recurrent Neural Networks, Convolutional Neural Networks and Normalizing Flows. One popular approach can represent any $d \times d$ orthogonal matrix using d Householder reflections (Mhammedi et al., 2017). The only practical drawback is low GPU utilization, which happens because the d reflections needs to be evaluated sequentially (Mathiasen et al., 2020). Previous work often increases GPU utilization by using $k \ll d$ reflections (Tomczak & Welling, 2016; Mhammedi et al., 2017; Zhang et al., 2018; Berg et al., 2018). Using fewer reflections limits the orthogonal transformations the reflections can represent, yielding a trade-off between representational power and computation time. This raises an intriguing question: can we circumvent the trade-off and attain full representational power without sacrificing computation time?

We answer this question with a surprising "yes." The key idea is to use an auxiliary neural network to compute a different reflection for each input. We prove that *one* such "auxiliary reflection" can represent any number of normal reflections.

1.1 OUR RESULTS

The Householder reflection of $x \in \mathbb{R}^d$ around $v \in \mathbb{R}^d$ can be represented by a matrix $H(v) \in \mathbb{R}^{d\times d}$.

$$H(v)x = \left(I - 2\frac{vv^T}{||v||^2}\right)x.$$

An auxiliary reflection uses a Householder matrix $H(v)$ with $v = n(x)$ for a neural network n.

$$f(x) = H(n(x))x = \left(I - 2\frac{n(x)n(x)^T}{||n(x)||^2}\right)x.$$

One auxiliary reflection can represent any composition of Householder reflections. We prove this claim even when we restrict the neural network $n(x)$ to have a single linear layer $n(x) = Wx$ for $W \in \mathbb{R}^{d\times d}$ such that $f(x) = H(Wx)x$.

Theorem 1. *For any k Householder reflections $U = H(v_1)\cdots H(v_k)$ there exists a neural network $n(x) = Wx$ with $W \in \mathbb{R}^{d\times d}$ such that $f(x) = H(Wx)x = Ux$ for all $x \in \mathbb{R}^d\backslash\{0\}$.*

Previous work (Mhammedi et al., 2017; Zhang et al., 2018) often employ $k \ll d$ reflections and compute Ux as k sequential Householder reflections $H(v_1)\cdots H(v_k)\cdot x$ with weights $V = (v_1 \ \cdots \ v_k)$. It is the sequential evaluation of these Householder reflections that cause low GPU utilization (Mathiasen et al., 2020), so lower values of k increase GPU utilization but decrease representational power. Theorem 1 states that it is sufficient to evaluate a single auxiliary reflection $H(Wx)x$ instead of k reflections $H(v_1)\cdots H(v_k)\cdot x$, thereby gaining high GPU utilization while retaining the full representational power of any number of reflections.

The use of auxiliary reflections is straightforward for Fully Connected Neural Networks and Recurrent Neural Networks, however, we needed additional ideas to support auxiliary reflections in Normalizing Flows. In particular, we developed further theory concerning the inverse and Jacobian of $f(x) = H(Wx)x$. Note f is invertible if there exists an unique x given $y = H(Wx)x$ and W.

Theorem 2. *Let $f(x) = H(Wx)x$ with $f(0) := 0$, then f is invertible on $\mathbb{R}^d$ with $d > 2$ if $W = W^T$ and has eigenvalues which satisfy $3/2 \cdot \lambda_{\min}(W) > \lambda_{\max}(W)$.*

Finally, we present a matrix formula for the Jacobian of the auxiliary reflection $f(x) = H(Wx)x$. This matrix formula is used in our proof of Theorem 2, but it also allows us to simplify the Jacobian determinant (Lemma 1) which is needed when training Normalizing Flows.

Theorem 3. *The Jacobian of $f(x) = H(Wx)x$ is:*

$$J = H(Wx)A - 2\frac{Wxx^TW}{||Wx||^2} \quad \textit{where} \quad A = I - 2\frac{x^TW^Tx}{||Wx||^2}W.$$

We prove Theorem 1 in Appendix A.1.1 while Theorems 2 and 3 are proved in Section 2.

2 Normalizing Flows

2.1 Background

Let $z \sim N(0,1)^d$ and f be an invertible neural network. Then $f^{-1}(z) \sim P_{model}$ defines a model distribution for which we can compute likelihood of $x \sim P_{data}$ (Dinh et al., 2015).

$$\log p_{model}(x) = \log p_z(f(x)) + \log\left|\det\left(\frac{\partial f(x)}{\partial x}\right)\right| \tag{1}$$

This allows us to train invertible neural networks as generative models by maximum likelihood. Previous work demonstrates how to construct invertible neural networks and efficiently compute the log jacobian determinant (Dinh et al., 2017; Kingma & Dhariwal, 2018; Ho et al., 2019).

2.2 Invertibility and Jacobian Determinant (Proof Sketch)

To use auxiliary reflections in Normalizing Flows, we need invertibility. That is, for every $y \in \mathbb{R}^d$ there must exist a unique $x \in \mathbb{R}^d$ so $f(x) = H(Wx)x = y$.[1] We find that f is invertible if its Jacobian determinant is non-zero for all x in $S^{d-1} = \{x \in \mathbb{R}^d \mid \|x\| = 1\}$.

Theorem 4. *Let $f(x) = H(Wx)x$ with $f(0) := 0$, then f is invertible on $\mathbb{R}^d$ with $d > 2$ if the Jacobian determinant of f is non-zero for all $x \in S^{d-1}$ and W is invertible.*

The Jacobian determinant of $H(Wx)x$ takes the following form.

Lemma 1. *The Jacobian determinant of $f(x) = H(Wx)x$ is:*

$$-\det(A)\left(1 + 2\frac{v^TA^{-1}u}{||u||^2}\right) \textit{ where } v^T = x^TW, u = Wx \textit{ and } A = I - 2\frac{x^TW^Tx}{||Wx||^2}W.$$

It is then sufficient that $\det(A) \neq 0$ and $1 + 2v^TA^{-1}u/||u||^2 \neq 0$. We prove that this happens if $W = W^T$ with eigenvalues $3/2 \cdot \lambda_{\min}(W) > \lambda_{\max}(W)$. This can be achieved with $W = I + VV^T$ if we guarantee $\sigma_{\max}(VV^T) < 1/2$ by spectral normalization (Miyato et al., 2018). Combining these results yields Theorem 2.

Theorem 2. *Let $f(x) = H(Wx)x$ with $f(0) := 0$, then f is invertible on $\mathbb{R}^d$ with $d > 2$ if $W = W^T$ and has eigenvalues which satisfy $3/2 \cdot \lambda_{\min}(W) > \lambda_{\max}(W)$.*

Computing the Inverse. In practice, we use Newton's method to compute x so $H(Wx)x = y$. Figure 1 show reconstructions $n^{-1}(n(x)) = x$ for an invertible neural network n with auxiliary reflections using Newton's method, see Appendix A.2.1 for details.

[1]Note that we do not know $H(Wx)$ so we cannot trivially compute $x = H(Wx)^{-1}y = H(Wx)y$.

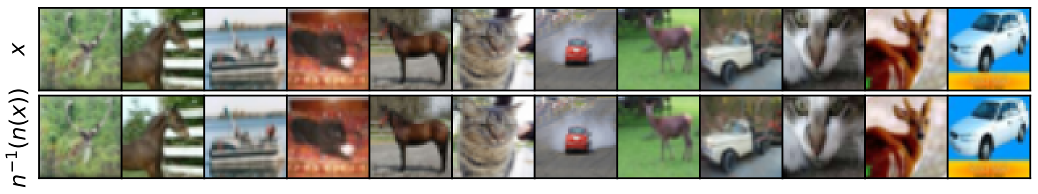


Figure 1: CIFAR10 (Krizhevsky et al., 2009) images x and reconstructions $n^{-1}(n(x))$ for an invertible neural network n called Glow (Kingma & Dhariwal, 2018). The network uses auxiliary reflections and we compute their inverse using Newton's method, see Appendix A.2.1 for details.

2.3 Proofs

The goal of this section is to prove that $f(x) = H(Wx)x$ is invertible. Our proof strategy has two parts. Section 2.3.1 first shows f is invertible if it has non-zero Jacobian determinant. Section 2.3.2 then presents an expression for the Jacobian determinant, Lemma 1, and prove the expression is non-zero if $W = W^T$ and $3/2 \cdot \lambda_{\min}(W) > \lambda_{\min}(W)$.

2.3.1 Non-Zero Jacobian Determinant Implies Invertibility

In this section, we prove that $f(x) = H(Wx)x$ is invertible on $\mathbb{R}^d$ if f has non-zero Jacobian determinant. To simplify matters, we first prove that invertibility on S^{d-1} implies invertibility on $\mathbb{R}^d$. Informally, invertibility on S^{d-1} is sufficient because $H(Wx)$ is scale invariant, i.e., $H(c \cdot Wx) = H(Wx)$ for all $c \neq 0$. This is formalized by Lemma 2.

Lemma 2. *If $f(x) = H(Wx)x$ is invertible on S^{d-1} it is also invertible on $\mathbb{R}^d \backslash \{0\}$.*

Proof. Assume that $f(x)$ is invertible on S^{d-1}. Pick any $y' \in \mathbb{R}^d$ such that $||y'|| = c$ for any $c > 0$. Our goal is to compute x' such that $H(Wx')x' = y'$. By normalizing, we see $y'/\|y'\| \in S^{d-1}$. We can then use the inverse f^{-1} on $y'/\|y'\|$ to find x such that $H(Wx)x = y'/\|y\|$. The result is then $x' = x\|y\|$ since $H(Wx')x' = H(Wx)x||y|| = y$ due to scale invariance of $H(Wx)$. □

The main theorem we use to prove invertibility on S^{d-1} is a variant of *Hadamards global function inverse theorem* from (Krantz & Parks, 2012). On a high-level, Hadamard's theorem says that a function is invertible if it has non-zero Jacobian determinant and satisfies a few additional conditions. It turns out that these additional conditions are meet by any continuously differentiable function $f(x)$ when (in the notation of Theorem 5) $M_1 = M_2 = S^{d-1}$.

Theorem 5. *(Krantz & Parks, 2012, 6.2.8) Let M_1 and M_2 be smooth, connected N-dimensional manifolds and let $f : M_1 \to M_2$ be continuously differentiable. If (1) f is proper, (2) the Jacobian of f is non-zero, and (3) M_2 is simply connected, then f is invertible.*

For $M_1 = M_2 = S^{d-1}$ the additional conditions are met if f is continuously differentiable.

Corollary 1. *Let $f : S^{d-1} \to S^{d-1}$ with $d > 2$ be continuously differentiable with non-zero Jacobian determinant, then f is invertible.*

Proof. Note that S^{d-1} is smooth and simply connected if $d > 2$ (Lee, 2013). Continuous functions on S^{d-1} are proper. We conclude f is invertible on S^{d-1} by Theorem 5. □

We now show that $f(x) = H(Wx)x$ is continuously differentiable on S^{d-1}.

Lemma 3. *The function $f(x) = H(Wx)x$ is continuously differentiable on S^{d-1} if W is invertible.*

Proof. Compositions of continuously differentiable functions are continuously differentiable by the chain rule. All the functions used to construct $H(Wx)x$ are continuously differentiable, except the division. However, the only case where division is not continuously differentiable is when $||Wx|| = 0$. Since W is invertible, $||Wx|| = 0$ iff $x = 0$. But $0 \notin S^{d-1}$ and we conclude f is continuously differentiable on S^{d-1}. □

Theorem 4. *Let $f(x) = H(Wx)x$ with $f(0) := 0$, then f is invertible on $\mathbb{R}^d$ with $d > 2$ if the Jacobian determinant of f is non-zero for all $x \in S^{d-1}$ and W is invertible.*

Proof. By Lemma 3, we see f is continuously differentiable since W is invertible, which by Corollary 1 means f is invertible on S^{d-1} if f has non-zero Jacobian determinant on S^{d-1}. By Lemma 2, we get that f is invertible on $\mathbb{R}^d$ if it has non-zero Jacobian on S^{d-1}. □

2.3.2 Enforcing Non-Zero Jacobian Determinant

The goal of this section is to present conditions on W that ensures the Jacobian determinant of $f(x)$ is non-zero for all $x \in S^{d-1}$. We first present a matrix formula for the Jacobian of f in Theorem 3. By using the *matrix determinant lemma*, we get a formula for the Jacobian determinant in Lemma 1. By investigating when this expression can be zero, we finally arrive at Lemma 4 which states that the Jacobian determinant is non-zero (and f thus invertible) if $W = W^T$ and $3/2 \cdot \lambda_{\text{min}} > \lambda_{\text{max}}$.

Theorem 3. *The Jacobian of $f(x) = H(Wx)x$ is:*

$$J = H(Wx)A - 2\frac{Wxx^TW}{||Wx||^2} \quad \textit{where} \quad A = I - 2\frac{x^TW^Tx}{||Wx||^2}W.$$

See Appendix A.2.2 for PyTorch implementation of J and a test case against PyTorch autograd.

Proof. The (i,j)'th entry of the Jacobian determinant is, by definition,

$$\frac{\partial(x - 2\cdot\frac{Wxx^TW^Tx}{||Wx||^2})_i}{\partial x_j} = \mathbb{1}_{i=j} - 2\cdot\frac{\partial(Wx)_i\cdot\frac{x^TW^Tx}{||Wx||^2}}{\partial x_j}.$$

Then, by the product rule, we get

$$\begin{aligned}\frac{\partial(Wx)_i\cdot\frac{x^TW^Tx}{||Wx||^2}}{\partial x_j} &= \frac{\partial(Wx)_i}{\partial x_j}\cdot\frac{x^TW^Tx}{||Wx||^2} + (Wx)_i\cdot\frac{\partial\frac{x^TW^Tx}{||Wx||^2}}{\partial x_j}\\ &= W_{ij}\cdot\frac{x^TW^Tx}{||Wx||^2} \qquad + (Wx)_i\cdot\frac{\partial x^TW^Tx\cdot\frac{1}{||Wx||^2}}{\partial x_j}.\end{aligned}$$

The remaining derivative can be found using the product rule.

$$\frac{\partial x^TW^Tx\cdot\frac{1}{||Wx||^2}}{\partial x_j} = \frac{\partial x^TW^Tx}{\partial x_j}\cdot\frac{1}{||Wx||^2} + x^TW^Tx\cdot\frac{\partial\frac{1}{||Wx||^2}}{\partial x_j}.$$

First, (Petersen & Pedersen, 2012) equation (81) gives $\frac{\partial x^TW^Tx}{\partial x_j} = ((W^T + W)x)_j$. Second $||Wx||^{-2}$ can be found using the chain rule:

$$\begin{aligned}\frac{\partial(||Wx||^2)^{-1}}{\partial x_j} &= \frac{\partial(||Wx||^2)^{-1}}{\partial||Wx||^2}\frac{\partial||Wx||^2}{\partial x_j}\\ &= -\frac{1}{||Wx||^4}\left(\frac{\partial x^TW^TWx}{\partial x}\right)_j\\ &= -\frac{1}{||Wx||^4}((W^TW + (W^TW)^T)x)_j \quad \text{(Petersen \& Pedersen, 2012, equ. 81)}\\ &= -\frac{1}{||Wx||^4}2(W^TWx)_j.\end{aligned}$$

Combining everything we get

$$J_{ij} = \mathbb{1}_{i=j} - 2\left[\frac{x^TW^Tx}{||Wx||^2}\cdot W_{ij} + (Wx)_i\left(\frac{1}{||Wx||^2}\cdot((W^T+W)x)_j - \frac{2x^TW^Tx}{||Wx||^4}\cdot(W^TWx)_j\right)\right].$$

In matrix notation, this translates into the following, if we let $A = I - 2 \cdot \frac{x^T W^T x}{||Wx||^2} W$.

$$\begin{aligned}
J &= I - 2\left[\frac{x^T W^T x}{||Wx||^2} \cdot W + Wx\left(\frac{1}{||Wx||^2} \cdot x^T(W + W^T) - \frac{2x^T W^T x}{||Wx||^4} \cdot x^T W^T W\right)\right] \\
&= I - 2 \cdot \frac{x^T W^T x}{||Wx||^2} \cdot W - 2 \cdot \frac{Wxx^T W}{||Wx||^2} - 2 \cdot \frac{Wxx^T W^T}{||Wx||^2}\left(I - 2 \cdot \frac{x^T W^T x}{||Wx||^2} W\right) \\
&= A - 2 \cdot \frac{Wxx^T W}{||Wx||^2} - 2 \cdot \frac{Wxx^T W^T}{||Wx||^2} A \\
&= \left(I - 2 \cdot \frac{Wxx^T W^T}{||Wx||^2}\right) A - 2 \cdot \frac{Wxx^T W}{||Wx||^2} = H(Wx)A - 2 \cdot \frac{Wxx^T W}{||Wx||^2}.
\end{aligned}$$

This concludes the proof. □

Theorem 3 allows us to write J as a rank one update $M + ab^T$ for $a, b \in \mathbb{R}^d$, which can be used to simplify $\det(J)$ as stated in the following lemma.

Lemma 1. *The Jacobian determinant of $f(x) = H(Wx)x$ is:*

$$-\det(A)\left(1 + 2\frac{v^T A^{-1} u}{||u||^2}\right) \text{ where } v^T = x^T W, u = Wx \text{ and } A = I - 2\frac{x^T W^T x}{||Wx||^2} W.$$

Proof. The *matrix determinant lemma* allows us to write $\det(M + ab^T) = \det(M)(1 + b^T M^{-1} a)$. Let $M = H(Wx)A$ and $b^T = -2 \cdot x^T W/||Wx||^2$ and $a = Wx$. The Jacobian J from Theorem 3 is then $J = M + ab^T$. The determinant of J is then:

$$\begin{aligned}
\det(J) &= \det(M)(1 + b^T M^{-1} a) \\
&= \det(H(Wx) \cdot A)\left(1 - 2\frac{x^T W (H(Wx) \cdot A)^{-1} Wx}{||Wx||^2}\right) \\
&= -\det(A)\left(1 + 2\frac{x^T W A^{-1} Wx}{||Wx||^2}\right).
\end{aligned}$$

This is true because $H(Wx)^{-1} = H(Wx)$, $H(Wx) \cdot Wx = -Wx$ and $\det(H(Wx)) = -1$. □

We can now use Lemma 1 to investigate when the Jacobian determinant is non-zero. In particular, the Jacobian determinant must be non-zero if both $\det(A) \neq 0$ and $1 + 2v^T A^{-1} u/||u||^2 \neq 0$. In the following lemma, we prove that both are non-zero if $W = W^T$ and $3/2 \cdot \lambda_{\min} > \lambda_{\max}$.

Lemma 4. *Let $W = W^T$ and $3/2 \cdot \lambda_{\min} > \lambda_{\max}$ then $\lambda_i(A^{-1}) < -1/2$ for A from Lemma 1. These conditions imply that $\det(A) \neq 0$ and $1 + 2v^T A^{-1} u/||u||^2 \neq 0$ with v^T, u from Lemma 1*

Proof. We first show that the inequality $3/2 \cdot \lambda_{\min}(W) > \lambda_{\max}(W)$ implies $\lambda_i(A^{-1}) < -1/2$.

$$\lambda_i(A^{-1}) = \frac{1}{\lambda_i(A)} = \frac{1}{1 - 2\frac{x^T W^T x}{||Wx||^2}\lambda_i(W)}$$

If $\gamma_i := \frac{x^T W^T x}{||Wx||^2} \cdot \lambda_i(W) \in (1/2, 3/2)$ we get that $1/(1 - 2\gamma_i) \in (-\infty, -1/2)$ so $\lambda_i(A^{-1}) < -1/2$. If we let $y := Wx$ we get $\frac{x^T W^T x}{||Wx||^2} = \frac{y^T W^{-1} y}{||y||^2}$. This is the *Rayleigh quotient* of W^{-1} at y, which for $W = W^T$ is within $[\lambda_{\min}(W^{-1}), \lambda_{\max}(W^{-1})]$. Therefore $\gamma_i \in [\frac{1}{\lambda_{\max}(W)}, \frac{1}{\lambda_{\min}(W)}] \cdot \lambda_i(W)$. Note first that $\gamma_{\min} \leq 1$ and $\gamma_{\max} \geq 1$. It is left to show that $\gamma_{\min} \geq \lambda_{\min}/\lambda_{\max} > 1/2$ and $\gamma_{\max} \leq \lambda_{\max}/\lambda_{\min} < 3/2$. Both conditions on eigenvalues are met if $3/2 \cdot \lambda_{\min} > \lambda_{\max}$.

We now want to show that $\det(A) \neq 0$ and $1 + 2v^T A^{-1} u/||u||^2 \neq 0$. First, notice that $\det(A) = \prod_{i=1}^d \lambda_i(A) \neq 0$ since $\lambda_i(A) < -1/2$. Second, note that $W = W^T$ implies that the v^T from Lemma 1 can be written as $v^T = x^T W = x^T W^T = u^T$. This means we only need to ensure $u^T A^{-1} u/||u||^2$, the Rayleigh quotient of A^{-1} at u, is different to $-1/2$. But $W = W^T$ implies $A = A^T$ because $A = I - 2x^T W^T x/||Wx||^2 \cdot W$. The Rayleigh quotient is therefore bounded by $[\lambda_{\min}(A^{-1}), \lambda_{\max}(A^{-1})]$, which means it is less than $-1/2$ since $\lambda_i(A^{-1}) < -1/2$. We can then conclude that also $1 + 2v^T A^{-1} u/||u||^2 = 1 + 2u^T A^{-1} u/||u||^2 < 1 + 2 \cdot -1/2 = 0$. □

So $\det(J) \neq 0$ by Lemma 4 and Lemma 1, which by Theorem 4 implies invertibility (Theorem 2).

Remark. Note that the constraints $W = W^T$ and $3/2 \cdot \lambda_{\min} > \lambda_{\max}$ were introduced only to guarantee $\det(A) \neq 0$ and $1 + 2v^T A^{-1} u/||u||^2 \neq 0$. Any argument or constraints on W that ensures $\det(A) \cdot (1 + v^T A^{-1} u/||u||^2) \neq 0$ are thus sufficient to conclude $f(x)$ is invertible.

3 Related Work

Orthogonal Weight Matrices. Orthogonal weight matrices have seen widespread use in deep learning. For example, they have been used in Normalizing Flows (Hoogeboom et al., 2019), Variational Auto Encoders (Berg et al., 2018), Recurrent Neural Networks (Mhammedi et al., 2017) and Convolutional Neural Networks (Bansal et al., 2018).

Different Approaches. There are several ways of constraining weight matrices to remain orthogonal. For example, previous work have used Householder reflections (Mhammedi et al., 2017), the Cayley map (Lezcano-Casado & Martínez-Rubio, 2019) and the matrix exponential (Casado, 2019). These approaches are sometimes referred to as *hard orthogonality constraints*, as opposed to *soft orthogonality constraints*, which instead provide approximate orthogonality by using, e.g., regularizers like $||WW^T - I||_F$ (see (Bansal et al., 2018) for a comprehensive review).

Reflection Based Approaches. The reflection based approaches introduce sequential computations, which is, perhaps, their main limitation. Authors often address this by reducing the number of reflections, as done in, e.g., (Tomczak & Welling, 2016; Mhammedi et al., 2017; Berg et al., 2018). This is sometimes undesirable, as it limits the expressiveness of the orthogonal matrix. This motivated previous work to construct algorithms that increase parallelization of Householder products, see, e.g., (Mathiasen et al., 2020; Likhosherstov et al., 2020).

Similar Ideas. Normalizing Flows have been used for variational inference, see, e.g., (Tomczak & Welling, 2016; Berg et al., 2018). Their use of reflections is very similar to auxiliary reflections, however, there is a very subtle difference which has fundamental consequences. For a full appreciation of this difference, the reader might want to consult the schematic in (Tomczak & Welling, 2016, Figure 1), however, we hope that the text below clarifies the high-level difference.

Recall that auxiliary reflections compute $H(Wx)x$ so $H(Wx)$ can depend on x. In contrast, the previous work on variational inference instead compute $H(v)z$ where v and z both depend on x. This limits $H(v)$ in that it can not explicitly depend on z. While this difference is subtle, it means our proof of Theorem 1 does not hold for reflections as used in (Tomczak & Welling, 2016).

References

Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can We Gain More From Orthogonality Regularizations in Training Deep Networks? In *NeurIPS*, 2018.

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester Normalizing Flows for Variational Inference. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Mario Lezcano Casado. Trivializations for Gradient-Based Optimization on Manifolds. In *NeurIPS*, 2019.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. In *ICLR, Workshop Proceedings*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Estimation using Real NVP. In *ICLR*, 2017.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *ICML*, 2019.

Emiel Hoogeboom, Rianne Van Den Berg, and Max Welling. Emerging Convolutions for Generative Normalizing Flows. In *ICML*, 2019.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In *NeurIPS*, 2018.

Steven G Krantz and Harold R Parks. *The Implicit Function Theorem: History, Theory, and Applications*. Springer Science & Business Media, 2012.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. 2009.

John M Lee. Smooth Manifolds. In *Introduction to Smooth Manifolds*. 2013.

Mario Lezcano-Casado and David Martínez-Rubio. Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. *ICML*, 2019.

Valerii Likhosherstov, Jared Davis, Krzysztof Choromanski, and Adrian Weller. CWY Parametrization for Scalable Learning of Orthogonal and Stiefel Matrices. *arXiv preprint arXiv:2004.08675*, 2020.

Alexander Mathiasen, Frederik Hvilshøj, Jakob Rødsgaard Jørgensen, Anshul Nasery, and Davide Mottin. Faster Orthogonal Parameterization with Householder Matrices. In *ICML, Workshop Proceedings*, 2020.

Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections. In *ICML*, 2017.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *ICLR*, 2018.

Kaare Brandt Petersen and Michael Syskind Pedersen. The Matrix Cookbook, 2012. Technical University of Denmark, Version 20121115.

Jakub M Tomczak and Max Welling. Improving Variational Auto-Encoders using Householder Flow. *arXiv preprint arXiv:1611.09630*, 2016.

Ruye Wang. Lecture Notes: Householder Transformation and QR Decomposition, 2015. URL `http://fourier.eng.hmc.edu/e176/lectures/NM/node10.html`.

Jiong Zhang, Qi Lei, and Inderjit Dhillon. Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization. In *ICML*, 2018.

A Appendix

A.1 Proofs

A.1.1 Theorem 1

Our proof of Theorem 1 use Lemma 5 which we state below.

Theorem 1. *For any k Householder reflections $U = H(v_1)\cdots H(v_k)$ there exists a neural network $n(x) = Wx$ with $W \in \mathbb{R}^{d\times d}$ such that $f(x) = H(Wx)x = Ux$ for all $x \in \mathbb{R}^d\backslash\{0\}$.*

Proof. Let $W = I - U$ then $H(Wx)x = H(x - Ux)x = Ux$ for all $x \in \mathbb{R}^d$ since $||Ux|| = ||x||$. □

Lemma 5. *Let $||x|| = ||y||$ then $H(x - y)x = y$.*

Proof. The result is elementary, see, e.g., (Wang, 2015). For completeness, we derive it below.

$$\begin{aligned}
H(x-y)x &= x - 2\frac{(x-y)(x-y)^T}{||x-y||^2}x \\
&= x - 2\frac{xx^T + yy^T - xy^T - yx^T}{x^Tx + y^Ty - 2x^Ty}x \\
&= x - 2\frac{xx^Tx + yy^Tx - xy^Tx - yx^Tx}{x^Tx + y^Ty - 2x^Ty} \\
&= x - 2\frac{x||x|| + yy^Tx - xy^Tx - y||x||}{2||x||^2 - 2x^Ty} \\
&= x - \frac{(x-y)||x||^2 + (y-x)(y^Tx)}{||x||^2 - x^Ty} \\
&= x - \frac{(x-y)||x||^2 + (y-x)(y^Tx)}{||x||^2 - x^Ty} \\
&= x - \frac{(x-y)(||x||^2 - x^Ty)}{||x||^2 - x^Ty} \\
&= x - (x-y) = y
\end{aligned}$$

□

A.2 PyTorch Examples and Test Cases

To ease the workload on reviewers, we opted to use small code snippets that can be copied into www.colab.research.google.com and run in a few seconds without installing any dependencies. Some PDF viewers do not copy line breaks, we found viewing the PDF in Google Chrome works.

A.2.1 Test Case: Inverse using Newton's Method

Given y we compute x such that $H(Wx)x = y$ using Newton's method. To be concrete, the code below contains a toy example where $x \in \mathbb{R}^4$ and $W = I + VV^T/(2 \cdot \sigma_{\max}(VV^T)) \in \mathbb{R}^{4\times4}$. The particular choice of W makes $H(Wx)x$ invertible, because $\lambda_i(W) = 1 + \lambda_i(VV^T) = 1 + \sigma_i(VV^T) \in [1, 3/2)$ because VV^T is positive definite. Any possible way of choosing the eigenvalues in the range $[1, 3/2)$ guarantees that $3/2 \cdot \lambda_{\min} > \lambda_{\max}$ which implies invertibility by Theorem 2.

```python
import torch
print("torch version: ", torch.__version__)
torch.manual_seed(42)
d = 4
# Create random test-case.
I = torch.eye(d)
V = torch.zeros((d, d)).uniform_()
x = torch.zeros((d, 1)).uniform_()
W = I + V @ V.T / torch.svd(V @ V.T)[1].max()
# Define the function f(x)=H(Wx)x.
def H(v): return torch.eye(d) - 2 * v @ v.T / (v.T @ v)
def f(x): return H(W @ x ) @ x
# Print input and output
print("x\t\t", x.data.view(-1).numpy())
print("f(x)\t", f(x).data.view(-1).numpy())
print("")

# Use Newtons Method to compute inverse.
y  = f(x)
xi = y
for i in range(10):
  print("[%.2i/%.2i]"%(i+1, 10), xi.data.view(-1).numpy())
  # Compute Jacobian using Theorem 3.
  A = torch.eye(d) - 2* (xi.T @ W.T @ xi) / torch.norm(W @ xi)**2 * W
  J = -2*W @ xi @ xi.T @ W/torch.norm(W@xi)**2 + H(W @ xi) @ A
  xi = xi - torch.inverse(J) @ (f(xi)- y)
assert torch.allclose(xi, x, atol=10**(-7))
print("The two vectors are torch.allclose")
```

```
torch version:  1.6.0+cu101
x     [0.8854429  0.57390445 0.26658005 0.62744915]
f(x)    [-0.77197534 -0.49936318 -0.5985155  -0.6120473 ]

[01/10] [-0.77197534 -0.49936318 -0.5985155  -0.6120473 ]
[02/10] [ 0.72816867  0.78074205 -0.02241153  1.0435152 ]
[03/10] [0.7348436  0.6478982  0.14960966 0.8003925 ]
[04/10] [0.8262452 0.6155189 0.2279686 0.6997254]
[05/10] [0.8765415 0.5831212 0.2592551 0.640691 ]
[06/10] [0.8852093  0.5742159  0.26631045 0.6278922 ]
[07/10] [0.88543946 0.5739097  0.26658094 0.62744874]
[08/10] [0.88544315 0.57390547 0.2665805  0.6274475 ]
[09/10] [0.885443   0.57390594 0.26658088 0.6274466 ]
[10/10] [0.8854408  0.57390743 0.2665809  0.6274484 ]
The two vectors are torch.allclose
```

Figure 1. Figure 1 contains reconstructions $n^{-1}(n(x))$ of the variant of Glow (Kingma & Dhariwal, 2018). The Glow variant has 1x1 convolutions with auxiliary reflections, i.e., for an input $x \in \mathbb{R}^{c\times h\times w}$ where (c, h, w) are (channels, heigh, width) it computes $z_{:,i,j} = H(Wx_{:,i,j})x_{:,i,j} \in \mathbb{R}^c$ where $i = 1, ..., h$ and $j = 1, ..., w$. Computing the inverse required computing the inverse of the auxiliary 1x1 convolutions, i.e., compute $x_{:,i,j}$ given W and $z_{:,i,j}$ $\forall i, j$. The weights were initialized as done in the above toy example.

A.2.2 Test Case: Jacobian and Autograd

```
import torch
print("torch version: ", torch.__version__)
torch.manual_seed(42)

# Create random test-case.
d = 4
W = torch.zeros((d, d)).uniform_(-1, 1)
x = torch.zeros((d, 1)).uniform_(-1, 1)
I = torch.eye(d)

# Compute Jacobian using autograd.
def H(v): return I - 2 * v @ v.T / (v.T @ v)
def f(x): return H(W @ x ) @ x
J     = torch.autograd.functional.jacobian(f, x)[:, 0, :, 0]
print(J)

# Compute Jacobian using Lemma 4.
A  = I - 2* (x.T @ W.T @ x) / torch.norm(W @ x)**2 * W
J_ = H(W @ x) @ A -2*W @ x @ x.T @ W/torch.norm(W@x)**2
print(J_)

# Test the two matrices are close.
assert torch.allclose(J, J_, atol=10**(-5))
print("The two matrices are torch.allclose")
```

```
torch version:  1.6.0+cu101
tensor([[ 0.2011, -1.4628,  0.7696, -0.5376],
        [ 0.3125,  0.6518,  0.7197, -0.5997],
        [-1.0764,  0.8388,  0.0020, -0.1107],
        [-0.8789, -0.3006, -0.4591,  1.3701]])
tensor([[ 0.2011, -1.4628,  0.7696, -0.5376],
        [ 0.3125,  0.6518,  0.7197, -0.5997],
        [-1.0764,  0.8388,  0.0020, -0.1107],
        [-0.8789, -0.3006, -0.4591,  1.3701]])
The two matrices are torch.allclose
```

Bibliography

[1] Allan Grønlund, Kasper Green Larsen, and Alexander Mathiasen. "Optimal Minimal Margin Maximization with Boosting." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4392–4401.

[2] Allan Grønlund et al. "Margin-Based Generalization Lower Bounds for Boosted Classifiers." In: *NeurIPS*. 2019.

[3] Alexander Mathiasen et al. "What if Neural Networks had SVDs?" In: *NeurIPS*. 2020.

[4] Alexander Mathiasen and Frederik Hvilshøj. "One Reflection Suffice." In: *Manuscript* (2021).

[5] Aarhus University - Graduate School of Natural Sciences. *Rules and Regulations Executive Order on the Ph. D.* `https://phd.nat.au.dk/for-phd-students/rules-regulations/`. 2021.

[6] *AI and Compute*. `https://openai.com/blog/ai-and-compute/`.

[7] Martin Arjovsky, Amar Shah, and Yoshua Bengio. "Unitary Evolution Recurrent Neural Networks." In: *ICML*. 2016.

[8] Peter Bartlett et al. "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods." In: *The Annals of Statistics* (1998).

[9] Jens Behrmann et al. "Invertible Residual Networks." In: *ICML*. 2019.

[10] Kristin P Bennett, Ayhan Demiriz, and John Shawe-Taylor. "A Column Generation Algorithm for Boosting." In: *ICML*. 2000.

[11] Christian Bischof and Charles Van Loan. "The WY Representation for Products of Householder Matrices." In: *SIAM Journal on Scientific and Statistical Computing* (1987).

[12] Leo Breiman. "Prediction Games and Arcing Algorithms." In: *Neural computation* (1999).

[13] Tom B. Brown et al. "Language Models are Few-Shot Learners." In: *NeurIPS*. 2020.

[14] Jia Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[15] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-linear Independent Components Estimation." In: *ICLR, Workshop Track Proceedings*. 2015.

[16] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP." In: *ICLR Conference Trakc Proceedings*. 2017.

[17] Drucker, Harris and Burges, Christopher J. C. and Kaufman, Linda and Smola, Alex and Vapnik, Vladimir. "Support Vector Regression Machines." In: *NeurIPS*. 1996.

[18] Yoav Freund, Robert Schapire, and Naoki Abe. "A Short Introduction to Boosting." In: *Journal-Japanese Society For Artificial Intelligence* (1999).

[19] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting." In: *Journal of computer and system sciences* (1997).

[20] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The Elements of Statistical Learning*. 2001.

[21] Wei Gao and Zhi-Hua Zhou. “On the Doubt about Margin Explanation of Boosting.” In: *Artificial Intelligence* (2013).

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[23] Allan Grønlund, Lior Kamma, and Kasper Green Larsen. “Margins are Insufficient for Explaining Gradient Boosting.” In: *NeurIPS*. 2020.

[24] Adam J Grove and Dale Schuurmans. “Boosting in the Limit: Maximizing the Margin of Learned Ensembles.” In: *AAAI/IAAI*. 1998.

[25] Kyle Helfrich, Devin Willmott, and Qiang Ye. “Orthogonal Recurrent Neural Networks with Scaled Cayley Transform.” In: *ICML*. 2018.

[26] Danny Hernandez and Tom B. Brown. “Measuring the Algorithmic Efficiency of Neural Networks.” In: *CoRR* (2020).

[27] John Jumper et al. “High Accuracy Protein Structure Prediction Using Deep Learning.” In: *CASP* (2020).

[28] Anatolii Alekseevich Karatsuba and Yu P Ofman. In: *Multiplication of Many-Digital Numbers by Automatic Computers*. Proceedings oft he USSR Academy of Sciences. 1962.

[29] Tero Karras et al. “Training Generative Adversarial Networks with Limited Data.” In: *NeurIPS*. 2020.

[30] Guolin Ke et al. “Lightgbm: A highly efficient gradient boosting decision tree.” In: *NeurIPS* (2017).

[31] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative Flow with Invertible 1x1 Convolutions.” In: *NeurIPS*. 2018.

[32] Philip Klein and Neal Young. “On the Number of Iterations for Dantzig-Wolfe Optimization and Packing-Covering Approximation Algorithms.” In: *International Conference on Integer Programming and Combinatorial Optimization*. 1999.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet Classification with Deep Convolutional Neural Networks.” In: *NeurIPS* (2012).

[34] Mario Lezcano-Casado and David Martınez-Rubio. “Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group.” In: *ICML*. 2019.

[35] Valerii Likhosherstov et al. “CWY Parametrization: a Solution for Parallelized Optimization of Orthogonal and Stiefel Matrices.” In: *International Conference on Artificial Intelligence and Statistics*. 2021.

[36] Llew Mason et al. “Boosting Algorithms as Gradient Descent.” In: *NeurIPS*. Ed. by S. Solla, T. Leen, and K. Müller. 2000.

[37] Zakaria Mhammedi et al. “Efficient Orthogonal Parametrisation of Recurrent Neural Networks using Householder Reflections.” In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2401–2409.

[38] Takeru Miyato et al. “Spectral Normalization for Generative Adversarial Networks.” In: *ICLR*. 2018.

[39] Gordon E Moore. “Cramming more Components onto Integrated Circuits.” In: *Proceedings of the IEEE* (1998).

[40] Jiazhong Nie et al. “Open Problem: Lower Bounds for Boosting with Hadamard Matrices.” In: *Conference on Learning Theory*. 2013.

[41] Adam Paszke et al. “Automatic Differentiation in PyTorch.” In: (2017).

[42] Daniil Polykovskiy et al. “Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models.” In: *Frontiers in Pharmacology* (2020).

[43] Gunnar Rätsch and Manfred K Warmuth. “Maximizing the Margin with Boosting.” In: *International Conference on Computational Learning Theory*. 2002.

[44] Gunnar Rätsch, Manfred K Warmuth, and John Shawe-Taylor. “Efficient Margin Maximizing with Boosting.” In: *Journal of Machine Learning Research* (2005).

[45] Lev Reyzin and Robert E Schapire. “How Boosting the Margin can also Boost Classifier Complexity.” In: *ICML*. 2006.

[46] F. Rotella and I. Zambettakis. “Block Householder Transformation for Parallel QR Factorization.” In: *Applied Mathematics Letters* (1999).

[47] The Danish Ministry of Research and Education. *Executive Order on the Ph. D. Education at Universities. Translated from Danish, see ’Kapitel 5 §11’*. `https://www.retsinformation.dk/eli/lta/2013/1039`. 2013.

[48] Frank Uhlig. “Constructive Ways for Generating (Generalized) Real Orthogonal Matrices as Products of (Generalized) Symmetries.” In: *Linear Algebra and its Applications* (2001).

[49] Vladimir N Vapnik and A Ya Chervonenkis. “On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities.” In: *Measures of Complexity*. 2015.

[50] Liwei Wang et al. “On the margin explanation of boosting algorithms.” In: *COLT*. 2008.

[51] Changyi Xiao and Ligang Liu. “Generative Flows with Matrix Exponential.” In: *ICML*. 2020.

[52] Jiong Zhang, Qi Lei, and Inderjit Dhillon. “Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization.” In: *ICML*. 2018.