Journal of Machine Learning Research 14 (2013) 3073-3096 Submitted 1/12; Revised 4/13; Published 10/13

Algorithms and Hardness Results for Parallel Large Margin Learning

Philip M. Long PLONG@MICROSOFT.COM
Microsoft
1020 Enterprise Way
Sunnyvale, CA 94089

Rocco A. Servedio ROCCO@CS.COLUMBIA.EDU
Department of Computer Science
Columbia University
1214 Amsterdam Ave., Mail Code: 0401
New York, NY 10027

Editor: Yoav Freund

Abstract

We consider the problem of learning an unknown large-margin halfspace in the context of parallel computation, giving both positive and negative results.

As our main positive result, we give a parallel algorithm for learning a large-margin halfspace, based on an algorithm of Nesterov's that performs gradient descent with a momentum term. We show that this algorithm can learn an unknown γ-margin halfspace over n dimensions using $n \cdot \mathrm{poly}(1/\gamma)$ processors and running in time $\tilde{O}(1/\gamma) + O(\log n)$. In contrast, naive parallel algorithms that learn a γ-margin halfspace in time that depends polylogarithmically on n have an inverse quadratic running time dependence on the margin parameter γ.

Our negative result deals with boosting, which is a standard approach to learning large-margin halfspaces. We prove that in the original PAC framework, in which a weak learning algorithm is provided as an oracle that is called by the booster, boosting cannot be parallelized. More precisely, we show that, if the algorithm is allowed to call the weak learner multiple times in parallel within a single boosting stage, this ability does not reduce the overall number of successive stages of boosting needed for learning by even a single stage. Our proof is information-theoretic and does not rely on unproven assumptions.

Keywords: PAC learning, parallel learning algorithms, halfspace learning, linear classifiers

1. Introduction

One of the most fundamental problems in machine learning is learning an unknown halfspace from labeled examples that satisfy a *margin constraint,* meaning that no example may lie too close to the separating hyperplane. In this paper we consider large-margin halfspace learning in the PAC (probably approximately correct) setting of learning from random examples: there is a target halfspace $f(\mathbf{x}) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x})$, where $\mathbf{w}$ is an unknown unit vector, and an unknown probability distribution $\mathcal{D}$ over the unit ball $\mathbf{B}_n = \{\mathbf{x} \in \mathbf{R}^n : \|\mathbf{x}\|_2 \leq 1\}$ which is guaranteed to have support contained in the set $\{\mathbf{x} \in \mathbf{B}_n : |\mathbf{w} \cdot \mathbf{x}| \geq \gamma\}$ of points that have Euclidean margin at least γ relative to the separating hyperplane. (Throughout this paper we refer to such a combination of target halfspace f and distribution $\mathcal{D}$ as a γ-*margin halfspace*.) The learning algorithm is given access to labeled examples $(\mathbf{x}, f(\mathbf{x}))$

where each $\mathbf{x}$ is independently drawn from $\mathcal{D}$, and it must with high probability output a $(1-\varepsilon)$-accurate hypothesis, that is, a hypothesis $h : \mathbf{R}^n \to \{-1,1\}$ that satisfies $\Pr_{\mathbf{x}\sim\mathcal{D}}[h(\mathbf{x}) \neq f(\mathbf{x})] \leq \varepsilon$.

One of the earliest, and still most important, algorithms in machine learning is the perceptron algorithm (Block, 1962; Novikoff, 1962; Rosenblatt, 1958) for learning a large-margin halfspace. The perceptron is an online algorithm but it can be easily transformed to the PAC setting described above (Vapnik and Chervonenkis, 1974; Littlestone, 1989; Freund and Schapire, 1999); the resulting PAC algorithms run in $\mathrm{poly}(n, \frac{1}{\gamma}, \frac{1}{\varepsilon})$ time, use $O(\frac{1}{\varepsilon\gamma^2})$ labeled examples in $\mathbf{R}^n$, and learn an unknown n-dimensional γ-margin halfspace to accuracy $1-\varepsilon$.

A motivating question: achieving perceptron's performance in parallel? The last few years have witnessed a resurgence of interest in highly efficient parallel algorithms for a wide range of computational problems in many areas including machine learning (Workshop, 2009, 2011). So a natural goal is to develop an efficient parallel algorithm for learning γ-margin halfspaces that matches the performance of the perceptron algorithm. A well-established theoretical notion of efficient parallel computation (see, for example, the text by Greenlaw et al. (1995) and the many references therein) is that an efficient parallel algorithm for a problem with input size N is one that uses $\mathrm{poly}(N)$ processors and runs in parallel time $\mathrm{polylog}(N)$. Since the input to the perceptron algorithm is a sample of $\mathrm{poly}(\frac{1}{\varepsilon}, \frac{1}{\gamma})$ labeled examples in $\mathbf{R}^n$, we naturally arrive at the following:

> **Main Question:** Is there a learning algorithm that uses $\mathrm{poly}(n, \frac{1}{\gamma}, \frac{1}{\varepsilon})$ processors and runs in time $\mathrm{poly}(\log n, \log\frac{1}{\gamma}, \log\frac{1}{\varepsilon})$ to learn an unknown n-dimensional γ-margin halfspace to accuracy $1-\varepsilon$?

Following Vitter and Lin (1992), we use a CRCW PRAM model of parallel computation. This abstracts away issues like communication and synchronization, allowing us to focus on the most fundamental issues. Also, as did Vitter and Lin (1992), we require that an efficient parallel learning algorithm's hypothesis must be efficiently evaluatable in parallel, since otherwise all the computation required to run any polynomial-time learning algorithm could be "offloaded" onto evaluating the hypothesis. Because halfspace learning algorithms may be sensitive to issues of numerical precision, these are not abstracted away in our model; we assume that numbers are represented as rationals.

As noted by Freund (1995) (see also Lemma 2 below), the existence of efficient boosting algorithms such as the algorithms of Freund (1995) and Schapire (1990) implies that any PAC learning algorithm can be efficiently parallelized in terms of its dependence on the accuracy parameter ε: more precisely, any PAC learnable class C of functions can be PAC learned to accuracy $1-\varepsilon$ using $O(1/\varepsilon)$ processors by an algorithm whose running time dependence on ε is $O(\log\left(\frac{1}{\varepsilon}\right) \cdot \mathrm{poly}(\log\log(1/\varepsilon)))$, by boosting an algorithm that learns to accuracy (say) $9/10$. We may thus equivalently restate the above question as follows.

> **Main Question (simplified):** Is there a learning algorithm that uses $\mathrm{poly}(n, \frac{1}{\gamma})$ processors and runs in time $\mathrm{poly}(\log n, \log\frac{1}{\gamma})$ to learn an unknown n-dimensional γ-margin halfspace to accuracy $9/10$?

The research reported in this paper is inspired by this question, which we view as a fundamental open problem about the abilities and limitations of efficient parallel learning algorithms.

Algorithm	No. processors	Running time
naive parallelization of perceptron	$\mathrm{poly}(n, 1/\gamma)$	$\tilde{O}(1/\gamma^2) + O(\log n)$
(Servedio, 2003)	$\mathrm{poly}(n, 1/\gamma)$	$\tilde{O}(1/\gamma^2) + O(\log n)$
poly-time linear programming (Blumer et al., 1989)	1	$\mathrm{poly}(n, \log(1/\gamma))$
this paper (algorithm of Section 2)	$n \cdot \mathrm{poly}(1/\gamma)$	$\tilde{O}(1/\gamma) + O(\log n)$

Table 1: Bounds on various parallel algorithms for learning a γ-margin halfspace over $\mathbf{R}^n$.

1.1 Relevant Prior Results

Table 1 summarizes the running time and number of processors used by various parallel algorithms to learn a γ-margin halfspace over $\mathbf{R}^n$.

The naive parallelization of perceptron in the first line of the table is an algorithm that runs for $O(1/\gamma^2)$ stages. In each stage it processes all of the $O(1/\gamma^2)$ examples simultaneously in parallel, identifies one that causes the perceptron algorithm to update its hypothesis vector, and performs this update. Since the examples are n-dimensional this can be accomplished in $O(\log(n/\gamma))$ time using $O(n/\gamma^2)$ processors; the mistake bound of the online perceptron algorithm is $1/\gamma^2$, so this gives a running time bound of $\tilde{O}(1/\gamma^2) \cdot \log n$. We do not see how to obtain parallel time bounds better than $O(1/\gamma^2)$ from recent analyses of other algorithms based on gradient descent (Collins et al., 2002; Dekel et al., 2011; Bradley et al., 2011), some of which use assumptions incomparable in strength to the γ-margin condition studied here.

The second line of the table corresponds to a similar naive parallelization of the boosting-based algorithm of Servedio (2003) that achieves perceptron-like performance for learning a γ-margin halfspace. This algorithm boosts for $O(1/\gamma^2)$ stages over a $O(1/\gamma^2)$-size sample. At each stage of boosting this algorithm computes a real-valued weak hypothesis based on the vector average of the (normalized) examples weighted according to the current distribution; since the sample size is $O(1/\gamma^2)$ this can be done in $O(\log(n/\gamma))$ time using $\mathrm{poly}(n, 1/\gamma)$ processors. Since the boosting algorithm runs for $O(1/\gamma^2)$ stages, the overall running time bound is $\tilde{O}(1/\gamma^2) \cdot \log n$. (For both this algorithm and the perceptron the time bound can be improved to $\tilde{O}(1/\gamma^2) + O(\log n)$ as claimed in the table by using an initial random projection step. We show how to do this in Section 2.3.)

The third line of the table, included for comparison, is simply a standard sequential algorithm for learning a halfspace based on polynomial-time linear programming executed on one processor (Blumer et al., 1989; Karmarkar, 1984).

In addition to the results summarized in the table, we note that efficient parallel algorithms have been developed for some simpler PAC learning problems such as learning conjunctions, disjunctions, and symmetric Boolean functions (Vitter and Lin, 1992). Bshouty et al. (1998) gave efficient parallel PAC learning algorithms for some geometric constant-dimensional concept classes. Collins et al. (2002) presented a family of boosting-type algorithms that optimize Bregman divergences by updating a collection of parameters in parallel; however, their analysis does not seem to imply that the algorithms need fewer than $\Omega(1/\gamma^2)$ stages to learn γ-margin halfspaces.

In terms of negative results for parallel learning, Vitter and Lin (1992) showed that (under a complexity-theoretic assumption) there is no parallel algorithm using $\mathrm{poly}(n)$ processors and $\mathrm{polylog}(n)$ time that constructs a halfspace hypothesis that is consistent with a given linearly separable data set of n-dimensional labeled examples. This does not give a negative answer to the

main question for several reasons: first, the main question allows any hypothesis representation that can be efficiently evaluated in parallel, whereas the hardness result requires the hypothesis to be a halfspace. Second, the main question allows the algorithm to use $\text{poly}(n, 1/\gamma)$ processors and to run in $\text{poly}(\log n, \log \frac{1}{\gamma})$ time, whereas the hardness result of Vitter and Lin (1992) only rules out algorithms that use $\text{poly}(n, \log \frac{1}{\gamma})$ processors and run in $\text{poly}(\log n, \log\log \frac{1}{\gamma})$ time. Finally, the main question allows the final hypothesis to err on up to (say) 5% of the points in the data set, whereas the hardness result of Vitter and Lin (1992) applies only to algorithms whose hypotheses correctly classify all points in the data set.

Finally, we note that the main question has an affirmative answer if it is restricted so that either the number of dimensions n or the margin parameter γ is fixed to be a constant (so the resulting restricted question asks whether there is an algorithm that uses polynomially many processors and polylogarithmic time in the remaining parameter). If γ is fixed to a constant then either of the first two entries in Table 1 gives a $\text{poly}(n)$-processor, $O(\log n)$-time algorithm. If n is fixed to a constant then the efficient parallel algorithm of Alon and Megiddo (1994) for linear programming in constant dimension can be used to learn a γ-margin halfspace using $\text{poly}(1/\gamma)$ processors in $\text{polylog}(1/\gamma)$ running time (see also Vitter and Lin, 1992, Theorem 3.4).

1.2 Our Results

We give positive and negative results on learning halfspaces in parallel that are inspired by the main question stated above.

1.2.1 POSITIVE RESULTS

Our main positive result is a parallel algorithm for learning large-margin halfspaces, based on a rapidly converging gradient method due to Nesterov (2004), which uses $O(n \cdot \text{poly}(1/\gamma))$ processors to learn γ-margin halfspaces in parallel time $\tilde{O}(1/\gamma) + O(\log n)$ (see Table 1). (An earlier version of this paper (Long and Servedio, 2011) analyzed on algorithm based on interior-point methods from convex optimization and fast parallel algorithms for linear algebra, showing that it uses $\text{poly}(n, 1/\gamma)$ processors to learn γ-margin halfspaces in parallel time $\tilde{O}(1/\gamma) + O(\log n)$.) We are not aware of prior parallel algorithms that provably learn γ-margin halfspaces running in time polylogarithmic in n and subquadratic in $1/\gamma$.

We note that simultaneously and independently of the initial conference publication of our work (Long and Servedio, 2011), Soheili and Peña (2012) proposed a variant of the perceptron algorithm and shown that it terminates in $O\left(\frac{\sqrt{\log n}}{\gamma}\right)$ iterations rather than the $1/\gamma^2$ iterations of the original perceptron algorithm. Like our algorithm, the Soheili and Peña (2012) algorithm uses ideas of Nesterov (2005). Soheili and Peña (2012) do not discuss a parallel implementation of their algorithm, but since their algorithm performs an n-dimensional matrix-vector multiplication at each iteration, it appears that a parallel implementation of their algorithm would use $\Omega(n^2)$ processors and would have parallel running time at least $\Omega\left(\frac{(\log n)^{3/2}}{\gamma}\right)$ (assuming that multiplying a $n \times n$ matrix by an $n \times 1$ vector takes parallel time $\Theta(\log n)$ using n^2 processors). In contrast, our algorithm requires a linear number of processors as a function of n, and has parallel running time $\tilde{O}(1/\gamma) + O(\log n)$.[1]

1. We note also that Soheili and Peña (2012) analyze the number of iterations of their algorithm, and not the computation time. In particular, they do not deal with finite precision issues, whereas a significant portion of our analysis concerns

1.2.2 NEGATIVE RESULTS

By modifying our analysis of the algorithm we present, we believe that it may be possible to establish similar positive results for other formulations of the large-margin learning problem, including ones (see Shalev-Shwartz and Singer, 2010) that have been tied closely to weak learnability. In contrast, our main negative result is an information-theoretic argument that suggests that such positive parallel learning results cannot be obtained by boosting alone. We show that in a framework where the weak learning algorithm must be invoked as an oracle, boosting cannot be parallelized: being able to call the weak learner multiple times in parallel within a single boosting stage does not reduce the overall number of sequential stages of boosting that are required. We prove that any parallel booster must perform $\Omega(\log(1/\varepsilon)/\gamma^2)$ sequential stages of boosting a "black-box" γ-advantage weak learner to learn to accuracy $1-\varepsilon$ in the worst case; this extends an earlier $\Omega(\log(1/\varepsilon)/\gamma^2)$ lower bound of Freund (1995) for standard (sequential) boosters that can only call the weak learner once per stage.

2. An Algorithm Based on Nesterov's Algorithm

In this section we describe and analyze a parallel algorithm for learning a γ-margin halfspace. The algorithm of this section applies an algorithm of Nesterov (2004) that, roughly speaking, approximately minimizes a suitably smooth convex function to accuracy ε using $O(\sqrt{1/\varepsilon})$ iterative steps (Nesterov, 2004), each of which can be easily parallelized.

Directly applying the basic Nesterov algorithm gives us an algorithm that uses $O(n)$ processors, runs in parallel time $O(\log(n)\cdot(1/\gamma))$, and outputs a halfspace hypothesis that has constant accuracy. By combining the basic algorithm with random projection and boosting we get the following stronger result:

Theorem 1 *There is a parallel algorithm with the following performance guarantee: Let $f,\mathcal{D}$ define an unknown γ-margin halfspace over $\mathbf{B}_n$. The algorithm is given as input $\varepsilon,\delta>0$ and access to labeled examples $(\mathbf{x},f(\mathbf{x}))$ that are drawn independently from $\mathcal{D}$. It runs in*

$$O(((1/\gamma)\text{polylog}(1/\gamma)+\log(n))\log(1/\varepsilon)\text{poly}(\log\log(1/\varepsilon))+\log\log(1/\delta))$$

parallel time, uses

$$n\cdot\text{poly}(1/\gamma,1/\varepsilon,\log(1/\delta))$$

processors, and with probability $1-\delta$ it outputs a hypothesis h satisfying $\Pr_{\mathbf{x}\sim\mathcal{D}}[h(\mathbf{x})\neq f(\mathbf{x})]\leq\varepsilon$.

We assume that the value of γ is "known" to the algorithm, since otherwise the algorithm can use a standard "guess and check" approach trying $\gamma=1,1/2,1/4$, etc., until it finds a value that works.

Freund (1995) indicated how to parallelize his boosting-by-filtering algorithm. In Appendix A, we provide a detailed proof of the following lemma.

Lemma 2 (Freund, 1995) *Let $\mathcal{D}$ be a distribution over (unlabeled) examples. Let A be a parallel learning algorithm, and c_δ and c_ε be absolute positive constants, such that for all $\mathcal{D}'$ with*

such issues, in order to fully establish our claimed bounds on the number of processors and the parallel running time of our algorithms.

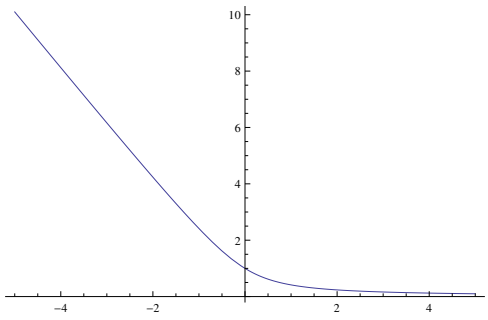

Figure 1: A plot of a loss function ϕ used in Section 2.

support$(\mathcal{D}') \subseteq$ *support*$(\mathcal{D})$*, given draws* $(x, f(x))$ *from* $\mathcal{D}'$*, with probability* c_δ*, A outputs a hypothesis with accuracy* $\frac{1}{2} + c_\varepsilon$ *(w.r.t.* $\mathcal{D}'$*) using* $\mathcal{P}$ *processors in* $\mathcal{T}$ *time. Then there is a parallel algorithm* B *that, given access to independent labeled examples* $(\boldsymbol{x}, f(\boldsymbol{x}))$ *drawn from* $\mathcal{D}$*, with probability* $1-\delta$*, constructs a* $(1-\varepsilon)$*-accurate hypothesis (w.r.t.* $\mathcal{D}$*) in* $O(\mathcal{T}\log(1/\varepsilon)\mathrm{poly}(\log\log(1/\varepsilon)) + \log\log(1/\delta))$ *time using* $\mathrm{poly}(\mathcal{P}, 1/\varepsilon, \log(1/\delta))$ *processors.*

In Section 2.1 we describe the basic way that Nesterov's algorithm can be used to find a halfspace hypothesis that approximately minimizes a smooth loss function over a set of γ-margin labeled examples. (This section has nothing to do with parallelism.) Then later we explain how this algorithm is used in the larger context of a parallel algorithm for halfspaces.

2.1 The Basic Algorithm

Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a data set of m examples labeled according to the target γ-margin halfspace f; that is, $y_i = f(\mathbf{x}_i)$ for all i.

We will apply Nesterov's algorithm to minimize a regularized loss as follows.
The loss part. For $z \in \mathbf{R}$ we define

$$\phi(z) = \sqrt{1+z^2} - z.$$

(See Figure 1 for a plot of ϕ.) For $\mathbf{v} \in \mathbf{R}^n$ we define

$$\Phi(\mathbf{v}) = \frac{1}{m}\sum_{t=1}^{m} \phi(y_t(\mathbf{v}\cdot\mathbf{x}_t)).$$

The regularization part. We define a regularizer

$$R(\mathbf{v}) = \gamma^2\|\mathbf{v}\|^2/100$$

where $\|\cdot\|$ denotes the 2-norm.

We will apply Nesterov's iterative algorithm to minimize the following function

$$\Psi(\mathbf{v}) = \Phi(\mathbf{v}) + R(\mathbf{v}).$$

Let $g(\mathbf{v})$ be the gradient of Ψ at $\mathbf{v}$. We will use the following algorithm, due to Nesterov (2004) (see Section 2.2.1), which we call A_{Nes}. The algorithm takes a single input parameter $\gamma > 0$.

Algorithm A_{Nes}:

- Set $\mu = \gamma^2/50$, $L = 51/50$.
- Initialize $\mathbf{v}_0 = \mathbf{z}_0 = \mathbf{0}$.
- For each $k = 0, 1, \ldots$, set
  - $\mathbf{v}_{k+1} = \mathbf{z}_k - \frac{1}{L} g(\mathbf{z}_k)$, and
  - $\mathbf{z}_{k+1} = \mathbf{v}_{k+1} + \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(\mathbf{v}_{k+1} - \mathbf{v}_k)$.

We begin by establishing various bounds on Ψ that Nesterov uses in his analysis of A_{Nes}.

Lemma 3 *The gradient g of Ψ has a Lipschitz constant at most $51/50$.*

Proof: We have

$$\frac{\partial \Psi}{\partial v_i} = \frac{1}{m}\sum_t \phi'(y_t(\mathbf{v}\cdot\mathbf{x}_t))y_t x_{t,i} + \gamma^2 v_i/50$$

and hence, writing $g(\mathbf{v})$ to denote the gradient of Ψ at $\mathbf{v}$, we have

$$g(\mathbf{v}) = \frac{1}{m}\sum_t \phi'(y_t(\mathbf{v}\cdot\mathbf{x}_t))y_t\mathbf{x}_t + \gamma^2\mathbf{v}/50.$$

Choose $\mathbf{r} \in \mathbf{R}^n$. Applying the triangle inequality, we have

$$\begin{aligned}
||g(\mathbf{v}) - g(\mathbf{r})|| &= \left\| \frac{1}{m}\sum_t (\phi'(y_t(\mathbf{v}\cdot\mathbf{x}_t)) - \phi'(y_t(\mathbf{r}\cdot\mathbf{x}_t)))y_t\mathbf{x}_t + \gamma^2(\mathbf{v}-\mathbf{r})/50 \right\| \\
&\leq \frac{1}{m}\sum_t ||(\phi'(y_t(\mathbf{v}\cdot\mathbf{x}_t)) - \phi'(y_t(\mathbf{r}\cdot\mathbf{x}_t)))y_t\mathbf{x}_t|| + \gamma^2||\mathbf{v}-\mathbf{r}||/50 \\
&\leq \frac{1}{m}\sum_t |\phi'(y_t(\mathbf{v}\cdot\mathbf{x}_t)) - \phi'(y_t(\mathbf{r}\cdot\mathbf{x}_t))| + \gamma^2||\mathbf{v}-\mathbf{r}||/50,
\end{aligned}$$

since each vector $\mathbf{x}_t$ has length at most 1. Basic calculus gives that ϕ'' is always at most 1, and hence

$$|\phi'(y_t(\mathbf{v}\cdot\mathbf{x}_t)) - \phi'(y_t(\mathbf{r}\cdot\mathbf{x}_t))| \leq |\mathbf{v}\cdot\mathbf{x}_t - \mathbf{r}\cdot\mathbf{x}_t| \leq ||\mathbf{v}-\mathbf{r}||,$$

again since $\mathbf{x}_t$ has length at most 1. The bound then follows from the fact that $\gamma^2 \leq 1$. ■

We recall the definition of strong convexity (Nesterov, 2004, pp. 63–64): a multivariate function q is μ-strongly convex if for all $\mathbf{v}, \mathbf{w}$ and all $\alpha \in [0,1]$ it holds that

$$q(\alpha\mathbf{v} + (1-\alpha)\mathbf{w}) \leq \alpha q(\mathbf{v}) + (1-\alpha)q(\mathbf{w}) - \frac{\mu\alpha(1-\alpha)||\mathbf{v}-\mathbf{w}||^2}{2}.$$

(For intuition's sake, it may be helpful to note that a suitably smooth q is μ-strongly convex if any restriction of q to a line has second derivative that is always at least μ.) We recall the fact that strongly convex functions have unique minimizers.

Lemma 4 *Ψ is μ-strongly convex.*

Proof: This follows directly from the fact that $\mu = \gamma^2/50$, Φ is convex, and $||\mathbf{v}||^2$ is 2-strongly convex. ■

Given the above, the following lemma is an immediate consequence of Theorem 2.2.3 of Nesterov's (2004) book. The lemma upper bounds the difference between $\Psi(\mathbf{v}_k)$, where $\mathbf{v}_k$ is the point computed in the k-th iteration of Nesterov's algorithm A_{Nes}, and the true minimum value of Ψ. A proof is in Appendix B.

Lemma 5 *Let* $\mathbf{w}$ *be the minimizer of* Ψ. *For each k, we have* $\Psi(\mathbf{v}_k) - \Psi(\mathbf{w}) \leq \frac{4L(1+\mu||\mathbf{w}||^2/2)}{(2\sqrt{L}+k\sqrt{\mu})^2}$.

2.2 The Finite Precision Algorithm

The algorithm analyzed in the previous subsection computes real numbers with infinite precision. Now we will analyze a finite precision variant of the algorithm, which we call A_{Nfp} (for "Nesterov finite precision").

(We note that d'Aspremont (2008), also analyzed a similar algorithm with an approximate gradient, but we were not able to apply his results in our setting because of differences between his assumptions and our needs. For example, the algorithm described by d'Aspremont (2008) assumed that optimization was performed over a compact set C, and periodically projected solutions onto C; it was not obvious to us how to parallelize this algorithm.)

We begin by writing the algorithm as if it took two parameters, γ and a precision parameter $\beta > 0$. The analysis will show how to set β as a function of γ. To distinguish between A_{Nfp} and A_{Nes} we use hats throughout our notation below.

Algorithm A_{Nfp}:

- Set $\mu = \gamma^2/50$, $L = 51/50$.
- Initialize $\hat{\mathbf{v}}_0 = \hat{\mathbf{z}}_0 = \mathbf{0}$.
- For each $k = 0, 1, \ldots,$
  - Let $\hat{\mathbf{r}}_k$ be such that $||\hat{\mathbf{r}}_k - \frac{1}{L}g(\hat{\mathbf{z}}_k)|| \leq \beta$. Set
  - $\hat{\mathbf{v}}_{k+1} = \hat{\mathbf{z}}_k - \hat{\mathbf{r}}_k$, and
  - $\hat{\mathbf{z}}_{k+1} = \hat{\mathbf{v}}_{k+1} + \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(\hat{\mathbf{v}}_{k+1} - \hat{\mathbf{v}}_k)$.

We discuss the details of exactly how this finite-precision algorithm is implemented, and the parallel running time required for such an implementation, at the end of this section.

Our analysis of this algorithm will proceed by quantifying how closely its behavior tracks that of the infinite-precision algorithm.

Lemma 6 *Let* $\mathbf{v}_0, \mathbf{v}_1, \ldots$ *be the sequence of points computed by the infinite precision version of Nesterov's algorithm, and* $\hat{\mathbf{v}}_0, \hat{\mathbf{v}}_1, \ldots$ *be the corresponding finite-precision sequence. Then for all k, we have* $||\mathbf{v}_k - \hat{\mathbf{v}}_k|| \leq \beta \cdot 7^k$.

Proof: Let $\hat{\mathbf{s}}_k = \hat{\mathbf{r}}_k - g(\hat{\mathbf{z}}_k)$. Our proof is by induction, with the additional inductive hypothesis that $||\mathbf{z}_k - \hat{\mathbf{z}}_k|| \leq 3\beta \cdot 7^k$.

The base case is trivially true.

We have

$$||\mathbf{v}_{k+1} - \hat{\mathbf{v}}_{k+1}|| = \left|\left|(\mathbf{z}_k - \frac{1}{L}g(\mathbf{z}_k)) - (\hat{\mathbf{z}}_k - (\frac{1}{L}g(\hat{\mathbf{z}}_k) + \hat{\mathbf{s}}_k)\right|\right|,$$

and, using the triangle inequality, we get

$$\begin{aligned}
||\mathbf{v}_{k+1} - \hat{\mathbf{v}}_{k+1}|| &\leq ||\mathbf{z}_k - \hat{\mathbf{z}}_k|| + \left|\left|\frac{1}{L}g(\mathbf{z}_k) - (\frac{1}{L}g(\hat{\mathbf{z}}_k) + \hat{\mathbf{s}}_k)\right|\right| \\
&\leq 3\beta \cdot 7^k + \left|\left|\frac{1}{L}g(\mathbf{z}_k) - (\frac{1}{L}g(\hat{\mathbf{z}}_k) + \hat{\mathbf{s}}_k)\right|\right| \\
&\leq 3\beta \cdot 7^k + \frac{1}{L}||g(\mathbf{z}_k) - g(\hat{\mathbf{z}}_k)|| + ||\hat{\mathbf{s}}_k|| \qquad \text{(triangle inequality)} \\
&\leq 3\beta \cdot 7^k + ||\mathbf{z}_k - \hat{\mathbf{z}}_k|| + ||\hat{\mathbf{s}}_k|| \qquad \text{(by Lemma 3)} \\
&\leq 3\beta \cdot 7^k + 3\beta \cdot 7^k + \beta \qquad \text{(by definition of } \hat{\mathbf{s}}_k) \\
&< \beta \cdot 7^{k+1}.
\end{aligned}$$

Also, we have

$$\begin{aligned}
||\mathbf{z}_{k+1} - \hat{\mathbf{z}}_{k+1}|| &= \left|\left|\frac{2}{1+\sqrt{\mu/L}}(\mathbf{v}_{k+1} - \hat{\mathbf{v}}_{k+1}) - \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(\mathbf{v}_k - \hat{\mathbf{v}}_k)\right|\right| \\
&\leq \left|\left|\frac{2}{1+\sqrt{\mu/L}}(\mathbf{v}_{k+1} - \hat{\mathbf{v}}_{k+1})\right|\right| + \left|\left|\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(\mathbf{v}_k - \hat{\mathbf{v}}_k)\right|\right| \\
&\leq 2||\mathbf{v}_{k+1} - \hat{\mathbf{v}}_{k+1}|| + ||\mathbf{v}_k - \hat{\mathbf{v}}_k|| \\
&\leq 2\beta \cdot 7^{k+1} + \beta \cdot 7^k \\
&\leq 3\beta \cdot 7^{k+1},
\end{aligned}$$

completing the proof. ■

2.3 Application to Learning

Now we are ready to prove Theorem 1. By Lemma 2 it suffices to prove the theorem in the case in which $\varepsilon = 7/16$ and $\delta = 1/2$.

We may also potentially reduce the number of variables by applying a random projection. We say that a *random projection matrix* is a matrix A chosen uniformly from $\{-1,1\}^{n\times d}$. Given such an A and a unit vector $\mathbf{w} \in \mathbf{R}^n$ (defining a target halfspace $f(\mathbf{x}) = \text{sign}(\mathbf{w}\cdot\mathbf{x})$), let $\mathbf{w}'$ denote the vector $(1/\sqrt{d})\mathbf{w}A \in \mathbf{R}^d$. After transformation by A the distribution $\mathcal{D}$ over $\mathbf{B}_n$ is transformed to a distribution $\mathcal{D}'$ over $\mathbf{R}^d$ in the natural way: a draw $\mathbf{x}'$ from $\mathcal{D}'$ is obtained by making a draw $\mathbf{x}$ from $\mathcal{D}$ and setting $\mathbf{x}' = (1/\sqrt{d})\mathbf{x}A$. We will use the following lemma, which is a slight variant of known lemmas (Arriaga and Vempala, 2006; Blum, 2006); we prove this exact statement in Appendix C.

Lemma 7 *Let $f(\mathbf{x}) = \text{sign}(\mathbf{w}\cdot\mathbf{x})$ and $\mathcal{D}$ define a γ-margin halfspace as described in the introduction. For $d = O((1/\gamma^2)\log(1/\gamma))$, a random $n \times d$ projection matrix A will with probability $99/100$ induce $\mathcal{D}'$ and $\mathbf{w}'$ as described above such that $\Pr_{\mathbf{x}'\sim\mathcal{D}'}\left[\left|\frac{\mathbf{w}'}{\|\mathbf{w}'\|}\cdot\mathbf{x}'\right| < \gamma/2 \quad or \quad \|\mathbf{x}'\|_2 > 2\right] \leq \gamma^4$.*

We assume without loss of generality that $\gamma = 1/\text{integer}$.

The algorithm first selects an $n \times d$ random projection matrix A where $d = O(\log(1/\gamma)/\gamma^2)$. This defines a transformation $\Phi_A : \mathbf{B}_n \to \mathbf{R}^d$ as follows: given $\mathbf{x} \in \mathbf{B}_n$, the vector $\Phi_A(\mathbf{x}) \in \mathbf{R}^d$ is obtained by

(i) rounding each $\mathbf{x}_i$ to the nearest integer multiple of $1/(4\lceil\sqrt{n/\gamma}\rceil)$; then

(ii) setting $\mathbf{x}' = \left(\frac{1}{2\sqrt{d}}\right)\mathbf{x}A$ (we scale down by an additional factor of two to get examples that are contained in the unit ball $\mathbf{B}_d$); and finally

(iii) rounding each $\mathbf{x}'_i$ to the nearest multiple of $1/(8\lceil d/\gamma\rceil)$.

Given $\mathbf{x}$ it is easy to compute $\Phi_A(\mathbf{x})$ using $O(n\log(1/\gamma)/\gamma^2)$ processors in $O(\log(n/\gamma))$ time. Let $\mathcal{D}'$ denote the distribution over $\mathbf{R}^d$ obtained by applying Φ_A to $\mathcal{D}$. Across all coordinates $\mathcal{D}'$ is supported on rational numbers with the same poly$(1/\gamma)$ common denominator. By Lemma 7, with probability $99/100$

$$\Pr_{\mathbf{x}' \sim \mathcal{D}'}\left[\left|\mathbf{x}' \cdot (\mathbf{w}'/\|\mathbf{w}'\|)\right| < \gamma' \stackrel{\text{def}}{=} \gamma/8 \quad \text{or} \quad \|\mathbf{x}'\|_2 > 1\right] \leq \gamma^4.$$

Our algorithm draws $c_0 d$ examples by sampling from $\mathcal{D}'$. Applying Lemma 7, we may assume without loss of generality that our examples have $d = O(\log(1/\gamma)/\gamma^2)$ and that the margin γ' after the projection is at least $\Theta(\gamma)$, and that all the coordinates of all the examples have a common denominator which is at most poly$(1/\gamma)$. Thus far the algorithm has used $O(\log(n/\gamma))$ parallel time and $O(n\log(1/\gamma)/\gamma^2)$ many processors.

Next, the algorithm applies A_{Nfp} from the previous section for K stages, where $K = \lceil c_1/\gamma'\rceil$ and $\beta = c_2 7^{-K}$. Here c_0, c_1, and c_2 are absolute positive constants; our analysis will show that there exist choices of these constants that give Theorem 1.

For our analysis, as before, let $\mathbf{w}$ be the minimizer of Ψ, and let $\mathbf{u}$ be a unit normal vector for the target halfspace $f(\mathbf{x}) = \text{sign}(\mathbf{u} \cdot \mathbf{x})$. (We emphasize that Ψ is now defined using the projected d-dimensional examples and with γ' in place of γ in the definition of the regularizer R.)

Our first lemma gives an upper bound on the optimal value of the objective function.

Lemma 8 $\Psi(\mathbf{w}) \leq 0.26$.

Proof Since $\mathbf{w}$ is the minimizer of Ψ we have $\Psi(\mathbf{w}) \leq \Psi(3\mathbf{u}/\gamma')$. In turn $\Psi(3\mathbf{u}/\gamma')$ is easily seen to be at most $\phi(3) + 9/100 \leq 0.26$, since every example has margin at least γ' with respect to $\mathbf{u}$. ■

Next, we bound the norm of $\mathbf{w}$.

Lemma 9 $||\mathbf{w}||^2 \leq 26/\gamma'^2$.

Proof: The definition of Ψ gives

$$||\mathbf{w}||^2 \leq 100\Psi(\mathbf{w})/\gamma'^2$$

and combining with Lemma 8 gives $||\mathbf{w}||^2 \leq 26/\gamma'^2$. ■

Now we can bound the objective function value of $\mathbf{v}_K$.

Lemma 10 *For c_1 a sufficiently large absolute constant, we have $\Psi(\mathbf{v}_K) \leq 2/5$.*

Proof: Plugging Lemma 9 into the RHS of Lemma 5 and simplifying, we get

$$\Psi(\mathbf{v}_K) - \Psi(\mathbf{w}) \leq \frac{751}{25(2\sqrt{51}+\gamma' K)^2}.$$

Applying Lemma 8, we get

$$\Psi(\mathbf{v}_K) \leq 0.26 + \frac{751}{25(2\sqrt{51}+\gamma' K)^2}.$$

from which the lemma follows. ■

Now we can bound $\mathbf{v}_K$ nearly the same way that we bounded $\mathbf{w}$:

Lemma 11 $||\mathbf{v}_K|| \leq 7/\gamma'$.

Proof: The argument is similar to the proof of Lemma 9, using Lemma 10 in place of Lemma 8. ■

Now we can bound the value of the objective function of the finite precision algorithm.

Lemma 12 *There exist absolute positive constants c_1, c_2 such that $\Psi(\hat{\mathbf{v}}_K) \leq 3/7$.*

Proof Because $\beta = c_2 7^{-\lceil c_1/\gamma' \rceil}$, Lemma 6 implies that $||\hat{\mathbf{v}}_K - \mathbf{v}_K|| \leq c_2$. Since ϕ has a Lipschitz constant of 2, so does Φ, and consequently we have that

$$\Phi(\hat{\mathbf{v}}_K) - \Phi(\mathbf{v}_K) \leq 2c_2. \tag{1}$$

Next, since Lemma 11 gives $||\mathbf{v}_K|| \leq 7/\gamma'$, and $||\hat{\mathbf{v}}_K - \mathbf{v}_K|| \leq c_2$, we have $||\hat{\mathbf{v}}_K|| \leq 7/\gamma' + c_2$, which in turn implies

$$\|\hat{\mathbf{v}}_K\|^2 - \|\mathbf{v}_K\|^2 \leq (7/\gamma' + c_2)^2 - (7/\gamma')^2 = 14c_2/\gamma' + c_2^2.$$

and thus

$$R(\hat{\mathbf{v}}_K) - R(\mathbf{v}_K) \leq \frac{14c_2\gamma'}{100} + \frac{(\gamma')^2 c_2^2}{100}.$$

Combining this with (1), we get that for c_2 less than a sufficiently small positive absolute constant, we have $\Psi(\hat{\mathbf{v}}_K) - \Psi(\mathbf{v}_K) < 3/7 - 2/5$, and combining with Lemma 10 completes the proof. ■

Finally, we observe that $\Psi(\hat{\mathbf{v}}_k)$ is an upper bound on the fraction of examples in the sample that are misclassified by $\hat{\mathbf{v}}_k$. Taking c_0 sufficiently large and applying standard VC sample complexity bounds, we have established the (ε, δ) PAC learning properties of the algorithm. (Recall from the start of this subsection that we have taken $\varepsilon = 7/16$ and $\delta = 1/2$.)

It remains to analyze the parallel time complexity of the algorithm. We have already analyzed the parallel time complexity of the initial random projection stage, and shown that we may take the finite-precision iterative algorithm A_{Nfp} to run for $O(1/\gamma)$ stages, so it suffices to analyze the parallel time complexity of each stage A_{Nfp}. We will show that each stage runs in parallel time $\text{polylog}(1/\gamma)$ and thus establish the theorem.

Recall that we have set $\beta = \Theta(7^{-K})$ and that $K = \Theta(1/\gamma)$. The invariant we maintain throughout each iteration k of algorithm A_{Nfp} is that each coordinate of $\hat{\mathbf{v}}_k$ is a $\text{poly}(K)$-bit rational number and each coordinate of $\hat{\mathbf{z}}_k$ is a $\text{poly}(K)$-bit rational number. It remains to show that given such values $\hat{\mathbf{v}}_k$ and $\hat{\mathbf{z}}_k$, in parallel time $\text{polylog}(1/\gamma)$ using $\log(1/\gamma)$ processors,

1. it is possible to compute each coordinate $g(\hat{\mathbf{z}}_k)_i$ to accuracy $2^{-100K}/\sqrt{d}$;
2. it is possible to determine a vector $\hat{\mathbf{r}}_k$ such that $\|\hat{\mathbf{r}}_k - g(\hat{\mathbf{z}}_k)\| \leq \beta$, and that each coefficient of the new value $\hat{\mathbf{v}}_{k+1} = \hat{\mathbf{z}}_k - \hat{\mathbf{r}}_k$ is again a poly(K)-bit rational number; and
3. it is possible to compute the new value $\hat{\mathbf{z}}_{k+1}$ and that each coordinate of $\hat{\mathbf{z}}_{k+1}$ is again a poly(K)-bit rational number.

We begin by analyzing the approximate computation of the gradient. Recall that

$$g(\mathbf{v}) = \frac{1}{m}\sum_t \phi'(y_t(\mathbf{v}\cdot\mathbf{x}_t))y_t\mathbf{x}_t + \gamma^2\mathbf{v}/50.$$

Note that

$$\phi'(z) = \frac{z}{\sqrt{1+z^2}} - 1.$$

To analyze the approximation of ϕ' we will first need a lemma about approximating the square root function efficiently in parallel. While related statements are known and our statement below can be proved using standard techniques, we have included a proof in Appendix D because we do not know a reference for precisely this statement.

Lemma 13 *There is an algorithm A_r that, given an L-bit positive rational number z and an L-bit positive rational number β as input, outputs $A_r(z)$ for which $|A_r(z) - \sqrt{z}| \leq \beta$ in* poly$(\log\log(1/\beta), \log L)$ *parallel time using* poly$(\log(1/\beta), L)$ *processors.*

Armed with the ability to approximate the square root, we can easily approximate ϕ'.

Lemma 14 *There is an algorithm A_p that, given an L-bit positive rational number z, and an L-bit positive rational number $\beta \leq 1/4$, outputs $A_p(z)$ for which $|A_p(z) - \phi'(z)| \leq \beta$ in at most* poly$(\log\log(1/\beta), \log L)$ *parallel time using* poly$(\log(1/\beta), L)$ *processors.*

Proof: Assume without loss of generality that $\beta \leq 1/4$. Then, because $\sqrt{1+z^2} \geq 1$, if an approximation s of $\sqrt{1+z^2}$ satisfies $|s - \sqrt{1+z^2}| \leq \beta/2^{L+1}$, then

$$\frac{1}{s} - \frac{1}{\sqrt{1+z^2}} \leq \beta/2^L.$$

Applying Lemma 13 and recalling the well-known fact that there are efficient parallel algorithms for division (see Beame et al., 1986) completes the proof. ■

Using this approximation for ϕ', and calculating the sums in the straightforward way, we get the required approximation $\hat{\mathbf{r}}_k$. We may assume without loss of generality that each component of $\hat{\mathbf{r}}_k$ has been rounded to the nearest multiple of $\beta/2$. Since each component of g has size at most 2, and the denominator of $\hat{\mathbf{r}}_k$ has $O(K)$ bits, $\hat{\mathbf{r}}_k$ in total requires at most $O(K)$ bits. We can assume without loss of generality that $\gamma^2/50$ is a perfect square, so multiplying the components of a vector by $\frac{1-\sqrt{\mu}}{1+\sqrt{\mu}}$ can be accomplished while adding $O(\log(1/\gamma))$ bits to each of their rational representations. Thus, a straightforward induction implies that each of the components of each of the denominators of $\mathbf{v}_k$ and $\mathbf{z}_k$ can be written with $k\log(1/\gamma) + O(1/\gamma) = O((1/\gamma)\log(1/\gamma))$ bits.

To bound the numerators of the components of $\mathbf{v}_k$ and $\mathbf{z}_k$, it suffices to bound the norms of $\mathbf{v}_k$ and $\mathbf{z}_k$. Lemma 11 implies that $||\mathbf{v}_k|| \leq 5/\gamma'$ and so Lemma 6 implies $||\hat{\mathbf{v}}_k|| \leq 5/\gamma' + 1$ which in turn directly implies $||\hat{\mathbf{z}}_k|| \leq 3(5/\gamma' + 1)$.

Thus, each iteration takes $O(\text{polylog}(1/\gamma))$ time, and there are a total of $O(1/\gamma)$ iterations. This completes the proof of Theorem 1.

3. Lower Bound for Parallel Boosting in the Oracle Model

Boosting is a widely used method for learning large-margin halfspaces. In this section we consider the question of whether boosting algorithms can be efficiently parallelized. We work in the original PAC learning setting (Valiant, 1984; Kearns and Vazirani, 1994; Schapire, 1990) in which a weak learning algorithm is provided as an oracle that is called by the boosting algorithm, which must simulate a distribution over labeled examples for the weak learner. Our main result for this setting is that boosting is inherently sequential; being able to call the weak learner multiple times in parallel within a single boosting stage does not reduce the overall number of sequential boosting stages that are required. In fact we show this in a very strong sense, by proving that a boosting algorithm that runs *arbitrarily* many copies of the weak learner in parallel in each stage cannot save *even one* stage over a sequential booster that runs the weak learner just once in each stage. This lower bound is unconditional and information-theoretic.

Below we first define the parallel boosting framework and give some examples of parallel boosters. We then state and prove our lower bound on the number of stages required by parallel boosters. A consequence of our lower bound is that $\Omega(\log(1/\varepsilon)/\gamma^2)$ stages of parallel boosting are required in order to boost a γ-advantage weak learner to achieve classification accuracy $1-\varepsilon$ no matter how many copies of the weak learner are used in parallel in each stage.

3.1 Parallel Boosting

Our definition of weak learning is standard in PAC learning, except that for our discussion it suffices to consider a single target function $f : X \to \{-1,1\}$ over a domain X.

Definition 15 *A* γ-advantage weak learner *L is an algorithm that is given access to a source of independent random labeled examples drawn from an (unknown and arbitrary) probability distribution $\mathcal{P}$ over labeled examples $\{(x, f(x))\}_{x\in X}$. L must*[2] *return a weak hypothesis $h : X \to \{-1,1\}$ that satisfies* $\Pr_{(x,f(x))\leftarrow\mathcal{P}}[h(x) = f(x)] \geq 1/2+\gamma$. *Such an h is said to have* advantage γ *w.r.t. $\mathcal{P}$.*

We fix $\mathcal{P}$ to henceforth denote the initial distribution over labeled examples; that is, $\mathcal{P}$ is a distribution over $\{(x, f(x))\}_{x\in X}$ where the marginal distribution $\mathcal{P}_X$ may be an arbitrary distribution over X.

Intuitively, a boosting algorithm runs the weak learner repeatedly on a sequence of carefully chosen distributions $\mathcal{P}_1, \mathcal{P}_2, \ldots$ to obtain weak hypotheses $h_1, h_2, \ldots$, and combines the weak hypotheses to obtain a final hypothesis h that has high accuracy under $\mathcal{P}$. We first give a definition that captures the idea of a "sequential" (non-parallel) booster, and then extend the definition to parallel boosters.

3.1.1 SEQUENTIAL BOOSTERS

We give some intuition to motivate our definition. In a normal (sequential) boosting algorithm, the probability weight that the $(t+1)$st distribution $\mathcal{P}_{t+1}$ puts on a labeled example $(x, f(x))$ may depend on the values of all the previous weak hypotheses $h_1(x), \ldots, h_t(x)$ and on the value of $f(x)$. No other dependence on x is allowed, since intuitively the only interface that the boosting algorithm should have with each data point is through its label and the values of the weak hypotheses. We

2. The usual definition of a weak learner would allow L to fail with probability δ. This probability can be made exponentially small by running L multiple times so for simplicity we assume there is no failure probability.

further observe that since the distribution $\mathcal{P}$ is the only source of labeled examples, a booster should construct the distribution $\mathcal{P}_{t+1}$ by somehow "filtering" examples drawn from $\mathcal{P}$ based on the values $h_1(x),\ldots,h_t(x),f(x)$. We thus define a sequential booster as follows:

Definition 16 (Sequential booster) *A T-stage* sequential boosting algorithm *is defined by a sequence $\alpha_1,\ldots,\alpha_T$ of functions $\alpha_t : \{-1,1\}^t \to [0,1]$ and a (randomized) Boolean function $h : \{-1,1\}^T \to \{-1,1\}$. In the t-th stage of boosting, the distribution $\mathcal{P}_t$ over labeled examples that is given to the weak learner by the booster is obtained from $\mathcal{P}$ by doing rejection sampling according to α_t. More precisely, a draw from $\mathcal{P}_t$ is made as follows: draw $(x,f(x))$ from $\mathcal{P}$ and compute the value $p_x := \alpha_t(h_1(x),\ldots,h_{t-1}(x),f(x))$. With probability p_x accept $(x,f(x))$ as the output of the draw from $\mathcal{P}_t$, and with the remaining $1-p_x$ probability reject this $(x,f(x))$ and try again. In stage t the booster gives the weak learner access to $\mathcal{P}_t$ as defined above, and the weak learner generates a hypothesis h_t that has advantage at least γ w.r.t. $\mathcal{P}_t$. Together with $h_1,\ldots,h_{t-1}$, this h_t enables the booster to give the weak learner access to $\mathcal{P}_{t+1}$ in the next stage.*

After T stages, weak hypotheses $h_1,\ldots,h_T$ have been obtained from the weak learner. The final hypothesis *of the booster is $H(x) := h(h_1(x),\ldots,h_T(x))$, and its* accuracy *is*

$$\min_{h_1,\ldots,h_T} \Pr_{(x,f(x))\leftarrow\mathcal{P}}[H(x)=f(x)],$$

where the min is taken over all sequences $h_1,\ldots,h_T$ of T weak hypotheses subject to the condition that each h_t has advantage at least γ w.r.t. $\mathcal{P}_t$.

Many PAC-model boosting algorithms in the literature are covered by Definition 16, such as the original boosting algorithm of Schapire (1990), Boost-by-Majority (Freund, 1995), MadaBoost (Domingo and Watanabe, 2000), BrownBoost (Freund, 2001), SmoothBoost (Servedio, 2003), FilterBoost (Bradley and Schapire, 2007) and others. All these boosters use $\Omega(\log(1/\varepsilon)/\gamma^2)$ stages of boosting to achieve $1-\varepsilon$ accuracy, and indeed Freund (1995) has shown that any sequential booster must run for $\Omega(\log(1/\varepsilon)/\gamma^2)$ stages. More precisely, Freund (1995) modeled the phenomenon of boosting using the majority function to combine weak hypotheses as an interactive game between a "weightor" and a "chooser" (see Freund, 1995, Section 2). He gave a strategy for the weightor, which corresponds to a boosting algorithm, and showed that after T stages of boosting this boosting algorithm generates a final hypothesis that is guaranteed to have error at most $\mathrm{vote}(\gamma,T) \stackrel{\text{def}}{=} \sum_{j=0}^{\lfloor T/2\rfloor}\binom{T}{j}\left(\frac{1}{2}+\gamma\right)^j(1/2-\gamma)^{T-j}$ (see Freund, 1995, Theorem 2.1). Freund also gives a matching lower bound by showing (see his Theorem 2.4) that any T-stage sequential booster must have error at least as large as $\mathrm{vote}(\gamma,T)$, and so consequently any sequential booster that generates a $(1-\varepsilon)$-accurate final hypothesis must run for $\Omega(\log(1/\varepsilon)/\gamma^2)$ stages. Our Theorem 18 below extends this lower bound to parallel boosters.

3.1.2 PARALLEL BOOSTING

Parallel boosting is a natural generalization of sequential boosting. In stage t of a parallel booster the boosting algorithm may simultaneously run the weak learner many times in parallel using different probability distributions. The distributions that are used in stage t may depend on any of the weak hypotheses from earlier stages, but may not depend on any of the weak hypotheses generated by any of the calls to the weak learner in stage t.

Definition 17 (Parallel booster) *A T-stage parallel boosting algorithm with N-fold parallelism is defined by TN functions $\{\alpha_{t,k}\}_{t\in[T],k\in[N]}$ and a (randomized) Boolean function h, where $\alpha_{t,k}$: $\{-1,1\}^{(t-1)N+1} \to [0,1]$ and $h : \{-1,1\}^{TN} \to \{-1,1\}$. In the t-th stage of boosting the weak learner is run N times in parallel. For each $k \in [N]$, the distribution $\mathcal{P}_{t,k}$ over labeled examples that is given to the k-th run of the weak learner is as follows: a draw from $\mathcal{P}_{t,k}$ is made by drawing a labeled example $(x, f(x))$ from $\mathcal{P}$, computing the value $p_x := \alpha_{t,k}(h_{1,1}(x), \ldots, h_{t-1,N}(x), f(x))$, and accepting $(x, f(x))$ as the output of the draw from $\mathcal{P}_{t,k}$ with probability p_x (and rejecting it and trying again otherwise). In stage t, for each $k \in [N]$ the booster gives the weak learner access to $\mathcal{P}_{t,k}$ as defined above and the weak learner generates a hypothesis $h_{t,k}$ that has advantage at least γ w.r.t. $\mathcal{P}_{t,k}$. Together with the weak hypotheses $\{h_{s,j}\}_{s\in[t-1],j\in[N]}$ obtained in earlier stages, these $h_{t,k}$'s enable the booster to give the weak learner access to each $\mathcal{P}_{t+1,k}$ in the next stage.*

After T stages, TN weak hypotheses $\{h_{t,k}\}_{t\in[T],k\in[N]}$ have been obtained from the weak learner. The final hypothesis *of the booster is $H(x) := h(h_{1,1}(x), \ldots, h_{T,N}(x))$, and its* accuracy *is*

$$\min_{h_{t,k}} \Pr_{(x,f(x))\leftarrow\mathcal{P}}[H(x) = f(x)],$$

where the min is taken over all sequences of TN weak hypotheses subject to the condition that each $h_{t,k}$ has advantage at least γ w.r.t. $\mathcal{P}_{t,k}$.

The parameter N above corresponds to the number of processors that the parallel booster is using. Parallel boosting algorithms that call the weak learner different numbers of times at different stages fit into our definition simply by taking N to be the max number of parallel calls made at any stage. Several parallel boosting algorithms have been given in the literature; in particular, all boosters that construct branching program or decision tree hypotheses are of this type. The number of stages of these boosting algorithms corresponds to the depth of the branching program or decision tree that is constructed, and the number of nodes at each depth corresponds to the parallelism parameter. Branching program boosters (Mansour and McAllester, 2002; Kalai and Servedio, 2005; Long and Servedio, 2005, 2008) all make $\mathrm{poly}(1/\gamma)$ many calls to the weak learner within each stage and all require $\Omega(\log(1/\varepsilon)/\gamma^2)$ stages, while the earlier decision tree booster (Kearns and Mansour, 1996) requires $\Omega(\log(1/\varepsilon)/\gamma^2)$ stages but makes $2^{\Omega(\log(1/\varepsilon)/\gamma^2)}$ parallel calls to the weak learner in some stages. Our results in the next subsection will imply that *any* parallel booster must run for $\Omega(\log(1/\varepsilon)/\gamma^2)$ stages no matter how many parallel calls to the weak learner are made in each stage.

3.2 The Lower Bound and Its Proof

Our lower bound theorem for parallel boosting is the following:

Theorem 18 *Let B be any T-stage parallel boosting algorithm with N-fold parallelism. Then for any $0 < \gamma < 1/2$, when B is used to boost a γ-advantage weak learner the resulting final hypothesis may have error as large as* $\mathrm{vote}(\gamma, T)$ *(see the discussion after Definition 17).*

We emphasize that Theorem 18 holds for any γ and any N that may depend on γ in an arbitrary way.

The theorem is proved as follows: fix any $0 < \gamma < 1/2$ and fix B to be any T-stage parallel boosting algorithm. We will exhibit a target function f and a distribution $\mathcal{P}$ over $\{(x, f(x))_{x\in X}$, and

describe a strategy that a weak learner W can use to generate weak hypotheses $h_{t,k}$ that all have advantage at least γ with respect to the distributions $\mathcal{P}_{t,k}$. We show that with this weak learner W, the resulting final hypothesis H that B outputs will have accuracy at most $1 - \text{vote}(\gamma, T)$.

We begin by describing the desired f and $\mathcal{P}$, both of which are fairly simple. The domain X of f is $X = Z \times \Omega$, where Z denotes the set $\{-1,1\}$ and Ω denotes the set of all infinite sequences $\omega = (\omega_1, \omega_2, \dots)$ where each ω_i belongs to $\{-1,1\}$. The target function f is simply $f(z,\omega) = z$; that is, f always simply outputs the first coordinate of its input vector. The distribution $\mathcal{P} = (\mathcal{P}^X, \mathcal{P}^Y)$ over labeled examples $\{(x, f(x))\}_{x \in X}$ is defined as follows.[3] A draw from $\mathcal{P}$ is obtained by drawing $x = (z,\omega)$ from $\mathcal{P}^X$ and returning $(x, f(x))$. A draw of $x = (z,\omega)$ from $\mathcal{P}^X$ is obtained by first choosing a uniform random value in $\{-1,1\}$ for z, and then choosing $\omega_i \in \{-1,1\}$ to equal z with probability $1/2 + \gamma$ independently for each i. Note that under $\mathcal{P}$, given the label $z = f(x)$ of a labeled example $(x, f(x))$, each coordinate ω_i of x is correct in predicting the value of $f(x,z)$ with probability $1/2 + \gamma$ independently of all other ω_j's.

We next describe a way that a weak learner W can generate a γ-advantage weak hypothesis each time it is invoked by B. Fix any $t \in [T]$ and any $k \in [N]$, and recall that $\mathcal{P}_{t,k}$ is the distribution over labeled examples that is used for the k-th call to the weak learner in stage t. When W is invoked with $\mathcal{P}_{t,k}$ it replies as follows (recall that for $x \in X$ we have $x = (z,\omega)$ as described above):

(i) If $\Pr_{(x,f(x)) \leftarrow \mathcal{P}_{t,k}}[\omega_t = f(x)] \geq 1/2 + \gamma$ then the weak hypothesis $h_{t,k}(x)$ is the function "ω_t," the $(t+1)$-st coordinate of x. Otherwise,

(ii) the weak hypothesis $h_{t,k}(x)$ is "z," the first coordinate of x. (Note that since $f(x) = z$ for all x, this weak hypothesis has zero error.)

It is clear that each weak hypothesis $h_{t,k}$ generated as described above indeed has advantage at least γ w.r.t. $\mathcal{P}_{t,k}$, so the above is a legitimate strategy for W. It is also clear that if the weak learner ever uses option (ii) above at some invocation (t,k) then B may output a zero-error final hypothesis simply by taking $H = h_{t,k} = f(x)$. On the other hand, the following crucial lemma shows that if the weak learner never uses option (ii) for any (t,k) then the accuracy of B is upper bounded by $\text{vote}(\gamma, T)$:

Lemma 19 *If W never uses option (ii) then* $\Pr_{(x,f(x)) \leftarrow \mathcal{P}}[H(x) \neq f(x)] \geq \text{vote}(\gamma, T)$.

Proof If the weak learner never uses option (ii) then H depends only on variables

$$\omega_1, \dots, \omega_T$$

and hence is a (randomized) Boolean function over these variables. Recall that for $(x = (z,\omega), f(x) = z)$ drawn from $\mathcal{P}$, each coordinate

$$\omega_1, \dots, \omega_T$$

independently equals z with probability $1/2 + \gamma$. Hence the optimal (randomized) Boolean function H over inputs $\omega_1, \dots, \omega_T$ that maximizes the accuracy $\Pr_{(x,f(x)) \leftarrow \mathcal{P}}[H(x) = f(x)]$ is the (deterministic) function $H(x) = \mathsf{Maj}(\omega_1, \dots, \omega_T)$ that outputs the majority vote of its input bits. (This can be

3. Note that $\mathcal{P}^X$ and $\mathcal{P}^Y$ are not independent; indeed, in a draw $(x, y = f(x))$ from $(\mathcal{P}^X, \mathcal{P}^Y)$ the outcome of x completely determines y.

easily verified using Bayes' rule in the usual "Naive Bayes" calculation.) The error rate of this H is precisely the probability that at most $\lfloor T/2 \rfloor$ "heads" are obtained in T independent $(1/2+\gamma)$-biased coin tosses, which equals $\text{vote}(\gamma, T)$. ■

Thus to prove Theorem 18 it suffices to prove the following lemma, which we prove by induction on t:

Lemma 20 *W never uses option (ii) (that is,* $\Pr_{(x,f(x))\leftarrow\mathcal{P}_{t,k}}[\omega_t = f(x)] \geq 1/2+\gamma$ *always).*

Proof *Base case ($t = 1$).* For any $k \in [N]$, since $t = 1$ there are no weak hypotheses from previous stages, so the value of the rejection sampling parameter p_x is determined by the bit $f(x) = z$ (see Definition 17). Hence $\mathcal{P}_{1,k}$ is a convex combination of two distributions which we call $\mathcal{D}_1$ and $\mathcal{D}_{-1}$. For $b \in \{-1,1\}$, a draw of $(x = (z,\omega); f(x) = z)$ from $\mathcal{D}_b$ is obtained by setting $z = b$ and independently setting each coordinate ω_i equal to z with probability $1/2+\gamma$. Thus in the convex combination $\mathcal{P}_{1,k}$ of $\mathcal{D}_1$ and $\mathcal{D}_{-1}$, we also have that ω_1 equals z (that is, $f(x)$) with probability $1/2+\gamma$. So the base case is done.

Inductive step ($t > 1$). Thanks to the conditional independence of different coordinates ω_i given the value of z in a draw from $\mathcal{P}$, the proof is quite similar to the base case.

Fix any $k \in [N]$. The inductive hypothesis and the weak learner's strategy together imply that for each labeled example $(x = (z,\omega), f(x) = z)$, since $h_{s,\ell}(x) = \omega_s$ for $s < t$, the rejection sampling parameter $p_x = \alpha_{t,k}(h_{1,1}(x), \ldots, h_{t-1,N}(x), f(x))$ is determined by $\omega_1, \ldots, \omega_{t-1}$ and z and does not depend on $\omega_t, \omega_{t+1}, \ldots$. Consequently the distribution $\mathcal{P}_{t,k}$ over labeled examples is some convex combination of 2^t distributions which we denote $\mathcal{D}_{\overline{b}}$, where $\overline{b}$ ranges over $\{-1,1\}^t$ corresponding to conditioning on all possible values for $\omega_1, \ldots, \omega_{t-1}, z$. For each $\overline{b} = (b_1, \ldots, b_t) \in \{-1,1\}^t$, a draw of $(x = (z,\omega); f(x) = z)$ from $\mathcal{D}_{\overline{b}}$ is obtained by setting $z = b_t$, setting $(\omega_1, \ldots, \omega_{t-1}) = (b_1, \ldots, b_{t-1})$, and independently setting each other coordinate ω_j $(j \geq t)$ equal to z with probability $1/2+\gamma$. In particular, because ω_t is conditionally independent of $\omega_1, \ldots, \omega_{t-1}$ given z, $\Pr(\omega_t = z | \omega_1 = b_1, \ldots, \omega_{t-1} = b_{t-1}) = \Pr(\omega_t = z) = 1/2+\gamma$. Thus in the convex combination $\mathcal{P}_{t,k}$ of the different $\mathcal{D}_{\overline{b}}$'s, we also have that ω_t equals z (that is, $f(x)$) with probability $1/2+\gamma$. This concludes the proof of the lemma and the proof of Theorem 18. ■

4. Conclusion

There are many natural directions for future work on understanding the parallel complexity of learning large-margin halfspaces. One natural goal, of course, is to give an algorithm that provides an affirmative answer to the main question. But it is not clear to us that such an algorithm must actually exist, and so another intriguing direction is to prove negative results giving evidence that parallel learning of large-margin halfspaces is computationally hard.

As one example of a possible negative result, perhaps it is the case that (assuming $P \neq NC$) there is no $\text{poly}(n)$-processor, $\text{polylog}(n)$-time algorithm with the following performance guarantee: given a sample of $\text{poly}(n)$ many n-dimensional labeled examples that are consistent with some $1/\text{poly}(n)$-margin halfspace, the algorithm outputs a consistent halfspace hypothesis. A stronger result would be that no such algorithm can even output a halfspace hypothesis which is consistent

with 99% (or 51%) of the labeled examples. Because of the requirement of a halfspace representation for the hypothesis such results would not directly contradict the main question, but they are contrary to it in spirit. We view the possibility of establishing such negative results as an interesting direction worthy of future study.

Acknowledgments

We thank Sasha Rakhlin for telling us about the paper of Soheili and Peña (2012), and anonymous reviewers for helpful comments.

Appendix A. Proof of Lemma 2

First, let us establish that we can "boost the confidence" efficiently. Suppose we have an algorithm that achieves accuracy $1-\varepsilon$ in parallel time $\mathcal{T}''$ with probability c_δ. Then we can run $O(\log(1/\delta))$ copies of this algorithm in parallel, then test each of their hypotheses in parallel using $O(\log(1/\delta)/\varepsilon)$ examples. The tests of individual examples can be done in parallel, and we can compute each empirical error rate in $O(\log(1/\varepsilon)+\log\log(1/\delta))$ time. Then we can output the hypothesis with the best accuracy on this additional test data. Finding the best hypothesis takes at most $O(\log\log(1/\delta))$ parallel time (with polynomially many processors). The total parallel time taken is then $O(\mathcal{T}''+\log(1/\varepsilon)+\log\log(1/\delta))$.

So now, we have as a subproblem the problem of achieving accuracy $1-\varepsilon$ with constant probability, say $1/2$.

The theorem statement assumes that we have as a subroutine an algorithm A that achieves constant accuracy with constant probability in time $\mathcal{T}$. Using the above reduction, we can use A to get an algorithm A' that achieves constant accuracy with probability $1-c/\log(1/\varepsilon)$ (for a constant c) in $\mathcal{T}'=O(\mathcal{T}+\log\log\log(1/\varepsilon))$ time. We will use such an algorithm A'. (Note that the time taken by A' is an upper bound on the number of examples needed by A'.)

Algorithm B runs a parallel version of a slight variant of the "boosting-by-filtering" algorithm due to Freund (1995), using A' as a weak learner. Algorithm B uses parameters α and T:

- For rounds $t=0,\ldots,T-1$
  - draw $m=\frac{2T\alpha}{\varepsilon}\max\{\mathcal{T}',4\ln\frac{32T^2\alpha}{\varepsilon}\}$ examples, call them
    $$S_t=\{(x_{t,1},y_{t,1}),\ldots,(x_{t,m},y_{t,m})\}.$$
  - for each $i=1,\ldots,m$,
    - let $r_{t,i}$ be the the number of previous base classifiers h_0,...,h_{t-1} that are correct on $(x_{t,i},y_{t,i})$, and
    - $w_{t,i}=\binom{T-t-1}{\lfloor\frac{T}{2}\rfloor-r_{t,i}}(\frac{1}{2}+\alpha)^{\lfloor\frac{T}{2}\rfloor-r_{t,i}}(\frac{1}{2}-\alpha)^{\lceil\frac{T}{2}\rceil-t-1+r_{t,i}}$,
  - let $w_{t,\max}=\max_r\binom{T-t-1}{\lfloor\frac{T}{2}\rfloor-r}(\frac{1}{2}+\alpha)^{\lfloor\frac{T}{2}\rfloor-r}(\frac{1}{2}-\alpha)^{\lceil\frac{T}{2}\rceil-t-1+r}$ be the largest possible value that any $w_{t,i}$ could take,
  - apply the rejection method as follows: for each $i\in S_t$,
    - choose $u_{t,i}$ uniformly from $[0,1]$,

  - ∗ if $u_{t,i} \leq \frac{w_{t,i}}{w_{t,\max}}$, set $a_{t,i} = 1$
- if there is a j such that $j > \frac{T\alpha w_{t,\max}}{\varepsilon} \max\left\{\sum_{i=1}^{j} a_{t,i}, 4\ln\frac{16T^2\alpha w_{t,\max}}{\varepsilon(1-\varepsilon)}\right\}$
  - ∗ output a hypothesis h_t that predicts randomly,
  - ∗ otherwise, pass the examples in S_t to Algorithm A', which returns h_t.

- Output the classifier obtained by taking a majority vote over $h_0, ..., h_{T-1}$.

The only difference between algorithm B, as described above, and the way the algorithm is described by Freund (1995) is that, in the above description, a batch of examples is chosen at the beginning of the round. The number of examples is set using Freund's upper bound on the number of examples that can be chosen in a given round (see the displayed equation of the boost-by-majority paper (Freund, 1995) immediately before (18)). In Freund's description of this algorithm, once the condition which causes the algorithm to output a random hypothesis is reached, the algorithm stops sampling, but, for a parallel version, it is convenient to sample all of the examples for a round in parallel.

Freund (1995) proves that, if α is a constant depending only on the accuracy of the hypotheses output by A', then $T = O(\log(1/\varepsilon))$ suffices for algorithm B to output a hypothesis with accuracy $1-\varepsilon$ with probability $1/2$. So the parallel time taken is $O(\log(1/\varepsilon))$ times the time taken in each iteration.

Let us now consider the time taken in each iteration. The weights for the various examples can be computed in parallel. The value of $w_{t,i}$ is a product of $O(T)$ quantities, each of which can be expressed using T bits, and can therefore be computed in $O(\text{poly}(\log T)) = O(\text{poly}(\log\log(1/\varepsilon)))$ parallel time, as can $w_{t,\max}$. The rejection step also may be done in $O(\text{poly}(\log\log(1/\varepsilon)))$ time in parallel for each example. To check whether there is a j such that

$$j > \frac{T\alpha w_{t,\max}}{\varepsilon} \max\left\{\sum_{i=1}^{j} a_{t,i}, 4\ln\frac{16T^2\alpha w_{t,\max}}{\varepsilon(1-\varepsilon)}\right\},$$

Algorithm B can compute the prefix sums $\sum_{i=1}^{j} a_{t,i}$, and then test them in parallel. The prefix sums can be computed in $\log(T)$ parallel rounds (each on $\log(T)$-bit numbers), using the standard technique of placing the values of $a_{t,i}$ on the leaves of a binary tree, and working up from the leaves to the root, computing the sums of subtrees, then making a pass down the tree, passing each node's sum to its right child, and using these to compute prefix sums in the obvious way.

Appendix B. Proof of Lemma 5

Algorithm A_{Nes} is a special case of the algorithm of (2.2.11) on page 81 of the book by Nesterov (2004), obtained by setting $y_0 \leftarrow \mathbf{0}$ and $x_0 \leftarrow \mathbf{0}$. The bound of Lemma 5 is a consequence of Theorem 2.2.3 on page 80 of Nesterov's book. This Theorem applies to all functions f that are μ-strongly convex, and continuously differentiable with a gradient that is L-Lipschitz (see pages 71, 63 and 20). Lemmas 3 and 4 of this paper imply that Theorem 2.2.3 of Nesterov's book applies to Ψ.

Plugging directly into Theorem 2.2.3 (in the special case of (2.2.11))

$$\Psi(\mathbf{v}_k) - \Psi(\mathbf{w}) \leq \frac{4L}{(2\sqrt{L} + k\sqrt{\mu})^2}(\Psi(\mathbf{0}) - \Psi(\mathbf{w}) + \mu||\mathbf{w}||^2)$$

which implies the Lemma 5, since $\Psi(\mathbf{0}) \leq 1$ and $\Psi(\mathbf{w}) \geq 0$.

Appendix C. Proof of Lemma 7

First, we prove

$$\Pr_A[\Pr_{\mathbf{x}' \sim \mathcal{D}'}\left[||\mathbf{x}'|| > 2\right] > \gamma^4/2] \; < \; 1/200. \tag{2}$$

Recall that we sample $\mathbf{x}'$ from D' by first sampling $\mathbf{x}$ from a distribution D over B_n (so that $||\mathbf{x}|| = 1$), and then setting $\mathbf{x}' = (1/\sqrt{d})\mathbf{x}A$, so that (2) is equivalent to

$$\Pr_A[\Pr_{\mathbf{x} \sim \mathcal{D}}\left[||\mathbf{x}A|| > 2\sqrt{d}\right] > \gamma^4/2] \; < \; 1/200.$$

Corollary 1 of the paper of Arriaga and Vempala (2006) directly implies that, for any $\mathbf{x}$ in $\mathbf{B}_n$, we have

$$\Pr_A[||\mathbf{x}A|| \geq 2\sqrt{d}] \leq 2e^{-\frac{d}{32}},$$

so

$$\mathbf{E}_{\mathbf{x} \in D}[\Pr_A[||\mathbf{x}A|| \geq 2\sqrt{d}]] \leq 2e^{-\frac{d}{32}},$$

which implies

$$\mathbf{E}_A[\Pr_{\mathbf{x} \in D}[||\mathbf{x}A|| \geq 2\sqrt{d}]] \leq 2e^{-\frac{d}{32}}.$$

Applying Markov's inequality,

$$\Pr_A[\Pr_{\mathbf{x} \in D}[||\mathbf{x}A|| \geq 2\sqrt{d}] > 400e^{-\frac{d}{32}}] \leq 1/200.$$

Setting $d = O(\log(1/\gamma))$ then suffices to establish (2).

Now, we want to show that $d = O((1/\gamma^2)\log(1/\gamma))$ suffices to ensure that

$$\Pr_A\left[\Pr_{\mathbf{x}' \sim \mathcal{D}'}\left[\left|\frac{\mathbf{w}'}{||\mathbf{w}'||} \cdot \mathbf{x}'\right| < \gamma/2\right] > \gamma^4/2\right] \leq 1/200.$$

As above, Corollary 1 of the paper by Arriaga and Vempala (2006) directly implies that there is an absolute constant $c_1 > 0$ such that

$$\Pr_A[||\mathbf{w}'|| = ||(1/\sqrt{d})\mathbf{w}A|| > 3/2] \leq 2e^{-c_1 d}.$$

Furthermore, for any $\mathbf{x} \in \mathbf{B}_n$, Corollary 2 of the paper by Arriaga and Vempala (2006) directly implies that there is an absolute constant $c_2 > 0$ such that

$$\Pr_A[\mathbf{w}' \cdot \mathbf{x}' \leq 3\gamma/4] \leq 4e^{-c_2\gamma^2 d}.$$

Thus,

$$\Pr_A\left[\frac{\mathbf{w}'}{||\mathbf{w}'||} \cdot \mathbf{x}' \leq \gamma/2\right] \leq 2e^{-c_1 d} + 4e^{-c_2\gamma^2 d}.$$

Arguing as above, we have

$$\mathbf{E}_{\mathbf{x} \in D}\left[\Pr_A\left[\frac{\mathbf{w}'}{||\mathbf{w}'||} \cdot \mathbf{x}' \leq \gamma/2\right]\right] \leq 2e^{-c_1 d} + 4e^{-c_2\gamma^2 d},$$

$$\mathbf{E}_A\left[\Pr_{\mathbf{x} \in D}\left[\frac{\mathbf{w}'}{||\mathbf{w}'||} \cdot \mathbf{x}' \leq \gamma/2\right]\right] \leq 2e^{-c_1 d} + 4e^{-c_2\gamma^2 d},$$

$$\Pr_A\left[\Pr_{\mathbf{x} \in D}\left[\frac{\mathbf{w}'}{||\mathbf{w}'||} \cdot \mathbf{x}' \leq \gamma/2\right] > 200(2e^{-c_1 d} + 4e^{-c_2\gamma^2 d})\right] \leq 1/200,$$

from which $d = O((1/\gamma^2)\log(1/\gamma))$ suffices to get

$$\Pr_A\left[\Pr_{\mathbf{x}\in D}\left[\frac{\mathbf{w}'}{||\mathbf{w}'||}\cdot\mathbf{x}' \leq \gamma/2\right] > \gamma^A/2\right] \leq 1/200,$$

completing the proof.

Appendix D. Proof of Lemma 13

First, A_r finds a rough guess u_1 such that

$$\sqrt{z}/2 \leq u_1 \leq \sqrt{z}. \tag{3}$$

This can be done by checking in parallel, for each of $\theta \in \{1/2^L, 1/2^{L-1}, ..., 1/2, 1, 2, ..., 2^L\}$, whether $\sqrt{z} \geq \theta$, and outputting the largest such θ. This first step takes $O(\log L)$ time using $O(L)$ processors. Then, using u_1 as the initial solution, A_r runs Newton's method to find a root of the function f defined by $f(u) = u^2 - z$, repeatedly

$$u_{k+1} = \frac{1}{2}\left(u_k + \frac{z}{u_k}\right). \tag{4}$$

As we will see below, this is done for $k = 1, \ldots, O(\log L + \log\log(1/\beta))$. Using the fact that the initial value u_1 is an L-bit rational number, a straightforward analysis using (4) shows that for all $k \leq O(\log L + \log\log(1/\beta))$ the number u_k is a rational number with $\text{poly}(L, \log(1/\beta))$ bits (if b_k is the number of bits required to represent u_k, then $b_{k+1} \leq 2b_k + O(L)$). Standard results on the parallel complexity of integer multiplication thus imply that for $k \leq O(\log L + \log\log(1/\beta))$ the exact value of u_k can be computed in the parallel time and processor bounds claimed by the Lemma. To prove the Lemma, then, it suffices to show that taking $k = O(\log L + \log\log(1/\beta))$ gives the desired accuracy; we do this next.

The Newton iterates defined by (4) satisfy

$$\frac{u_{k+1} - \sqrt{z}}{u_{k+1} + \sqrt{z}} = \left(\frac{u_k - \sqrt{z}}{u_k + \sqrt{z}}\right)^2$$

(see Weisstein, 2011), which, using induction, gives

$$\frac{u_{k+1} - \sqrt{z}}{u_{k+1} + \sqrt{z}} = \left(\frac{u_1 - \sqrt{z}}{u_1 + \sqrt{z}}\right)^{2^k}.$$

Solving for u_{k+1} yields

$$u_{k+1} = \sqrt{z}\left(\frac{1 + \left(\frac{u_1-\sqrt{z}}{u_1+\sqrt{z}}\right)^{2^k}}{1 - \left(\frac{u_1-\sqrt{z}}{u_1+\sqrt{z}}\right)^{2^k}}\right) = \sqrt{z}\left(1 + \frac{2\left(\frac{u_1-\sqrt{z}}{u_1+\sqrt{z}}\right)^{2^k}}{1 - \left(\frac{u_1-\sqrt{z}}{u_1+\sqrt{z}}\right)^{2^k}}\right).$$

Thus,

$$u_{k+1} - \sqrt{z} = \frac{2\sqrt{z}\left(\frac{u_1-\sqrt{z}}{u_1+\sqrt{z}}\right)^{2^k}}{1 - \left(\frac{u_1-\sqrt{z}}{u_1+\sqrt{z}}\right)^{2^k}}$$

and, therefore, to get $|u_{k+1} - \sqrt{z}| \leq \beta$, we only need

$$\left(\frac{u_1 - \sqrt{z}}{u_1 + \sqrt{z}}\right)^{2^k} \leq \min\left\{\frac{\beta}{4\sqrt{z}}, 1/2\right\}.$$

Applying (3),

$$(1/4)^{2^k} \leq \min\left\{\frac{\beta}{4\sqrt{z}}, 1/2\right\}$$

also suffices, and, solving for k, this means that

$$O(\log\log z + \log\log(1/\beta)) = O(\log L + \log\log(1/\beta))$$

iterations are enough. ■

References

N. Alon and N. Megiddo. Parallel linear programming in fixed dimension almost surely in constant time. *J. ACM*, 41(2):422–434, 1994.

R. I. Arriaga and Santosh Vempala. An algorithmic theory of learning: Robust concepts and random projection. *Machine Learning*, 63(2):161–182, 2006.

P. Beame, S.A. Cook, and H.J. Hoover. Log depth circuits for division and related problems. *SIAM J. on Computing*, 15(4):994–1003, 1986.

H. Block. The Perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34:123–135, 1962.

A. Blum. Random Projection, Margins, Kernels, and Feature-Selection. In *Subspace, Latent Structure and Feature Selection*, pages 52–68, 2006.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

J. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *Proc. 28th ICML*, pages 321–328, 2011.

J. K. Bradley and R. E. Schapire. Filterboost: Regression and classification on large datasets. In *Proc. 21st NIPS*, 2007.

N. Bshouty, S. Goldman, and H.D. Mathias. Noise-tolerant parallel learning of geometric concepts. *Inf. and Comput.*, 147(1):89–110, 1998. ISSN 0890-5401. doi: DOI: 10.1006/inco.1998.2737.

M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. *Machine Learning*, 48(1-3):253–285, 2002.

A. d'Aspremont. Smooth optimization with approximate gradient. *SIAM Journal on Optimization*, 19 (3): 1171–1183, 2008.

O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction. In *Proc. 28th ICML*, pages 713–720, 2011.

C. Domingo and O. Watanabe. MadaBoost: A modified version of AdaBoost. In *Proc. 13th COLT*, pages 180–189, 2000.

Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121 (2): 256–285, 1995.

Y. Freund. An adaptive version of the boost-by-majority algorithm. *Machine Learning*, 43(3):293–318, 2001.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, New York, 1995.

A. Kalai and R. Servedio. Boosting in the presence of noise. *Journal of Computer & System Sciences*, 71(3):266–290, 2005.

N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinat.*, 4:373–395, 1984.

M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proc. 28th STOC*, pages 459–468, 1996.

M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, 1994.

N. Littlestone. From online to batch learning. In *Proc. 2nd COLT*, pages 269–284, 1989.

P. Long and R. Servedio. Martingale boosting. In *Proc. 18th COLT*, pages 79–94, 2005.

P. Long and R. Servedio. Adaptive martingale boosting. In *Proc. 22nd NIPS*, pages 977–984, 2008.

P. Long and R. Servedio. Algorithms and hardness results for parallel large margin learning. In *Proc. 25th NIPS*, 2011.

Y. Mansour and D. McAllester. Boosting using branching programs. *Journal of Computer & System Sciences*, 64(1):103–112, 2002.

Y. Nesterov. *Introductory lectures on Convex Optimization*. Kluwer, 2004.

Y. Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM J. Optimization*, 16(1):235–249, 2005.

A. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on Mathematical Theory of Automata*, volume XII, pages 615–622, 1962.

F. Rosenblatt. The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.

R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

R. Servedio. Smooth boosting and learning with malicious noise. *JMLR*, 4:633–648, 2003.

S. Shalev-Shwartz and Y. Singer. On the equivalence of weak learnability and linear separability: New relaxations and efficient boosting algorithms. *Machine Learning*, 80(2):141–163, 2010.

N. Soheili and J. Peña. A smooth perceptron algorithm. *SIAM J. Optimization*, 22(2):728–737, 2012.

L. Valiant. A theory of the learnable. *Communications of the ACM*, 27 (11): 1134–1142, 1984.

V. N. Vapnik and A. Y. Chervonenkis. *Theory of Pattern Recognition*. Nauka, 1974. In Russian.

J. S. Vitter and J. Lin. Learning in parallel. *Inf. Comput.*, 96(2):179–202, 1992.

E. W. Weisstein. Newton's iteration, 2011. http://mathworld.wolfram.com/NewtonsIteration.html.

DIMACS 2011 Workshop. Parallelism: A 2020 Vision. 2011.

NIPS 2009 Workshop. Large-Scale Machine Learning: Parallelism and Massive Datasets. 2009.