On the Learnability of Fully-connected Neural Networks

Yuchen Zhang
Stanford University
zhangyuc@stanford.edu

Jason D. Lee
University of Southern California
jasonlee@marshall.usc.edu

Martin J. Wainwright and Michael I. Jordan
University of California, Berkeley
wainwrig@berkeley.edu
jordan@berkeley.edu

Abstract

Despite the empirical success of deep neural networks, there is limited theoretical understanding of the learnability of these models with respect to polynomial-time algorithms. In this paper, we characterize the learnability of fully-connected neural networks via both positive and negative results. We focus on ℓ_1-regularized networks, where the ℓ_1-norm of the incoming weights of every neuron is assumed to be bounded by a constant $B > 0$. Our first result shows that such networks are properly learnable in $\text{poly}(n, d, \exp(1/\epsilon^2))$ time, where n and d are the sample size and the input dimension, and $\epsilon > 0$ is the gap to optimality. The bound is achieved by repeatedly sampling over a low-dimensional manifold so as to ensure approximate optimality, but avoids the $\exp(d)$ cost of exhaustively searching over the parameter space. We also establish a hardness result showing that the exponential dependence on $1/\epsilon$ is unavoidable unless $\mathbf{RP} = \mathbf{NP}$. Our second result shows that the exponential dependence on $1/\epsilon$ can be avoided by exploiting the underlying structure of the data distribution. In particular, if the positive and negative examples can be separated with margin $\gamma > 0$ by an unknown neural network, then the network can be learned in $\text{poly}(n, d, 1/\epsilon)$ time. The bound is achieved by an ensemble method which uses the first algorithm as a weak learner. We further show that the separability assumption can be weakened to tolerate noisy labels. Finally, we show that the exponential dependence on $1/\gamma$ is unimprovable under a certain cryptographic assumption.

Proceedings of the 20^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54.

1 Introduction

Deep neural networks have been successful applied to various problems in machine learning, including image classification [12], speech recognition [8], natural language processing [2] and reinforcement learning [19] problems. Despite this empirical success, the theoretical understanding of learning neural networks remains relatively limited. It is known that training a two-layer neural network on the worst-case data distribution is NP-hard [5]. Real data, however, is rarely generated from the worst-case distribution. It is thus natural to wonder whether there are conditions under which an accurate neural network can be learned in polynomial time.

This paper provides some theoretical analysis of the learnability of neural networks. We focus on the problem of binary classification, and study fully-connected neural networks with a constant number m of layers, and such that the ℓ_1-norm of the incoming weights of every neuron is bounded by a constant $B > 0$. This ℓ_1-regularization scheme has been studied by many authors [see, e.g., 3, 11, 4, 21]. Under the same setting, Zhang et al. [28] proposed a kernel-based method for improperly learning a classifier that is competitive against the best possible neural network. Our goal, in contrast, is to explicitly learn the neural network and its parameters, in the *proper learning* regime.

The main challenge of learning neural networks comes from the nonconvexity of the loss function. An exhaustive search over the parameter space can be conducted to obtain a global optimum, but its time complexity will be exponential in the number of parameters. Existing learnability results either assume a constant network scale [18], or assume that every hidden node connects to a constant number of input coordinates [16]. To the best of our knowledge, no agnostic learning algorithm has been shown to learn fully-connected neural networks with time complexity polynomial in the number of network parameters.

Our first result is to exhibit an algorithm whose running time is polynomial in the number of parameters to achieve a constant optimality gap. Specifically, it is guaranteed to achieve an empirical loss that is at most $\epsilon > 0$ greater than that of the best neural network with time complex-

ity poly$(n, d, C_{m,B,1/\epsilon})$. Here the integers n and d are the sample size and the input dimension, and the constant $C_{m,B,1/\epsilon}$ only depends on the triplet $(m, B, 1/\epsilon)$, with this dependence possibly being exponential. Thus, for a constant optimality gap $\epsilon > 0$, number of layers m and ℓ_1-bound B, the method runs in polynomial time in the pair (n, d). We refer to this method as the *agnostic learning algorithm*, since it makes no assumption on the data distribution. It is remarkably simple, using only multiple rounds of random sampling followed by optimization rounds. The insight is that although the network contains $\Omega(d)$ parameters, the empirical loss can be approximately minimized by only considering parameters lying on a k-dimensional manifold. Thus it suffices to optimize the loss over the manifold. The dimension k is independent of scale of the network, only relying on the target optimality gap ϵ.

Due to the exponential dependence on $1/\epsilon$, this first algorithm is too expensive to achieve a diminishing excess risk for large datasets. In our next result, we show how this exponential dependence can be removed by exploiting the underlying structure of the data distribution. In particular, by assuming that the positive and the negative examples of the dataset are separable by some unknown neural network with a constant margin $\gamma > 0$, we propose an algorithm that learns the network in polynomial time, and correctly classifies all training points with margin $\Omega(\gamma)$. As a consequence, it achieves a generalization error bounded by ϵ with sample complexity $n = \text{poly}(d, 1/\epsilon)$ and time complexity poly$(n, d, 1/\epsilon)$. Both complexities have a polynomial dependence on $1/\epsilon$. We name it the *BoostNet algorithm* because it uses the AdaBoost approach [7] to construct a m-layer neural network, by incrementally ensembling shallower $(m-1)$-layer neural networks. Each shallow network is trained by the agnostic learning algorithm presented earlier, focusing on instances that are not correctly addressed by existing shallow networks. Although each shallow network only guarantees a constant optimality gap, such constant gaps can be boosted to a diminishing error rate via a suitable ensembling procedure. We note that for real-world data, the labels are often noisy so that the separability assumption is unlikely to hold hold. A more realistic assumption would be that the underlying "true labels" are separable, but the observed labels are corrupted by random noise and are not separable. For such noisy data, we prove that the same poly$(n, d, 1/\epsilon)$-time learnability can be achieved by a variant of the BoostNet algorithm.

To provide some historical context, in earlier work, a number of practioners [13, 24] have reported good empirical results using a neural network as a weak learner for AdaBoost. However, to date, there has been a limited theoretical understanding of this approach. The fundamental issue is that AdaBoost requires the classification error of the weak learner to be bounded away from 0.5 for arbitrary re-weighting of the data; such a strong condition is NP-hard for a neural network learner to achieve. Our weak learner avoids the hardness by assuming separability, and secures the polynomial complexity by agnostic learning. It draws a theoretical connection between neural network learning and boosting.

With the aim of understanding the fundamental limits of our learnability problem, we show that the time-complexity guarantees for both algorithms are unimprovable under their respective assumptions. Under the assumption that $\mathbf{RP} \neq \mathbf{NP}$, we prove that the agnostic learning algorithm's exponential complexity in $1/\epsilon$ cannot be avoided. More precisely, we demonstrate that there is no algorithm achieving arbitrary excess risk $\epsilon > 0$ in poly$(n, d, 1/\epsilon)$ time. On the other hand, we demonstrate that the BoostNet algorithm's exponential complexity in $1/\gamma$ is unimprovable as well—in particular, by showing that a poly$(d, 1/\epsilon, 1/\gamma)$ complexity is impossible for any algorithm under a certain cryptographic assumption.

Finally, we report two empirical results on the BoostNet algorithm. The first experiment is a classical problem in computational learning theory, called *learning parity functions with noise*. We show that BoostNet learns a two-layer neural network that encodes the correct function, while the performance of backpropagation is as poor as random guessing. The second experiment is digit recognition on the MNIST dataset, where we show that BoostNet consistently outperforms backpropagation for training neural networks with the same number of hidden nodes.

Other related work Several recent papers address the challenge of establishing polynomial-time learnability for neural networks [1, 25, 9, 28, 17]. Sedghi and Anandkumar [25] and Janzamin et al. [9] study the supervised learning of neural networks under the assumption that the score function of the data distribution is known. They show that by certain computations on the score function, the first network layer can be learned by a polynomial-time algorithm. In contrast, our algorithms does not requires knowledge of the data distribution. Another approach to the problem is via improper learning, in which case the goal is to find a predictor that need not be a neural network, but performs as well as the best possible neural network in terms of the generalization error. Livni et al. [17] propose a polynomial-time algorithm to learn networks whose activation function is quadratic. Zhang et al. [28] propose an algorithm for the improper learning of sigmoidal neural networks. The algorithm is based on the kernel method, and so its output does not characterize to the parameters of a neural network. In contrast, our method learns the model parameters explicitly.

2 Problem set-up

Let $\mathcal{D}$ be a dataset containing n points $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathcal{X} \subset \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. The goal is to learn

a function $f : \mathcal{X} \to \mathbb{R}$ so that $f(x_i)$ is as close to y_i as possible. We may write the loss function as

$$\ell(f) := \sum_{i=1}^{n} \alpha_i h(-y_i f(x_i)). \tag{1}$$

where $h_i : \mathbb{R} \to \mathbb{R}$ is a L-Lipschitz continuous function that depends on y_i, and $\{\alpha_1, \ldots, \alpha_n\}$ are non-negative importance weights that sum to one. In this paper, we study the minimization of the loss function in equation (1) when f is a multi-layer neural network.

Next, we formalize the function class of multi-layer neural networks. Given two numbers $p \in (1, 2]$ and $q \in [2, \infty)$ such that $1/p + 1/q = 1$, we assume that the input vector satisfies $\|x_i\|_q \leq 1$ for every $i \in [n]$. The class of m-layer neural networks is recursively defined in the following way. A one-layer neural network is a linear mapping from $\mathbb{R}^d$ to $\mathbb{R}$, and we consider the set of mappings:

$$\mathcal{N}_1 := \{x \to \langle w, x \rangle : \ \|w\|_p \leq B\}.$$

For $m > 1$, an m-layer neural network is a linear combination of $(m-1)$-layer neural networks activated by a sigmoid function, and so we define:

$$\mathcal{N}_m := \Big\{ x \to \sum_{j=1}^{d} w_j \sigma(f_j(x)) : \ d < \infty, \\ f_j \in \mathcal{N}_{m-1}, \ \|w\|_1 \leq B \Big\}.$$

In this definition, the function $\sigma : \mathbb{R} \to [-1, 1]$ is an arbitrary 1-Lipschitz continuous function. At each hidden layer, we allow the number of neurons d to be arbitrarily large, but the per-unit ℓ_1-norm must be bounded by a constant B. This particular regularization scheme has been studied a number of researchers in past work (e.g., [3, 11, 4, 21]).

Assuming a constant ℓ_1-norm bound might be restrictive for some applications, but without this constraint, it is known that the neural network class activated by any sigmoid-like or ReLU-like function *cannot* be learned in polynomial time [28]. On the other hand, ℓ_1-regularization imposes sparsity on the neural network. It is observed in practice that sparse neural networks such as convolutional nets are capable of learning meaningful representations. Moreover, it has been argued that sparse connectivity is a natural constraint that can lead to improved performance in practice [see, e.g., 27].

Throughout this paper, we use $[n]$ to denote the set of indices $\{1, 2, \ldots, n\}$. For $q \in [1, \infty)$, let $\|x\|_q$ denote the ℓ_q-norm of vector x, given by $\|x\|_q := (\sum_{j=1}^{d} x_j^q)^{1/q}$. If $u \in \mathbb{R}^d$ and $\sigma : \mathbb{R} \to \mathbb{R}$ is a function, we use $\sigma(u)$ as a convenient shorthand for the vector $(\sigma(u_1), \ldots, \sigma(u_d)) \in \mathbb{R}^d$. Given a class $\mathcal{F}$ of real-valued functions, we define the new function class $\sigma \circ \mathcal{F} := \{\sigma \circ f \mid f \in \mathcal{F}\}$.

3 Positive results for learnability

In this section, we present two positive results for learning fully-connected neural networks. The first result, applying to arbitrary data distribution, is achieved by an *agnostic learning algorithm*. The second result makes a separability assumption on the data distribution to achieve stronger theoretical guarantees.

3.1 Agnostic learning

In the agnostic setting, our goal is to compute a neural network $\widehat{f} \in \mathcal{N}_m$ that minimizes the loss function over the space of all given networks. Letting $f^* \in \mathcal{N}_m$ be the network that minimizes the empirical loss ℓ, we now present and analyze a method (see Algorithm 1) that computes a network whose loss is at most ϵ worse that that of f^*. We first state our main guarantee for this algorithm, before providing intuition. More precisely, for any $\epsilon, \delta > 0$, the following theorem applies to Algorithm 1 with the choices:

$$k := \lceil q/\epsilon^2 \rceil, \quad s := \lceil 1/\epsilon^2 \rceil, \quad \text{and} \\ T := \left\lceil 5(4/\epsilon)^{k(s^m-1)/(s-1)} \log(1/\delta) \right\rceil. \tag{3}$$

Theorem 1. *For given $B \geq 1$ and $\epsilon, \delta > 0$, with the choices of (k, s, T) given above, Algorithm 1 outputs a predictor $\widehat{f} \in \mathcal{N}_m$ such that*

$$\ell(\widehat{f}) \leq \ell(f^*) + (2m + 9)\epsilon L B^m, \tag{4}$$

with probability at least $1 - \delta$. The computational complexity is bounded by poly$(n, d, e^{q(1/\epsilon^2)^m \log(1/\epsilon)}, \log(1/\delta))$.

We remark that if $m = 1$, then $\mathcal{N}_m$ is a class of linear mappings. Thus, a special case of Algorithm 1 learns a linear classifier under a (potentially) non-convex loss. For the general case, as long as the number of layer m, the Lipschitz constant L, the norm bound B and the target optimality gap ϵ are constants, the algorithm's computation complexity is polynomial in (n, d). This complexity bound is non-trivial because even for linear classifiers, the number of parameter is proportional to the dimension d, thus the probability that a random classifier succeeds is exponentially small in dimension d.

Instead of initializing the model parameters uniformly at random, Algorithm 1 generates a sequence of random vectors u from a k-dimensional space ($k \ll d$). Then it maps the vector u to the parameter space by solving a least-squares problem. It can be shown that the probability of getting a good enough initialization with this method only depends on k, independent of the number of parameters. Thus we are able to avoid the exponential dependence on the input dimension.

Underlying intuition: Before presenting the proof of Theorem 1, we describe the intuition underlying the algorithm. Each iteration involves resampling k independent

Algorithm 1 The agnostic learning algorithm

Input: Feature-label pairs $\{(x_i, y_i)\}_{i=1}^n$; number of layers m; parameters k, s, T, B.

For $t = 1, 2, \ldots, T$:

1. Sample k points $\{(x'_j, y'_j)\}_{j=1}^k$ from the dataset with respect to their importance weights.
2. Generate a neural network $g \in \mathcal{N}_m$ in the following recursive way:
   - If $m = 1$, then draw a vector u uniformly from $[-B, B]^k$. Let $v := \arg\min_{w \in \mathbb{R}^d : \|w\|_p \le B} \sum_{j=1}^k (\langle w, x'_j \rangle - u_j)^2$ and return $g : x \to \langle v, x \rangle$.
   - If $m > 1$, then generate the $(m-1)$-layer networks $g_1, \ldots, g_s \in \mathcal{N}_{m-1}$ using this recursive program. Draw a vector u uniformly from $[-B, B]^k$. Let
   $$v := \arg \min_{w \in \mathbb{R}^s : \|w\|_1 \le B} \sum_{j=1}^k \Big(\sum_{l=1}^s w_l \sigma(g_l(x'_j)) - u_j \Big)^2 \tag{2}$$
   and return $g : x \to \sum_{l=1}^s v_l \sigma(g_l(x))$.
3. Choose $f_t := g$, or compute f_t by a poly(n, d)-time algorithm such that $f_t \in \mathcal{N}_m$ and $\ell(f_t) \le \ell(g)$.

Output: $\widehat{f} := \arg\min_{f \in \{f_1, \ldots, f_T\}} \ell(f)$.

points from the dataset. Rademacher complexity theory implies that minimizing the loss function $G(f)$—an empirical loss that only depends on the k random samples—will approximately minimizes the original loss $\ell(f)$ if the sample size $k = \Omega(1/\epsilon^2)$. The value of $G(f)$ is uniquely determined by the vector $\varphi(f) := (f(x'_1), \ldots, f(x'_k))$. As a consequence, if we draw $u \in [-B, B]^k$ sufficiently close to $\varphi(f^*)$, then a nearly-optimal neural network will be obtained by approximately solving $\varphi(g) \approx u$, or equivalently by solving $\varphi(g) \approx \varphi(f^*)$.

In general, directly solving the equation $\varphi(g) \approx u$ would be difficult even if the vector u were known. In particular, since our class $\mathcal{N}_m$ is highly non-linear, solving this equation cannot be reduced to solving a convex program unless $m = 1$. To address this difficulty, suppose that we can write f^* as $f^*(x) = \sum_{l=1}^s w_l \sigma(f_l^*(x))$ for some functions $f_l^* \in \mathcal{N}_m$. The problem becomes much easier if the quantities $\sigma(f_l^*(x'_j))$ are already known for every $(j, l) \in [k] \times [d]$. With this perspective in mind, we can approximately solve the equation $\varphi(\sum_{l=1}^s w_l \sigma(f_l^*(x))) = u$ by minimizing:

$$\min_{w \in \mathbb{R}^d : \|w\|_1 \le B} \sum_{j=1}^k \Big(\sum_{l=1}^s w_l \sigma(f_l^*(x'_j)) - u_j \Big)^2. \tag{5}$$

Accordingly, suppose that we draw vectors $a_1, \ldots, a_s \in [-B, B]^k$ such that each a_j is sufficiently close to $\varphi(f_j^*)$—any such draw is called "successful"—then we may then recursively compute $(m-1)$-layer networks $g_1, \ldots, g_s$ by first solving the approximate equation $\varphi(g_l) \approx a_l$ (and, as a consequence, we have $\varphi(g_l) \approx \varphi(f_j^*)$). We then rewrite problem (5) as

$$\min_{w \in \mathbb{R}^d : \|w\|_1 \le B} \sum_{j=1}^k \Big(\sum_{l=1}^s w_l \sigma(g_l(x'_j)) - u_j \Big)^2.$$

This ℓ_1-constrained least-squares problem matches step (2) in the algorithm, and can be efficiently solved by convex optimization. Note that the probability of a successful draw $\{a_1, \ldots, a_s\}$ depends on the dimension s. Although there is no constraint on the dimension of $\mathcal{N}_m$, the Maurey-Barron-Jones lemma (see Appendix A.3) asserts that it suffices to choose $s = \mathcal{O}(1/\epsilon^2)$, thus the probability of success can be lower bounded.

It is important to note that in each iteration of the algorithm, the network g is sampled from a low-dimensional manifold whose dimension is characterized by the hyper-parameters (k, s). Since we choose $k = \Theta(1/\epsilon^2)$ and $s = \Theta(1/\epsilon^2)$, independent of the scale of the network, the iteration complexity has a polynomial dependence on (n, d).

Proof of Theorem 1: Due to space constraints, we present only the high-level logic of the proof, deferring proofs of the technical lemmas to the appendix.

We start by defining the Rademacher complexity of a function class. Given the input dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, let $\{(x'_j, y'_j)\}_{j=1}^k$ be a set of k i.i.d. samples drawn from $\mathcal{D}$ such that the probability of drawing (x_i, y_i) is proportional to α_i. For function $f : \mathcal{X} \to \mathbb{R}$, we define the subsample-based loss function:

$$G(f) := \frac{1}{k} \sum_{j=1}^k h(-y'_j f(x'_j)). \tag{6}$$

It is straightforward to verify that $\mathbb{E}[G(f)] = \ell(f)$. For a given function class $\mathcal{F}$, the Rademacher complexity of $\mathcal{F}$ with respect to these k samples is defined as

$$R_k(\mathcal{F}) := \mathbb{E}\Big[\sup_{f \in \mathcal{F}} \frac{1}{k} \sum_{j=1}^k \varepsilon_j f(x'_j) \Big], \tag{7}$$

where the $\{\varepsilon_j\}$ are independent Rademacher random variables.

Lemma 1. *Given a loss that is L-Lipschitz and a function class $\mathcal{F}$ that contains the constant zero function $f(x) \equiv 0$, we have*

$$\mathbb{E}\Big[\sup_{f\in\mathcal{F}} |G(f) - \ell(f)|\Big] \leq 4LR_k(\mathcal{F}).$$

This lemma is based on a slight sharpening of the usual Ledoux-Talagrand contraction for Rademacher variables [15]; see Appendix A.1 for the proof.

Lemma 1 shows that the Rademacher complexity $R_k(\mathcal{F})$ controls the distance between $G(f)$ and $\ell(f)$. For the neural network classes studied in this paper, we have $R_k(\mathcal{N}_m) = \mathcal{O}(1/\sqrt{k})$. Thus, the function $G(f)$ should be a good approximation to $\ell(f)$ as long as the sample size is large enough. This intuition is formalized by the following lemma:

Lemma 2. *The Rademacher complexity of $\mathcal{N}_m$ is bounded as $R_k(\mathcal{N}_m) \leq \sqrt{\frac{q}{k}}\, B^m$.*

See Appendix A.2 for the proof.

Given these preliminary results, let us now focus on a single iteration of Algorithm 1, and study the random network $g \in \mathcal{N}_m$ that it generates. Conditioning on the random samples $\{(x'_j, y'_j)\}$, consider the empirical loss $G(f)$ defined in equation (6). Since the function h is L-Lipschitz, we have

$$h(-y'_j g(x'_j)) - h(-y'_j f^*(x'_j)) \leq L|g(x'_j) - f^*(x'_j)| \quad \text{for any } j \in [k]. \tag{8}$$

For any function f, we use $\varphi(f)$ as a shorthand notation for the vector $(f(x'_1), \ldots, f(x'_k))$. Using this notation, we have the equivalence

$$\|\varphi(g) - \varphi(f^*)\|_2 = (\sum_{j=1}^{k} |g(x'_j) - f^*(x'_j)|^2)^{1/2}.$$

Inequality (8) and the Cauchy-Schwarz inequality imply that

$$\begin{aligned} G(g) - G(f^*) &\leq \frac{L}{k}\sum_{j=1}^{k} |g(x'_j) - f^*(x'_j)| \\ &\leq \frac{L}{\sqrt{k}}\|\varphi(g) - \varphi(f^*)\|_2. \end{aligned}$$

Thus, in order to bound the distance between the empirical loss with respect to functions g and f^*, it suffices to bounded the ℓ_2-distance between two vectors $\varphi(g)$ and $\varphi(f)$. Such a bound is provided by the following:

Lemma 3. *For any fixed function $f^* \in \mathcal{N}_m$ and any sample set $\{(x'_j, y'_j)\}_{j=1}^k$, the random network g satisfies*

$$\|\varphi(g) - \varphi(f^*)\|_2 \leq (2m-1)\epsilon\sqrt{k}B^m,$$

with probability at least $p_m := (\frac{\epsilon}{4})^{k(s^m-1)/(s-1)}$.

Lemma 3 is crucial to our analysis. It shows that with a probability independent of (n, d), the algorithm generates a random network g that approximates the best possible neural network f^* . The proof uses the intuition about Algorithm 1 that we described earlier, and exploits the recursive definition of the neural network class. See Appendix A.3 for the proof.

We are now equipped to complete the proof of Theorem 1. Lemma 1 implies that

$$\mathbb{E}\Big[\sup_{f\in\mathcal{N}_m} \big|\ell(f) - G(f)\big|\Big] \leq 4LR_k(\mathcal{N}_m).$$

By Markov's inequality, we have $\sup_{f\in\mathcal{N}_m} \big|\ell(f) - G(f)\big| \leq 5LR_k(\mathcal{N}_m)$ with probability at least $1/5$. This event only depends on the choice of $\{(x'_j, y'_j)\}$. Whenever this event holds, we are guaranteed that

$$\begin{aligned} \ell(g) &\leq G(g) + 5LR_k(\mathcal{N}_m) \\ &= G(f^*) + (G(g) - G(f^*)) + 5LR_k(\mathcal{N}_m) \\ &\leq \ell(f^*) + \frac{L}{\sqrt{k}}\|\varphi(g) - \varphi(f^*)\|_2 + 10LR_k(\mathcal{N}_m). \end{aligned}$$

By Lemma 2, we have $R_k(\mathcal{N}_m) \leq \sqrt{\frac{q}{k}}\prod_{l=1}^m B_l$. Thus, setting $k = q/\epsilon^2$, substituting the bound into Lemma 3 and simplifying yields

$$\ell(g) \leq \ell(f^*) + (2m+9)\epsilon LB^m,$$

with probability at least $p_m/5$. If we repeat the procedure for $T = (5/p_m)\log(1/\delta)$ times, then the desired bound holds with probability at least $1-\delta$. The time complexity is obtained by plugging in the choices of (s, k, T). ■

3.2 Learning with separable data

We turn to the case in which the data are separable with a positive margin. Throughout this section, we assume that the activation function of $\mathcal{N}_m$ is an odd function (i.e., $\sigma(-x) = -\sigma(x)$). We note that this assumption can be removed by using a slightly different algorithm. We say that a given data set $\{(x_i, y_i)\}_{i=1}^n$ is *separable with margin* γ, or γ-separable for short, if there is a network $f^* \in \mathcal{N}_m$ such that $y_i f^*(x_i) \geq \gamma$ for each $i \in [n]$. Given a distribution $\mathbb{P}$ over the space $\mathcal{X} \times \{-1, 1\}$, we say that it is γ-separable if there is a network $f^* \in \mathcal{N}_m$ such that $yf^*(x) \geq \gamma$ almost surely (with respect to $\mathbb{P}$).

Algorithm 2 learns a neural network on the separable data. It uses the AdaBoost approach [7] to construct the network, and we refer to it as the *BoostNet algorithm*. In each iteration, it trains a shallower network $\widehat{g}_t \in \mathcal{N}_{m-1}$ with an error rate slightly better than random guessing, then adds it to the classifier to construct an m-layer network. The shallow network is trained by the agnostic learning algorithm described in Section 3.1. It is worth noting that in each iteration, the network $\widehat{g}_t$ is trained on a reweighted version of the data. The goal is to focus on instances that were not

Algorithm 2 The BoostNet algorithm for learning with separable data

Input: Feature-label pairs $\{(x_i, y_i)\}_{i=1}^n$; number of layers $m \geq 2$; parameters δ, γ, T, B.

Initialize $f_0 = 0$ and $b_0 = 0$. For $t = 1, 2, \ldots, T$:

1. Define $G_t(g) := \sum_{i=1}^n \alpha_{t,i} \sigma(-y_i g(x_i))$ where $g \in \mathcal{N}_{m-1}$ and $\alpha_{t,i} := \frac{e^{-y_i \sigma(f_{t-1}(x_i))}}{\sum_{j=1}^n e^{-y_i \sigma(f_{t-1}(x_i))}}$.
2. Compute $\widehat{g}_t \in \mathcal{N}_{m-1}$ by Algorithm 1 such that

$$G_t(\widehat{g}_t) \leq \inf_{g \in \mathcal{N}_{m-1}} G_t(g) + \frac{\gamma}{2B} \tag{9}$$

   with probability at least $1 - \delta/T$*. Let $\mu_t := \max\{-\frac{1}{2}, G_t(\widehat{g}_t)\}$.
3. Set $f_t = f_{t-1} + \frac{1}{2}\log(\frac{1-\mu_t}{1+\mu_t})\widehat{g}_t$ and $b_t = b_{t-1} + \frac{1}{2}\left|\log(\frac{1-\mu_t}{1+\mu_t})\right|$.

Output: $\widehat{f} := \frac{B}{b_T} f_T$.

correctly handled by earlier networks.

The following theorem provides guarantees for its performance when the algorithm is run for

$$T := \left\lceil \frac{16B^2 \log(n+1)}{\gamma^2} \right\rceil$$

iterations. The running time depends on a quantity $C_{m,B,1/\gamma}$ that is a constant for any choice of the triple $(m, B, 1/\gamma)$, but with exponential dependence on $1/\gamma$.

Theorem 2. *With the above choice of T, the algorithm achieves:*

(a) In-sample error: For any γ-separable dataset $\{(x_i, y_i)\}_{i=1}^n$ Algorithm 2 outputs a neural network $\widehat{f} \in \mathcal{N}_m$ such that $y_i \widehat{f}(x_i) \geq \frac{\gamma}{16}$ for every $i \in [n]$, with probability at least $1 - \delta$. The time complexity is bounded by poly$(n, d, \log(1/\delta), C_{m,B,1/\gamma})$.

(b) Generalization error: Given a data set consisting of $n =$ poly$(1/\epsilon, \log(1/\delta))$ i.i.d. samples from any γ-separable distribution $\mathbb{P}$, Algorithm 2 outputs a network $\widehat{f} \in \mathcal{N}_m$ such that

$$\mathbb{P}\big[\text{sign}(\widehat{f}(x)) \neq y\big] \leq \epsilon \tag{10}$$

with probability at least $1 - 2\delta$. The time complexity is bounded by poly$(d, 1/\epsilon, \log(1/\delta), C_{m,B,1/\gamma})$.

The majority of the technical work is devoted to proving part (a). The generalization bound in part (b) follows by combining part (a) with bounds on the Rademacher complexity of the network class, which then allows us to translate the in-sample error bound to generalization error in the usual way. It is worth comparing the BoostNet algorithm with the agnostic learning algorithm. In order to bound the generalization error by $\epsilon > 0$, the time complexity of the agnostic learning algorithm is exponential in $1/\epsilon$, while for BoostNet we obtain the polynomial complexity.

A key step for proving part (a) is to use the equivalence between separability and weak learnability [26]. More precisely, if the dataset is γ-separable, then with any α_t-reweighting on the data, there exists a $(m-1)$-layer network g^* such that the classification loss of $\sigma \circ g^*$ is upper bounded by:

$$\sum_{i=1}^n \alpha_{t,i}(-y_i)(\sigma \circ g^*(x_i)) \leq -\frac{\gamma}{B}. \tag{11}$$

The inequality (9) in Algorithm 2 implies that the loss of $\sigma \circ \widehat{g}_t$ is at most $\frac{\gamma}{2B}$ worse than that of $\sigma \circ g^*$. Combining with inequality (11), it implies that the classifier $\sigma \circ \widehat{g}_t$ outperforms random guessing by a constant margin, which is good enough for AbaBoost to establish part (a). For the computation complexity, since the target optimality gap $\frac{\gamma}{2B}$ is a constant, Theorem 1 shows that $\widehat{g}_t$ can be computed in poly$(n, d, \log(1/\delta))$ time. See Appendix B for a complete proof.

In practice, real data are often noisy so that the separability assumption is not likely to hold. A more realistic assumption would be that the "true labels" are separable, but the observed labels are corrupted by a random noise, hence not separable. We show that the learnability results of Theorem 2 still hold for such noisy data.

Formally, for every pair (x, y) sampled from a γ-separable distribution, suppose that the learning algorithm actually receives the corrupted pair $(x, \widetilde{y})$, where

$$\widetilde{y} = \begin{cases} y & \text{with probability } 1 - \eta, \\ -y & \text{with probability } \eta. \end{cases}$$

Here the parameter $\eta \in [0, \frac{1}{2})$ corresponds to the noise level. Since the labels are flipped, the BoostNet algorithm cannot be directly applied. However, we can use the improper learning algorithm of [28] to learn an improper classifier $\widehat{h}$ such that $\widehat{h}(x) = y$ with high probability, and then

*We may choose the hyper-parameters of Algorithm 1 by equation (3), with the additive error ϵ defined by $\gamma/((4m + 10)LB^m)$. Theorem 1 guarantees that the error bound (9) holds with high probability.

apply the BoostNet algorithm taking $(x, \widehat{h}(x))$ as input. Doing so yields the following guarantee:

Corollary 1. *Assume that $q = 2$ and $\eta < 1/2$. For any constant (m, B), consider the neural network class $\mathcal{N}_m$ activated by $\sigma(x) := \mathrm{erf}(x)$[†]. Given a random dataset of size $n = \mathrm{poly}(1/\epsilon, 1/\delta)$ for any γ-separable distribution, there is a $\mathrm{poly}(d, 1/\epsilon, 1/\delta)$-time algorithm that outputs a network $\widehat{f} \in \mathcal{N}_m$ such that*

$$\mathbb{P}(\mathrm{sign}(\widehat{f}(x)) \neq y) \leq \epsilon,$$

with probability at least $1 - \delta$.

See Appendix C for the proof.

4 Hardness results

Given the agnostic learning algorithm's exponential dependence on $1/\epsilon$, and the BoostNet algorithm's exponential dependence on $1/\gamma$, it is natural to wonder if we can find better algorithms that avoid these exponential dependences. In this section, we present two hardness results demonstrating that it is impossible given classical hardness assumptions.

4.1 Hardness of agnostic learning

Since our interest is to prove a lower bound, it suffices to study a special case of the general minimization problem (1) —namely training a two-layer neural network with one hidden neuron activated by the ReLU function. In that case, the function f can be written as $f(x) = \max\{0, \langle w, x\rangle\}$ for some vector $w \in \mathbb{R}^d$. If we optimize the linear loss $h(x) := -x$, then the loss function can be written as

$$\ell(w) := \sum_{i=1}^{n} \alpha_i y_i \max\{0, \langle w, x_i\rangle\}. \tag{12}$$

We study the case where $\|w^*\|_2 \leq 1$ and $\|x_i\|_2 \leq 1$, $y_i = -1$, and $\alpha_i = 1/n$ for any $i \in [n]$. The following proposition shows that approximately minimizing the loss function is hard.

Proposition 1. *Unless $\mathbf{RP} = \mathbf{NP}$[‡], there is no randomized $\mathrm{poly}(n, d, 1/\epsilon)$-time algorithm computing a vector $\widehat{w}$ which satisfies $\|\widehat{w}\|_2 \leq 1$ and $\ell(\widehat{w}) \leq \ell(w^*) + \epsilon$ with probability at least $1/2$.*

Proposition 1 provides a strong evidence that training neural networks with respect to a non-convex loss cannot be done in $\mathrm{poly}(n, d, 1/\epsilon)$ time. The proposition is proved by reducing to the NP-hardness of the MAX-2-SAT problem [23]. See Appendix D for the proof.

[†]The erf function can be replaced by any function σ satisfying polynomial expansion $\sigma(x) = \sum_{j=0}^{\infty} \beta_j x^j$, such that $\sum_{j=0}^{\infty} 2^j \beta_j^2 \lambda^{2j} < +\infty$ for any finite $\lambda \in \mathbb{R}^+$.

[‡]The class $\mathbf{RP}$ refers to all randomized polynomial-time algorithms.

4.2 Hardness of γ-separable problems

We prove the hardness of γ-separable problems by reducing to the hardness of the standard PAC learning of the intersection of halfspaces given in Klivans et al. [10]. More precisely, consider the family of halfspace indicator functions mapping $\mathcal{X} = \{-1, 1\}^d$ to $\{-1, 1\}$ given by

$$H = \{x \to \mathrm{sign}(w^T x - b - 1/2) : x \in \{-1, 1\}^d, \\ b \in \mathbb{N},\ w \in \mathbb{N}^d,\ |b| + \|w\|_1 \leq \mathrm{poly}(d)\}.$$

Given a T-tuple of functions $\{h_1, \ldots, h_T\}$ belonging to H, we define the intersection function

$$h(x) = \begin{cases} 1 & \text{if } h_1(x) = \cdots = h_T(x) = 1, \\ -1 & \text{otherwise,} \end{cases}$$

which represents the intersection of T half-spaces. Letting H_T denote the set of all such functions, for any distribution on $\mathcal{X}$, we want an algorithm taking a sequence of $(x, h^*(x))$ as input where x is a sample from $\mathcal{X}$ and $h^* \in H_T$. It should output a function $\widehat{h}$ such that $\mathbb{P}(\widehat{h}(x) \neq h^*(x)) \leq \epsilon$ with probability at least $1 - \delta$. If there is such an algorithm whose sample complexity and time complexity scale as $\mathrm{poly}(d, 1/\epsilon, 1/\delta)$, then we say that H_T is efficiently learnable. Klivans et al. [10] show that if $T = \Theta(d^\rho)$, then the class H_T is not efficiently learnable under a certain cryptographic assumption. This hardness statement implies the hardness of learning a certain class of neural networks with a separable data. In particular, the following proposition applies to the class of two-layer neural networks $\mathcal{N}_2$ activated by the piecewise linear function $\sigma(x) := \min\{1, \max\{-1, x\}\}$ or the ReLU function $\sigma(x) := \max\{0, x\}$, and with the norm constraint $B = 2$.

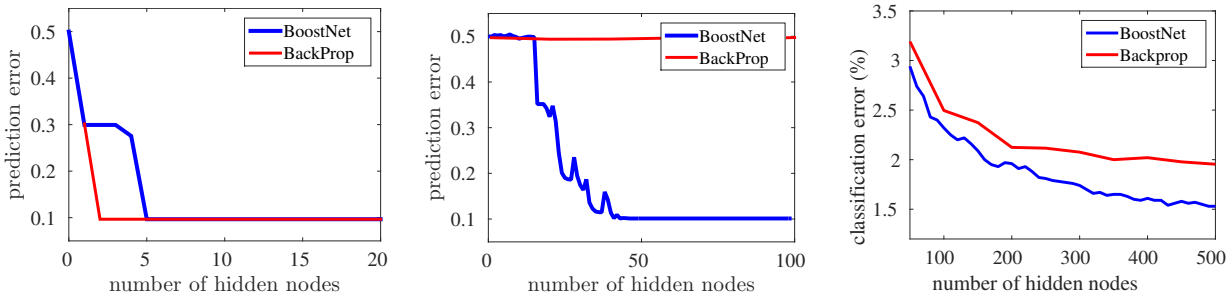
Proposition 2. *Consider the above function class, and assume that H_T is not efficiently learnable for $T = \Theta(d^\rho)$. Consider any algorithm such that when applied to any γ-separable data distribution, it is guaranteed to output a neural network $\widehat{f}$ satisfying $\mathbb{P}(\mathrm{sign}(\widehat{f}(x)) \neq y) \leq \epsilon$ with probability at least $1 - \delta$. Then it cannot terminate in $\mathrm{poly}(d, 1/\epsilon, 1/\delta, 1/\gamma)$-time.*

See Appendix E for the proof.

5 Experiments

In this section, we compare BoostNet with the backpropagation approach for training two-layer neural networks.

Learning parity function with noise The learning of parity functions is a classical problem in computational learning theory [see, e.g., 6]. We construct an instance of this problem based on a dataset with $n = 50,000$ points. Each point (x, y) is generated as follows: first, the vector x is uniformly drawn from $\{-1, 1\}^d$ and concatenated with a constant 1 as the $(d+1)$-th coordinate. The label is generated as follows: for some unknown subset of p indices



(a) parity function ($p = 2$) (b) parity function ($p = 5$) (c) MNIST digit recognition

Figure 1. Performance of BoostNet and backpropagation for training two-layer neural networks. The testing errors are plotted as a function of the number of hidden nodes.

$1 \le i_1 < \cdots < i_p \le d$, we set

$$y = \begin{cases} x_{i_1} x_{i_2} \dots x_{i_p} & \text{with probability } 0.9\,, \\ -x_{i_1} x_{i_2} \dots x_{i_p} & \text{with probability } 0.1\,. \end{cases}$$

The goal is to learn a function $f : \mathbb{R}^{d+1} \to \mathbb{R}$ such that $\mathrm{sign}(f(x))$ predicts the value of y. The optimal rate is achieved by the parity function $f(x) = x_{i_1} x_{i_2} \dots x_{i_p}$, in which case the prediction error is 0.1. For parity degrees $p > 1$, it is easy to verify that the optimal rate can be achieved by a two-layer neural network, but it is impossible to achieve using any linear classifier.

We performed experiments in dimension $d = 50$, with parities $p \in \{2, 5\}$, and the activation function $\sigma(x) := \tanh(x)$. The training set, the validation set and the test set contain respectively 25K, 5K and 20K points. In order to train a two-layer BoostNet, we choose the hyper-parameter $B = 10$, and use Algorithm 1 as the subroutine to train shallow networks with hyper-parameters $(k, T) = (10, 1)$. In order to train a classical two-layer neural network, we use the random initialization scheme of Nguyen and Widrow [22] and the backpropagation algorithm of Møller [20]. We execute the algorithms for ten independent rounds to select the best solution.

Figure 4.1 (a)–(b) compares the prediction errors of BoostNet and backpropagation. It shows that both algorithms correctly learn the degree-2 parity function with a few hidden nodes. In contrast, BoostNet learns the degree-5 parity function with less than 50 hidden nodes, while the performance of backpropagation is no better than random guessing. This suggests that the BoostNet algorithm is less likely to be trapped in a bad local optimum.

Hand-written digit recognition In our second experiment, we train a two-layer neural network for hand-written digit recognition on the MNIST dataset [14]. We take 60k images for training and 10k images for testing. For the BoostNet algorithm, we train one-versus-all classifiers for the ten classes with hyper-parameter $B = 20$. The subroutine Algorithm 1 takes hyper-parameters $(k, T) = (100, 1)$ and uses mini-batch SGD for optimization. For the backpropagation approach, we train a ten-class classifier that minimizes the cross-entropy loss. Figure 4.1 (c) plots the testing error as a function of the number of hidden nodes included. It is observed that with the same number of hidden nodes, BoostNet consistently outperforms backpropagation. In particular, with 500 hidden nodes, BoostNet achieves an error rate of 1.53%, while backpropagation achieves an error rate of 1.95%. It shows that for this problem, the BoostNet algorithm learns a better model on the same neural network architecture.

6 Conclusion

In this paper, we have presented two learnability results for learning neural networks with non-convex loss functions. For agnostic learning, we demonstrated that the time complexity is polynomial in the input dimension and in the sample size but exponential in the excess risk. Under the separability condition, we show that the exponential dependence on the excess risk can be removed. We also show that these complexity bounds are unimprovable under their respective assumptions. We hope that these results improve the understanding of learnability for deep learning networks, and shed light on the usage of sampling and ensemble methods in deep learning.

Acknowledgements

MW and YZ were partially supported by NSF grant CIF-31712-23800 from the National Science Foundation, AFOSR-FA9550-14-1-0016 grant from the Air Force Office of Scientific Research, ONR MURI grant N00014-11-1-0688 from the Office of Naval Research. MJ was supported by the Office of Naval Research under grant number N00014-15-1-2670. We thank Sivaraman Balakrishnan for helpful comments.

References

[1] S. Arora, A. Bhaskara, R. Ge, and T. Ma. Provable bounds for learning some deep representations. In *ICML*, pages 584–592, 2014.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

[3] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *Information Theory, IEEE Transactions on*, 44 (2):525–536, 1998.

[4] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *The Journal of Machine Learning Research*, 3:463–482, 2003.

[5] A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992.

[6] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, 2003.

[7] 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.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[9] M. Janzamin, H. Sedghi, and A. Anandkumar. Generalization bounds for neural networks through tensor factorization. *arXiv:1506.08473*, 2015.

[10] A. R. Klivans, A. Sherstov, et al. Cryptographic hardness for learning intersections of halfspaces. In *47th Annual IEEE Symposium on Foundations of Computer Science*, pages 553–562. IEEE, 2006.

[11] V. Koltchinskii and D. Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics*, pages 1–50, 2002.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 24, pages 1097–1105, 2012.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits, 1998.

[15] M. Ledoux and M. Talagrand. *Probability in Banach Spaces: Isoperimetry and Processes*, volume 23. Springer Science & Business Media, 2013.

[16] W. S. Lee, P. L. Bartlett, and R. C. Williamson. Efficient agnostic learning of neural networks with bounded fan-in. *IEEE Transactions on Information Theory*, 42(6):2118–2132, 1996.

[17] R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, volume 26, pages 855–863, 2014.

[18] W. Maass. Agnostic PAC learning of functions on analog neural nets. *Neural Computation*, 7(5):1054–1078, 1995.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[20] M. F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4): 525–533, 1993.

[21] B. Neyshabur, R. Tomioka, and N. Srebro. Norm-based capacity control in neural networks. In *COLT*, pages 1376–1401, 2015.

[22] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *International Joint Conference on Neural Networks*, pages 21–26, 1990.

[23] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[24] H. Schwenk and Y. Bengio. Boosting neural networks. *Neural Computation*, 12(8):1869–1887, 2000.

[25] H. Sedghi and A. Anandkumar. Provable methods for training neural networks with sparse connectivity. *arXiv:1412.2693*, 2014.

[26] S. Shalev-Shwartz and Y. Singer. On the equivalence of weak learnability and linear separability: New relaxations and efficient boosting algorithms. *Machine Learning*, 80(2-3):141–163, 2010.

[27] M. Thom and G. Palm. Sparse activity and sparse connectivity in supervised learning. *The Journal of Machine Learning Research*, 14(1):1091–1143, 2013.

[28] Y. Zhang, J. D. Lee, and M. I. Jordan. l1-regularized neural networks are improperly learnable in polynomial time. *ICML*, 2016.