ℓ_1-regularized Neural Networks are Improperly Learnable in Polynomial Time

Yuchen Zhang   YUCZHANG@EECS.BERKELEY.EDU
Jason D. Lee   JASONDLEE@BERKELEY.EDU
Michael I. Jordan   JORDAN@CS.BERKELEY.EDU
Department of EECS, University of California, Berkeley, CA 94720 USA

Abstract

We study the improper learning of multi-layer neural networks. Suppose that the neural network to be learned has k hidden layers and that the ℓ_1-norm of the incoming weights of any neuron is bounded by L. We present a kernel-based method, such that with probability at least $1 - \delta$, it learns a predictor whose generalization error is at most ϵ worse than that of the neural network. The sample complexity and the time complexity of the presented method are polynomial in the input dimension and in $(1/\epsilon, \log(1/\delta), F(k, L))$, where $F(k, L)$ is a function depending on (k, L) and on the activation function, independent of the number of neurons. The algorithm applies to both sigmoid-like activation functions and ReLU-like activation functions. It implies that any sufficiently sparse neural network is learnable in polynomial time.

1. Introduction

Neural networks have been successfully applied in many areas of artificial intelligence, such as image classification, face recognition, speech recognition and natural language processing. Practical successes have been driven by the rapid growth in the size of data sets and the increasing availability of large-scale parallel and distributed computing platforms. Examples of recent work in this area include (Le, 2013; Krizhevsky et al., 2012; Chen & Manning, 2014; Dahl et al., 2013; Hinton et al., 2012).

The theoretical understanding of learning in neural networks has lagged the practical successes. It is known that any smooth function can be approximated by a network with just one hidden layer (Barron, 1993), but training such a network is NP-hard (Blum & Rivest, 1992). In practice,

people use optimization algorithms such as stochastic gradient descent (SGD) to train neural networks. Although strong theoretical results are available for SGD in the setting of convex objective functions, there are few such results in the nonconvex setting of neural networks. While it is possible to transform the neural network training problem to a convex optimization problem involving an infinite number of variables (Bengio et al., 2005), the infinitude of variables means that there is no longer a guarantee that the learning algorithm will terminate in polynomial time.

Several recent papers have risen to the challenge of establishing polynomial-time learnability results for neural networks. These papers necessarily (given that the problem is NP-hard) introduce additional assumptions or relaxations. For instance, one may assume that the data is in fact generated by the neural network. Under this assumption, Arora et al. (2013) study the recovery of denoising auto-encoders which are represented by multi-layer neural networks. They assume that the top-layer values of the network are randomly generated and all network weights are randomly drawn from $\{-1, 1\}$. As a consequence, the bottom layer generates a sequence of random observations using which the algorithm can recover the network weights. The algorithm has polynomial-time complexity and is capable of learning random networks that are drawn from a specific distribution. However, in practice people want to learn deterministic networks that encode data-dependent representations.

Sedghi & Anandkumar (2014) study the supervised learning of neural networks under the assumption that the data distribution has a score function that is known in advance. They show that if the input dimension is large enough and the network is sparse enough, then the first network layer can be learned by a polynomial-time algorithm. Learning the deeper layers remains as an open problem. In addition, their method assumes that the network weights are randomly drawn from a Bernoulli-Gaussian distribution. More recently, Janzamin et al. (2015) propose another algorithm based on the score function that removes the restrictions of Sedghi & Anandkumar (2014). The assumption in this case

is that the network weights satisfy a non-degeneracy condition; moreover, the algorithm is only capable of learning neural networks with one hidden layer.

Another approach to the problem is via the improper learning framework. The goal in this case is to find a predictor that is not a neural network, but performs as well as the best possible neural network in terms of the generalization error. Livni et al. (2014) consider changing the activation function and over-specifying the network to make it easier to train. They show that polynomial networks (e.g., networks whose activation function is quadratic) with sufficient width and depth are as expressive as the sigmoid-activated neural networks. Although a deep polynomial network is still hard to train, they propose training in a superclass—the class of all polynomial functions with bounded degree. As a consequence, there is an improper learning algorithm which achieves a generalization error at most ϵ worse than that of the best neural network. The time complexity is polynomial in the input dimension d and quasi-polynomial in $1/\epsilon$. Since the dependence on d has a large power, the algorithm is not practical unless d is quite small. Livni et al. (2014) further show, however, that there is a practical algorithm to directly train the polynomial network if it has one or two hidden layers.

A recent line of work has focused on understanding the energy landscape of a neural network. After several simplifying assumptions, a neural network can be shown to be a Gaussian field whose critical points can be analyzed using the Kac-Rice formula and properties of the Gaussian Orthogonal Ensemble (Auffinger et al., 2013; Dauphin et al., 2014; Choromanska et al., 2014). The conclusion of these papers is that all critical points with nonnegative eigenvalues tend to have objective value near the global minimum. Thus in such networks if we could find such a point, it would have small objective value and thus small training error. This combined with generalization error bounds would imply finding a neural network with low excess risk. However, there is no provably efficient algorithm for finding a critical point with nonnegative eigenvalues.

1.1. Our contribution

In this paper, we propose a practical algorithm called the *recursive kernel method* for learning multi-layer neural networks, under the framework of improper learning. Our method is inspired by the work of Shalev-Shwartz et al. (2011), which shows that for binary classification with the sigmoidal loss, there is a kernel-based method that achieves the same generalization error as the best linear classifier. We extend this method to deeper networks. In particular, we assume that the neural network to be learned takes d-dimensional input. It has k hidden layers and the ℓ_1-norm of the incoming weights of any neuron is bounded by L.

Under these assumptions, the algorithm learns a kernel-based predictor whose generalization error is at most ϵ worse than that of the best neural network. The sample and the time complexity of the algorithm are polynomial in $(d, 1/\epsilon, \log(1/\delta), F(k, L))$, where $F(k, L)$ is a function depending on (k, L) and on the activation function, independent of the input dimension or the number of neurons. The theoretical result holds for any data distribution.

As concrete examples, we demonstrate that if the activation function is a quadratic function, then $F(k, L)$ is a polynomial function of L. Thus, the algorithm recovers the theoretical guarantee of Livni et al. (2014). We also demonstrate two activation functions, one that approximates the sigmoid function and the other that approximates the ReLU function, under which $F(k, L)$ is finite. Thus, the algorithm also learns neural networks activated by sigmoid-like or ReLU-like functions. For these latter examples, the dependence on L is no longer polynomial. This non-polynomial dependence is in fact inevitable: Under a hardness assumption in cryptographics and assuming sigmoid-like or ReLU-like activation, we prove that no algorithm running in $\text{poly}(L)$ time can improperly learn the neural network.

2. Problem Setup

We consider a fully-connected neural network $\mathcal{N}$ that maps a vector $x \in \mathbb{R}^d$ to a real number $\mathcal{N}(x)$ via k hidden layers. Let $d^{(p)}$ represent the number of neurons in the p-th layer. Let $y_i^{(p)}$ represent the output of the i-th neuron in the p-th layer. We define the zero-th layer to be the input vector so that $d^{(0)} = d$ and $y^{(0)} = x$. The transformation performed by the neural network is defined as follows:

$$y_i^{(p)} := \sigma\Big(\sum_{j=1}^{d^{(p-1)}} w_{i,j}^{(p-1)} y_j^{(p-1)} \Big) \quad \text{and}$$

$$\mathcal{N}(x) := \sum_{j=1}^{d^{(k)}} w_{1,j}^{(k)} y_j^{(k)},$$

where $w_{i,j}^{(p-1)}$ is the weight of the edge that connects the neuron j on the $(p-1)$-th layer to the neuron i on the p-th layer. The activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is a one-dimensional nonlinear function. We will discuss the choice of function σ later in this section.

We assume that the input vector has bounded ℓ_2-norm and the edge weights have bounded ℓ_1 or ℓ_2 norms. The assumptions are formalized as follows.

Assumption A. *The input vector x satisfies $\|x\|_2 \leq 1$. The*

neuron edge weights satisfy

$$\sum_{j=1}^{d}(w_{i,j}^{(0)})^2 \le L^2 \quad \forall\, i \in \{1,\dots,d\}.$$

$$\sum_{j=1}^{d^{(p)}}|w_{i,j}^{(p)}| \le L \quad \forall\,(p,i) \in \{1,\dots,k\} \times \{1,\dots,d^{(p+1)}\}.$$

Let $\mathcal{N}_{k,L,\sigma}$ be the set of k-layer neural networks with activation function σ that satisfy the edge weight constraints.

Assumption A implies that for all neurons on the first hidden layer, the ℓ_2-norm of their incoming weights is bounded by L. For other neurons, the ℓ_1-norm of their incoming weights is bounded by L. The ℓ_1-regularization imposes sparsity on the neural network. It is observed in practice that sparse neural networks are capable of learning meaningful representations. For example, the convolution neural network has sparse edges. It has been argued that sparse connectivity is a natural constraint which can lead to improved performance in practice (Thom & Palm, 2013).

In a prediction task, there is a convex function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ that measures the loss of the prediction. For a feature-label pair $(x,y) \in \mathcal{X} \times R$, its prediction loss is measured by $\ell(\mathcal{N}(x), y)$. We assume that (x,y) is sampled from an underlying distribution $\mathcal{D}$. The prediction risk of the neural network is defined by $\mathbb{E}[\ell(\mathcal{N}(x), y)]$. Our goal is to learn a predictor $f : \mathcal{X} \to \mathbb{R}$, which is not necessarily a neural network, such that

$$\mathbb{E}[\ell(f(x), y)] \le \arg\min_{\mathcal{N} \in \mathcal{N}_{k,L,\sigma}} \mathbb{E}[\ell(\mathcal{N}(x), y)] + \epsilon. \quad (1)$$

In other words, we want to learn a predictor whose generalization loss is at most ϵ worse than that of the best neural network in $\mathcal{N}_{k,L,\sigma}$.

In practice, both the sigmoid function $\sigma(x) = (1+e^{-\beta x})^{-1}$ and the ReLU function $\sigma(x) = \max(0, x)$ are widely used as activation functions for neural networks. We define two classes of activation functions that includes the sigmoid and ReLU respectively.

Definition 1 (sigmoid-like activation). *A function σ is called sigmoid-like if it is non-decreasing on $(-\infty, +\infty)$ and*

$$\lim_{x \to -\infty} x^c \sigma(x) = 0 \quad and \quad \lim_{x \to \infty} x^c(1 - \sigma(x)) = 0$$

for some positive constant c.

Definition 2 (ReLU-like activation). *A function σ is called ReLU-like if $\sigma(x) - \sigma(x-1)$ a sigmoid-like function.*

Intuitively, a sigmoid-like function is a non-decreasing function on $[0,1]$. When $x \to -\infty$ or $x \to \infty$, the function value approaches 0 or 1 at a polynomial rate (or faster) in

x. A ReLU-like function is a convex function on $[0, \infty)$. When $x \to \infty$, it approaches a linear function with unit slope.

3. Algorithm and Theoretical Result

In this section, we present a kernel method which learns a predictor performing as well as the neural network. We begin by recursively defining a sequence of kernels. Let $K : \mathbb{R}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}} \to \mathbb{R}$ be a function defined by

$$K(x,y) := \frac{1}{2 - \langle x, y \rangle},$$

where both $\|x\|_2$ and $\|y\|_2$ are assumed to be bounded by one. The function K is a kernel function because we can find a mapping $\psi : \mathbb{R}^{\mathbb{N}} \to \mathbb{R}^{\mathbb{N}}$ such that $K(x,y) = \langle \psi(x), \psi(y) \rangle$. The function ψ maps an infinite-dimensional vector to an infinite-dimensional vector. We use x_i to represent the i-th coordinate of an infinite-dimensional vector x. The $(k_1, \dots, k_j)$-th coordinate of $\psi(x)$, where $j \in \mathbb{N}$ and $k_1, \dots, k_j \in \mathbb{N}$, is defined as $2^{-\frac{j+1}{2}} x_{k_1} \dots x_{k_j}$. By this definition, we have

$$\langle \psi(x), \psi(y) \rangle =$$
$$\sum_{j=0}^{\infty} 2^{-(j+1)} \sum_{(k_1,\dots,k_j) \in \mathbb{N}^j} x_{k_1} \dots x_{k_j} y_{k_1} \dots y_{k_j}. \quad (2)$$

The inner term on the right-hand side of Eq. (2) can be simplified to

$$\sum_{(k_1,\dots,k_j) \in \mathbb{N}^j} x_{k_1} \dots x_{k_j} y_{k_1} \dots y_{k_j} = (\langle x, y \rangle)^j. \quad (3)$$

Combining Eqs. (2) and (3) and using the fact that $\langle x, y \rangle \le 1$, we have

$$\langle \psi(x), \psi(y) \rangle = \sum_{j=0}^{\infty} 2^{-(j+1)} (\langle x, y \rangle)^j$$
$$= \frac{1}{2 - \langle x, y \rangle} = K(x,y),$$

which verifies that K is a kernel function and ψ is the associated mapping. Since ψ maps from $\mathbb{R}^{\mathbb{N}}$ to $\mathbb{R}^{\mathbb{N}}$ and $\|x\|_2 \le 1$ implies $\|\psi(x)\|_2^2 = K(x,x) \le 1$, we can recursively define a sequence of mappings

$$\psi^{(0)}(x) = x \quad \text{and} \quad \psi^{(p)}(x) = \psi(\psi^{(p-1)}(x)).$$

Using the relation between K and ψ, it is easy to verify that the associated kernels are

$$K^{(0)}(x,y) = \langle x, y \rangle \quad \text{and}$$
$$K^{(p)}(x,y) = \frac{1}{2 - K^{(p-1)}(x,y)}, \quad (4)$$

which satisfy $\langle \psi^{(p)}(x), \psi^{(p)}(y) \rangle = K^{(p)}(x,y)$. Thus, the kernel function $K^{(k)}(x,y)$ can be easily computed from the inner product of x and y.

Algorithm 1 Recursive Kernel Method

Input: Feature-label pairs $\{(x_i, y_i)\}_{i=1}^n$; Loss function $\ell :$ $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$; Number of hidden layers k; Regularization coefficient B.

Solve the following convex optimization problem:

$$\widehat{\alpha} = \arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{j=1}^n \ell\left(\sum_{i=1}^n \alpha_i K^{(k)}(x_i, x_j), y_i\right)$$

$$\text{s.t.} \quad \sum_{i,j=1}^n \alpha_i \alpha_j K^{(k)}(x_i, x_j) \le B^2$$

where $K^{(k)}$ is defined in Eq. (4).
Output: Predictor $\widehat{f}_n(x) = \sum_{i=1}^n \widehat{\alpha}_i K^{(k)}(x_i, x)$.

3.1. Algorithm

We are now ready to specify the algorithm to learn the neural network. Suppose that the neural network has k hidden layers. Let $\mathcal{F}_k$ represent the Reproducing Kernel Hilbert Space (RKHS) induced by the kernel $K^{(k)}$ and let $\mathcal{F}_{k,B} \subset \mathcal{F}_k$ be the set of RKHS elements whose norm are bounded by B. Given training examples $\{(x_i, y_i)\}_{i=1}^n$, define the predictor

$$\widehat{f}_n := \arg\min_{f \in \mathcal{F}_{k,B}} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i).$$

According to the representer theorem, we can represent $\widehat{f}_n$ by

$$\widehat{f}_n(x) = \sum_{i=1}^n \alpha_i K^{(k)}(x_i, x) \quad \text{where}$$

$$\sum_{i,j=1}^n \alpha_i \alpha_j K^{(k)}(x_i, x_j) \le B^2, \quad (5)$$

Computing the vector α is a convex optimization problem in $\mathbb{R}^n$ and therefore can be solved in time $poly(n, d)$ using standard optimization tools. We call this algorithm the *recursive kernel method* and summarize it in Algorithm 1. It is an improper learning algorithm since the learned predictor $\widehat{f}_n$ cannot be represented by a neural network.

3.2. Main Result

Applying classical results from learning theory, we can upper bound the Rademacher complexity of $\mathcal{F}_{k,B}$ by $\sqrt{2B^2/n}$ (see, e.g. Kakade et al., 2009). Thus, with probability at least $1 - \delta$, we can upper bound the generalization loss of predictor $\widehat{f}_n(x)$ by

$$\mathbb{E}[\ell(\widehat{f}_n(x), y)] \le \arg\min_{f \in \mathcal{F}_{k,B}} \mathbb{E}[\ell(f(x), y)] + \epsilon,$$

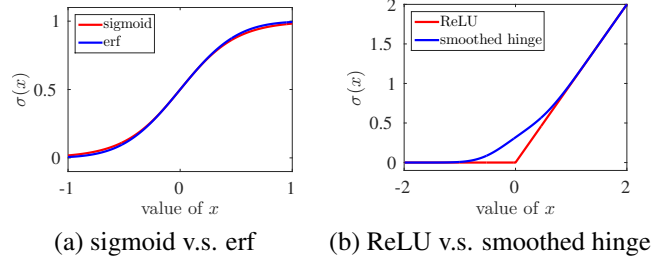(a) sigmoid v.s. erf   (b) ReLU v.s. smoothed hinge

Figure 1. Comparing different activation functions. The two functions in (a) are quite similar. The smooth hinge loss in (b) is a smoothed version of ReLU.

when the sample size $n = \Omega(B^2 \log(1/\delta)/\epsilon^2)$. See (Shalev-Shwartz et al., 2011, Theorem 2.2) for the proof of this claim. In order to establish the bound (1), it suffices to show that $\mathcal{N}_{k,L,\sigma} \subset \mathcal{F}_{k,B}$ where B is a constant that only depends on k and L. The following lemma establishes the claim. We defer the proof to Section 6.1.

Lemma 1. *Assume that the function $\sigma(x)$ has a polynomial expansion $\sigma(x) = \sum_{j=0}^\infty \beta_j x^j$. Let $H(\lambda) :=$ $L \cdot \sqrt{\sum_{j=0}^\infty 2^{j+1} \beta_j^2 \lambda^{2j}}$ and define $H^{(k)}(x)$ be the degree-k composition of function H, then $\mathcal{N}_{k,L,\sigma} \subset \mathcal{F}_{k,H^{(k)}(L)}$.*

Using Lemma 1 and the above analyses, we obtain the main result of this paper.

Theorem 1. *Let Assumption A be true and define $F(k, L) := H^{(k)}(L)$ where $H^{(k)}(L)$ is specified in Lemma 1. If $F(k, L)$ is finite, then with probability at least $1 - \delta$, the predictor defined in Algorithm 1 achieves*

$$\mathbb{E}[\ell(\widehat{f}_n(x), y)] \le \arg\min_{\mathcal{N} \in \mathcal{N}_{k,L,\sigma}} \mathbb{E}[\ell(\mathcal{N}(x), y)] + \epsilon.$$

The sample complexity is bounded by $poly(1/\epsilon, \log(1/\delta), F(k, L))$; the time complexity is bounded by $poly(d, 1/\epsilon, \log(1/\delta), F(k, L))$.

3.3. Examples

We study several concrete examples where $F(k, L)$ is finite. Our first example is the quadratic activation function:

$$\sigma_{\text{sq}}(x) = x^2.$$

This activation function has been studied by Livni et al. (Livni et al., 2014), who refer to a neural network activated by this function as a polynomial network. In Theorem 1, if the quadratic activation function is employed, we have $H(\lambda) = 2L\lambda^2$. As a consequence, we have $F(1, L) = 2L^2$ and more generally $F(k, L) \le (2L)^{2^{k+1}-1}$ by induction. Thus, the sample and the time complexity of Algorithm 1 is a polynomial function of $(d, 1/\epsilon, \log(1/\delta), L)$ for any constant k.

Next, we study sigmoid-like or ReLU-like activation functions. We consider a shifted erf function defined as:

$$\sigma_{\mathrm{erf}}(x) = \frac{1}{2}(1 + \mathrm{erf}(\sqrt{\pi}x)),$$

and a smoothed hinge loss function defined as:

$$\sigma_{\mathrm{sh}}(x) = \int_{-\infty}^{x} \sigma_{\mathrm{erf}}(t)dt = \sigma_{\mathrm{erf}}(x) \cdot x + \frac{e^{-\pi x^2}}{2\pi}.$$

In Figure 1, we compare σ_{erf} and σ_{sh} with the sigmoid function and the ReLU function. It is seen that σ_{erf} is similar to the sigmoid function and σ_{sh} is a smoothed version of ReLU. It is also easy to verify that σ_{erf} is sigmoid-like and σ_{sh} is ReLU-like. The following proposition shows that if either σ_{erf} or σ_{sh} is used as the activation function, the quantity $F(k, L)$ is finite. See Appendix 6.2 for the proof.

Proposition 1. *For the σ_{erf} function, we have*

$$H(\lambda) \leq L \cdot \sqrt{\frac{1}{2} + 4\lambda^2(1 + 3e\pi\lambda^2 e^{4\pi\lambda^2})} \quad \text{for any } \lambda \geq 3.$$

For the σ_{sh} function, we have

$$H(\lambda) \leq L \cdot \sqrt{\lambda^2 + 8\lambda^4(1 + 3e\pi\lambda^2 e^{4\pi\lambda^2})} \quad \text{for any } \lambda \geq 3.$$

Thus, Theorem 1 implies that the neural network activated by σ_{erf} or σ_{sh} is learnable in polynomial time given any constant (k, L).

Finally, we demonstrate how the conditions of Assumption A could be modified. Consider a sigmoid-activated network with k hidden layers which satisfies the following:

$$\sum_{j=1}^{d^{(p)}} |w_{i,j}^{(p)}| \leq L \quad \forall \, (p, i) \in \{1, \dots, k\} \times \{0, \dots, d^{(p+1)}\}.$$

This means that the ℓ_1-norm of all layers is bounded by L. In addition, we assume that the input vector satisfies $\|x\|_\infty \leq 1$. This is in contrast to the condition $\|x\|_2 \leq 1$ in Assumption A. It was shown by (Livni et al., 2014, Theorem 4) that this sigmoid network can be approximated by a polynomial network with arbitrarily small approximation error ϵ. The associated polynomial network has $\mathcal{O}(k \log(Lk + L \log(1/\epsilon)))$ hidden layers, whose ℓ_1-norms are bounded by $e^{\mathcal{O}(L \log(1/\epsilon))}$. If we normalize the input vector $x \in \mathbb{R}^d$ by $x \leftarrow x/\sqrt{d}$ and multiple all first-layer weights by $\sqrt{d}$, the output of the network remains invariant and it satisfies Assumption A. Thus, combining our result for the polynomial network and the above analysis, the sigmoid network can be learned in

$$\text{poly}\left(d^{(Lk + L \log(1/\epsilon))^{\mathcal{O}(k)}}, \log(1/\delta)\right)$$

sample and time complexity. This is a quasi-polynomial dependence on $1/\epsilon$ for any constant (k, L). Notice that the dimension d comes into the expression.

4. Hardness Result

In Section 3.3, we see that the dependence of the time complexity on L is at least exponential for σ_{erf} and σ_{sh}, but it is polynomial for the quadratic activation. It is thus natural to wonder if there is a sigmoid-like or ReLU-like activation function that makes the time complexity a polynomial function of L. In this section, we prove that this is impossible given standard hardness assumptions.

Our proof relies on the hardness of standard (nonagnostic) PAC learning of intersection of halfspaces given in Klivans and Sherstov (Klivans et al., 2006). More precisely, let

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

be the family of halfspace indicator functions mapping $\mathcal{X} = \{-1, 1\}^d$ to $\{-1, 1\}$, and let H_T be the set of functions taking the form:

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

Thus, H_T is the set of functions that indicates the intersection of T halfspaces. For any distribution on $\mathcal{X}$, an algorithm $\mathcal{A}$ takes a sequence of $(x, h^*(x))$ as input where x is a sample from $\mathcal{X}$ and $h^* \in H_T$. The algorithm learns a function $\widehat{h}$ such that with probability at least $1 - \delta$, one has

$$P(\widehat{h}(x) \neq h^*(x)) \leq \epsilon. \tag{6}$$

If there is such an algorithm $\mathcal{A}$ whose sample complexity and time complexity scale as $\text{poly}(d)$, then we say that H_T is efficiently learnable. Klivans and Sherstov (Klivans et al., 2006) show that H_T is not efficiently learnable under a certain cryptographic assumption.

Theorem 2 (Klivans and Sherstov (Klivans et al., 2006))**.** *If $T = d^\rho$ for some constant $\rho > 0$, then under a certain cryptographic assumption, H_T is not efficiently learnable.*

We use this hardness result to prove the hardness of learning neural networks. In particular, we construct a neural network $\mathcal{N}$ such that if there is a learning algorithm computing a predictor $\widehat{f}$ such that $\mathbb{E}[\ell(\widehat{f}(x), y)] \leq \mathbb{E}[\ell(\mathcal{N}(x), y)] + \epsilon$, then the error bound (6) is satisfied. Thus, the hardness of learning intersection of halfspaces implies the hardness of learning neural networks. See Section 6.3 for the proof.

Theorem 3. *Assume the cryptographic assumption of Theorem 2. Let σ be a sigmoid-like or ReLU-like function and let $\ell(f(x), y) = \max(0, 1 - yf(x))$ be the hinge loss. For fixed (δ, ϵ), there is no algorithm running in $\text{poly}(L)$ time*

that learns a predictor $\widehat{f}$ satisfying

$$\mathbb{E}[\ell(\widehat{f}(x), y)] \leq \arg\min_{\mathcal{N} \in \mathcal{N}_{1,L,\sigma}} \mathbb{E}[\ell(\mathcal{N}(x), y)] + \epsilon$$

$$\text{with probability at least } 1 - \delta. \quad (7)$$

The hardness of learning sigmoid-activated and ReLU-activated neural networks has been proved by Livni et al. (Livni et al., 2014) when ℓ is the zero-one loss. Theorem 3 presents a more general result, showing that any activation function that is sigmoid-like or ReLU-like leads to the computational hardness, even if the loss function ℓ is convex.

5. Experiments

In this section, we compare the proposed algorithm with several baseline algorithms on the MNIST digit recognition task. Since the basic MNIST digits are relatively easy to classify, we introduce three variations which make the problem more challenging.

We use the MNIST handwritten digits dataset and three variations of it. See Figure 2 for the description of these datasets and several exemplary images. All the images are of size 28×28. For all datasets, we use 10,000 images for training, 2,000 images for validation and 50,000 images for testing. This partitioning is recommended by the source of the data (Win).

For the recursive kernel method, we train one-vs-all SVM classifiers with Algorithm 1. The hyper-parameters are given by $k \in \{1, 4\}$ and $B = 100$. All images are pre-processed by the following steps: deskewing, centering and normalization. The deskewing step computes the principal axis of the shape that is closest to the vertical, and shifts the lines so as to make it vertical. It is a common preprocessing step for the kernel method (LeCun et al., 1998). The centering and normalization steps center the feature vector and scale it to have the unit ℓ_2-norm.

We compare with the following baseline models: multi-class logistic regression, multi-layer perceptron and convolution neural networks. The multi-layer perceptron is a fully connected neural network with a single hidden layer which contains 500 hidden neurons. It covers the networks that can be learned by the method of Janzamin et al. (Janzamin et al., 2015). The convolution neural networks implement the LeNet5 architecture (LeCun et al., 1998). All baseline models are trained via stochastic gradient descent.

The classification error rates are summarized in Table 1. As the table shows, the recursive kernel method is consistently more accurate than logistic regression and the multi-layer perceptron. On the Basic and the Rotation datasets, the proposed algorithm is comparable with LeNet5. On the other two datasets, LeNet5 wins over other methods by a relatively large margin. It is worth noting that when we choose a greater k, the performance of the proposed algorithm gets better. Recall that a greater k learns a deeper neural network, thus the empirical observation is intuitive.

Although the recursive kernel method doesn't outperform the LeNet5 model, the experiment demonstrates that it does learn better predictors than fully connected neural networks such as the multi-layer perceptron. The LeNet5 architecture encodes prior knowledge about digit recogniition via the convolution and pooling operations; thus its performance is better than the generic architectures.

6. Proofs

In this section, we present proofs to the technical results in this paper.

6.1. Proof of Lemma 1

Consider an arbitrary neural network $\mathcal{N} \in \mathcal{N}_{k,L,\sigma}$. Let $g_i^{(p)} := \sum_{j=1}^{d^{(p)}} w_{ji}^{(p)} y_j^{(p)}$ represent the input of the neuron i at layer $p+1$. Note that $g_i^{(p)}$ is a function of the input vector x. By this definition, it suffices to show that $g_1^{(k)} \in \mathcal{F}_{k,H^{(k)}(L)}$.

We claim that $g_i^{(p)} \in \mathcal{F}_{p,H^{(p)}(L)}$ for any $p \in \{0, 1, \ldots, k\}$ and prove the claim by induction. For $p = 0$, we have

$$g_i^{(0)}(x) = \sum_{j=1}^{d} w_{i,j}^{(0)} x_j = \langle w_i^{(0)}, \psi^{(0)}(x) \rangle.$$

Thus, $g_i^{(0)}$ belongs to the RKHS induced by the kernel $K^{(0)}$. Furthermore, we have $\|g_i^{(0)}\|_{\mathcal{F}_0} = \|w_i^{(0)}\|_2 \leq L = H^{(0)}(L)$, which implies $g_i^{(0)} \in \mathcal{F}_{0,H^{(0)}(L)}$.

For $p > 0$, we assume that the claim holds for $p-1$ and we will prove it for p. The definition of $g_i^{(p)}$ implies

$$g_i^{(p)}(x) = \sum_{j=1}^{d^{(p)}} w_{ji}^{(p)} \sigma\left(g_j^{(p-1)}(x)\right).$$

Using the inductive hypothesis, we have $g_j^{(p-1)} \in \mathcal{F}_{p-1,H^{(p-1)}(L)}$, which implies that $g_j^{(p-1)}(x) = \langle v_j, \psi^{(p-1)}(x) \rangle$ for some $v_j \in \mathbb{R}^{\mathbb{N}}$, and $\|v_j\|_2 \leq H^{(p-1)}(L)$. This implies

$$g_i^{(p)}(x) = \sum_{j=1}^{d^{(p)}} w_{i,j}^{(p)} \sigma(\langle v_j, \psi^{(p-1)}(x) \rangle). \quad (8)$$

Let $x^{(p-1)}$ be a shorthand notation of $\psi^{(p-1)}(x)$. We define vector $u_j \in \mathbb{R}^{\mathbb{N}}$ as follow: the $(k_1, \ldots, k_t)$-th coordinate of u_j, where $t \in \mathbb{N}$ and $k_1, \ldots, k_t \in \mathbb{N}^+$, is equal to

(a) Basic

(b) Rotation

(c) Background

(d) Background + Rotation

Figure 2. The MNIST dataset and its variations. (a) the basic MNIST dataset; (b) the digits were rotated by an angle generated uniformly between 0 and 2π. (c) a black and white image was used as the background for the digit image; (d) the background perturbation and the rotation perturbation are combined.

	Basic	Rotation	Background	Background+Rotation
Logistic Regression	9.53%	46.01%	28.05%	66.93%
Multilayer Perceptron	4.98%	14.72%	28.68%	63.91%
LeNet5	**2.08%**	9.27%	**9.35%**	**32.36%**
Recursive Kernel ($k = 1$)	3.31%	9.71%	22.39%	53.72%
Recursive Kernel ($k = 4$)	3.08%	**8.78%**	22.13%	52.94%

Table 1. Classification error rates of different methods on the MNIST dataset and its variations. The best results are marked by the bold font.

$2^{\frac{t+1}{2}}\beta_t v_{j,k_1}\ldots v_{j,k_t}$. By this definition, we have

$$\sigma(\langle v_j, x^{(p-1)}\rangle) = \sum_{t=0}^{\infty}\beta_t(\langle v_j, x^{(p-1)}\rangle)^t$$

$$= \sum_{t=0}^{\infty}\beta_t \sum_{(k_1,\ldots,k_t)\in\mathbb{N}^t} v_{j,k_1}\ldots v_{j,k_t}x_{k_1}^{(p-1)}\ldots x_{k_t}^{(p-1)}$$

$$= \langle u_j, \psi(x^{(p-1)})\rangle, \tag{9}$$

where the first equation holds since $\sigma(x)$ has a polynomial expansion $\sigma(x) = \sum_{t=0}^{\infty}\beta_t x^t$, the second by expanding the inner product, and the third by definition of $\psi(x)$. Combining Eq. (8) and Eq. (9), we have

$$g_i^{(p)}(x) = \sum_{j=1}^{d^{(p)}} w_{i,j}^{(p)}\langle u_j, \psi(\psi^{(p-1)}(x))\rangle$$

$$= \left\langle \sum_{j=1}^{d^{(p)}} w_{ji}^{(p)} u_j, \psi^{(p)}(x)\right\rangle.$$

This implies that $g_i^{(p)}$ belongs to the RKHS induced by the kernel $K^{(p)}$.

Finally, we upper bound the norm of $g_i^{(p)}$. Notice that

$$\|g_i^{(p)}\|_{\mathcal{F}_p} = \left\|\sum_{j=1}^{d^{(p)}} w_{i,j}^{(p)} u_j\right\|_2 \tag{10}$$

$$\leq \sum_{i=1}^{d^{(p)}}|w_{i,j}^{(p)}|\cdot\|u_j\|_2 \leq L\cdot\max_{j\in[d^{(p)}]}\{\|u_j\|_2\}.$$

Using the definition of u_j and the inductive hypothesis, we

have

$$\|u_j\|_2^2 = \sum_{t=0}^{\infty}2^{t+1}\beta_t^2 \sum_{(k_1,\ldots,k_t)\in\mathbb{N}^t} v_{j,k_1}^2 v_{j,k_2}^2\cdots v_{j,k_t}^2 \tag{11}$$

$$= \sum_{t=0}^{\infty}2^{t+1}\beta_t^2\|v_j\|_2^{2t} \leq \sum_{t=0}^{\infty}2^{t+1}\beta_t^2(H^{(p-1)}(L))^{2t}.$$

Combining inequality (10) and (11), we have $\|g_i^{(p)}\|_{\mathcal{F}_p} \leq H^{(p)}(L)$, which verifies that $g_i^{(p)} \in \mathcal{F}_{p,H^{(p)}(L)}$.

6.2. Proof of Proposition 1

For the σ_{erf} function, the polynomial expansion is

$$\sigma_{\mathrm{erf}}(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}}\sum_{j=0}^{\infty}\frac{(-1)^j(\sqrt{\pi}x)^{2j+1}}{j!(2j+1)}.$$

Therefore, we have

$$H(\lambda) = L\cdot\sqrt{\frac{1}{2} + \frac{2}{\pi}\sum_{j=0}^{\infty}\frac{(2\pi\lambda^2)^{2j+1}}{(j!)^2(2j+1)^2}}. \tag{12}$$

(Shalev-Shwartz et al., 2011, Corollary C) provides an upper bound on the right-hand side of Eq. (12). In particular, they prove that for any $\lambda \geq 3$:

$$\frac{2}{\pi}\sum_{j=0}^{\infty}\frac{(2\pi\lambda^2)^{2j+1}}{(j!)^2(2j+1)^2} \leq 4\lambda^2(1 + 3e\pi\lambda^2 e^{4\pi\lambda^2}). \tag{13}$$

Plugging this upper bound to Eq. (12) completes the proof.

For the σ_{sh} function, since it is the integral of the σ_{erf} func-

tion, its polynomial expansion is

$$\sigma_{\mathrm{sh}}(x) = \frac{x}{2} + \frac{1}{\sqrt{\pi}} \sum_{j=0}^{\infty} \frac{(-1)^j (\sqrt{\pi}x)^{2j+1} x}{j!(2j+1)(2j+2)},$$

and consequently,

$$H(\lambda) = L\sqrt{\lambda^2 + \frac{2}{\pi}\sum_{j=0}^{\infty}\frac{(2\pi\lambda^2)^{2j+1}(2\lambda^2)}{(j!)^2(2j+1)^2(2j+2)^2}}. \quad (14)$$

We upper bound the right-hand side of Eq. (14) by

$$\frac{2}{\pi}\sum_{j=0}^{\infty}\frac{(2\pi\lambda^2)^{2j+1}(2\lambda^2)}{(j!)^2(2j+1)^2(2j+2)^2} \le \frac{4\lambda^2}{\pi}\sum_{j=0}^{\infty}\frac{(2\pi\lambda^2)^{2j+1}}{(j!)^2(2j+1)^2}$$

$$\le 8\lambda^4(1 + 3e\pi\lambda^2 e^{4\pi\lambda^2}) \quad \forall\, \lambda \ge 3,$$

where the final inequality holds because of Eq. (13). Plugging this upper bound into Eq. (14) completes the proof.

6.3. Proof of Theorem 3

We construct a one-hidden-layer neural network that encodes the intersection of T halfspaces. Suppose that the t-th halfspace is characterized by $g_t(x) = w_t^T x - b_t - 1/2$. Since both x, w_t and b_t are composed of integers, we have $g_t(x) \ge 1/2$ when $h_t(x) = 1$, and $g_t(x) \le -1/2$ when $h_t(x) = -1$. We extend x to be $(x, 1)$, then extend w_t to be (w_t, b_t), and define

$$\widetilde{g}_t(x) = \langle \widetilde{w}_t, \widetilde{x} \rangle \quad \text{where}$$

$$\widetilde{x} := \frac{1}{\sqrt{d+1}}(x, 1) \quad \text{and} \quad \widetilde{w} := 2\lambda\sqrt{d+1}(w_t, b_t),$$

where λ is a scalar to be specified. According to this definition, we have $\|\widetilde{x}\|_2 = 1$ and $\|\widetilde{w}\|_2 = \mathrm{poly}(d)$. In addition, we have $\widetilde{g}_t(x) \ge \lambda$ when $h_t(x) = 1$, and $\widetilde{g}_t(x) \le -\lambda$ when $h_t(x) = -1$.

Sigmoid-like Activation If σ a is sigmoid-like function, there is a constant c such that

$$\lim_{x \to -\infty} x^c \sigma(x) = \lim_{x \to \infty} x^c(1 - \sigma(x)) = 0.$$

Thus, there is a sufficiently large constant C such that $\sigma(x) \le x^{-c}$ for all $x \le -C$ and $\sigma(x) \ge 1 - x^{-c}$ for all $x \ge C$. Note that the number T of intersecting halfspaces is a polynomial function of dimension d. As a consequence, there is a sufficiently large constant $\lambda \sim \mathrm{poly}(d)$ such that

$$\forall\, x > \lambda,\ \sigma(x) \ge 1 - \frac{1}{4T} \quad \text{and} \quad \forall\, x \le -\lambda,\ \sigma(x) \le \frac{1}{4T}.$$

Thus, we have $\sigma(\widetilde{g}_t(x)) \ge 1 - \frac{1}{4T}$ if $h_t(x) = 1$ and $\sigma(\widetilde{g}_t(x)) \le \frac{1}{4T}$ if $h_t(x) = -1$.

We define the neural network $\mathcal{N}$ to be

$$\mathcal{N}(x) = \sum_{t=1}^{T} 4\, \sigma(\widetilde{g}_t(x)) - (4T - 2). \quad (15)$$

It is easy to verify that $\mathcal{N} \in \mathcal{N}_{1,L,\sigma}$ for some $L \sim \mathrm{poly}(d)$. If $h^*(x) = 1$, then x belongs to the intersection of halfspaces. It implies that $\sigma(\widetilde{g}_t(x)) \ge 1 - \frac{1}{4T}$ for all $t \in [T]$. Combining with Eq. (15), we obtain $\mathcal{N}(x) \ge 1$. On the other hand, if $h^*(x) = -1$, then there is some t such that $\sigma(\widetilde{g}_t(x)) \le \frac{1}{4T}$. Thus, Eq. (15) implies $\mathcal{N}(x) \le -1$. In summary, we have $h^*(x)\mathcal{N}(x) \ge 1$ for any $x \in \mathcal{X}$. As a consequence, we have $\ell(\mathcal{N}(x), h^*(x)) \equiv 0$ where ℓ is the hinge loss.

Assume that there is a predictor $\widehat{f}$ satisfying the error bound (7). Let $\widehat{h}(x) = \mathrm{sign}(\widehat{f}(x))$ be a classifier that judges the intersection of hyperplanes. Since the hinge loss is an upper bound on the zero-one loss, we have

$$P(\widehat{h}(x) \ne h^*(x)) = \mathbb{E}[\mathbb{I}(\widehat{h}(x) \ne h^*(x))]$$
$$= \mathbb{E}[\mathbb{I}(\mathrm{sign}(\widehat{f}(x)) \ne h^*(x))] \le \mathbb{E}[\ell(\widehat{f}(x), h^*(x))]$$
$$\le \mathbb{E}[\ell(\mathcal{N}(x), h^*(x))] + \epsilon = \epsilon,$$

where the final inequality follows from inequality (7). The last equation holds since $\ell(\mathcal{N}(x), h^*(x)) \equiv 0$. This implies that the associated classifier $\widehat{h}$ satisfies the error bound (6). Since $\widehat{h}$ cannot be computed in $\mathrm{poly}(d)$ time, we conclude that $\widehat{f}$ cannot be computed in $\mathrm{poly}(L)$ time.

ReLU-like Activation If σ is a ReLU-like function, then by definition, we have $\sigma'(x) := \sigma(x) - \sigma(x - 1)$ is a sigmoid-like function. Following the argument for the sigmoid-like activation, if we treat σ' as the activation function, then the remaining part of the proof will go through without any further modification. This completes the proof for the ReLU-like activation.

7. Discussion

There is an increasing interest on understanding the connection between the kernel method and the deep neural networks. Cho & Saul (2009) propose an arc-cosine kernel that mimics the computation of neural networks. Mairal et al. (2014) use the kernel as a tool to train convolutional neural networks. A key ingredient of our analysis is to approximate the neural network by a polynomial expansion. Andoni et al. (2014) study the reverse problem: they show that a polynomial function can be learned via gradient descent on neural networks.

We view this line of work as a contribution to the ongoing effort to develop learning algorithms for neural networks that are both understandable in theory and useful in practice. The theoretical results in this paper are likely to be conservative. Theorem 3 shows that the exponential dependence on the norm L is unavoidable under the worst-case data distribution. The real data, however, are rarely generated from the worst-case distribution. An intriguing open problem is to exploit the property of the data distribution to achieve a polynomial dependence on the norm L.

References

Variations on the MNIST digits. `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations`.

Andoni, Alexandr, Panigrahy, Rina, Valiant, Gregory, and Zhang, Li. Learning polynomials with neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1908–1916, 2014.

Arora, Sanjeev, Bhaskara, Aditya, Ge, Rong, and Ma, Tengyu. Provable bounds for learning some deep representations. *ArXiv:1310.6343*, 2013.

Auffinger, Antonio, Arous, Gérard Ben, and Černỳ, Jiří. Random matrices and complexity of spin glasses. *Communications on Pure and Applied Mathematics*, 66(2): 165–201, 2013.

Barron, Andrew R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

Bengio, Yoshua, Roux, Nicolas L, Vincent, Pascal, Delalleau, Olivier, and Marcotte, Patrice. Convex neural networks. In *Advances in Neural Information Processing Systems*, pp. 123–130, 2005.

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

Chen, Danqi and Manning, Christopher D. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pp. 740–750, 2014.

Cho, Youngmin and Saul, Lawrence K. Kernel methods for deep learning. In *Advances in neural information processing systems*, pp. 342–350, 2009.

Choromanska, Anna, Henaff, Mikael, Mathieu, Michael, Arous, Gérard Ben, and LeCun, Yann. The loss surface of multilayer networks. *ArXiv:1412.0233*, 2014.

Dahl, George E, Sainath, Tara N, and Hinton, Geoffrey E. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8609–8613. IEEE, 2013.

Dauphin, Yann N, Pascanu, Razvan, Gulcehre, Caglar, Cho, Kyunghyun, Ganguli, Surya, and Bengio, Yoshua. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pp. 2933–2941, 2014.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29 (6):82–97, 2012.

Janzamin, Majid, Sedghi, Hanie, and Anandkumar, Anima. Generalization bounds for neural networks through tensor factorization. *ArXiv:1506.08473*, 2015.

Kakade, Sham M, Sridharan, Karthik, and Tewari, Ambuj. On the complexity of linear prediction: Risk bounds, margin bounds, and regularization. In *Advances in neural information processing systems*, pp. 793–800, 2009.

Klivans, Adam R, Sherstov, Alexander, et al. Cryptographic hardness for learning intersections of halfspaces. In *47th Annual IEEE Symposium on Foundations of Computer Science*, pp. 553–562. IEEE, 2006.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Le, Quoc V. Building high-level features using large scale unsupervised learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8595–8598. IEEE, 2013.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Livni, Roi, Shalev-Shwartz, Shai, and Shamir, Ohad. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pp. 855–863, 2014.

Mairal, Julien, Koniusz, Piotr, Harchaoui, Zaid, and Schmid, Cordelia. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pp. 2627–2635, 2014.

Sedghi, Hanie and Anandkumar, Anima. Provable methods for training neural networks with sparse connectivity. *ArXiv:1412.2693*, 2014.

Shalev-Shwartz, Shai, Shamir, Ohad, and Sridharan, Karthik. Learning kernel-based halfspaces with the 0-1 loss. *SIAM Journal on Computing*, 40(6):1623–1646, 2011.

Thom, Markus and Palm, Günther. Sparse activity and sparse connectivity in supervised learning. *Journal of Machine Learning Research*, 14(1):1091–1143, 2013.