Learning Polynomials with Neural Networks

Alexandr Andoni ANDONI@MICROSOFT.COM
Microsoft Research

Rina Panigrahy RINA@MICROSOFT.COM
Microsoft Research

Gregory Valiant GREGORY.VALIANT@GMAIL.COM
Stanford University

Li Zhang LZHA@MICROSOFT.COM
Microsoft Research

Abstract

We study the effectiveness of learning low degree polynomials using neural networks by the gradient descent method. While neural networks have been shown to have great expressive power, and gradient descent has been widely used in practice for learning neural networks, few theoretical guarantees are known for such methods. In particular, it is well known that gradient descent can get stuck at local minima, even for simple classes of target functions. In this paper, we present several positive theoretical results to support the effectiveness of neural networks. We focus on two-layer neural networks where the bottom layer is a set of non-linear hidden nodes, and the top layer node is a linear function, similar to Barron (1993). First we show that for a randomly initialized neural network with sufficiently many hidden units, the generic gradient descent algorithm learns any low degree polynomial, assuming we initialize the weights randomly. Secondly, we show that if we use complex-valued weights (the target function can still be real), then under suitable conditions, there are no "robust local minima": the neural network can always escape a local minimum by performing a random perturbation. This property does not hold for real-valued weights. Thirdly, we discuss whether sparse polynomials can be learned with *small* neural networks, with the size dependent on the sparsity of the target function.

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

1. Introduction

Neural networks have drawn signification attention from the machine learning community, in part due to the recent empirical successes (see the surveys Bengio (2009; 2013)). Neural networks have led to state-of-the-art systems for crucial applications such as image recognition, speech recognition, natural language processing, and others; see, e.g., (Krizhevsky et al., 2012; Goodfellow et al., 2013; Wan et al., 2013). There are several key concepts that have been instrumental in the success of learning the neural networks, including gradient descent, parallel implementations, carefully-designed network structure (e.g., convolutional networks), unsupervised pre-training (e.g., auto-encoders, RBMs) (Hinton et al., 2006; Ranzato et al., 2006; Bengio et al., 2007), among others.

With the flurry of empirical successes and anecdotal evidence suggesting (sometimes conflicting) principles in the design and training of neural networks, there seems to be a need for a more solid theoretical understanding of neural networks. Perhaps the most natural starting point for such investigations is the question of when a neural network *provably* learns a given target function. A classical result is that, if the target function is linear, the perceptron algorithm will learn it. This is equivalent to saying that gradient descent on a neural network with a single node (or layer) can successfully learn a linear function.

But what can we prove about gradient descent on networks with more than one layer? In full generality, this question is nearly hopeless: standard complexity-theoretic results strongly suggest there are no efficient algorithms for learning arbitrary functions (let alone gradient descent), even for target functions representable by very low-depth networks (circuits) (Applebaum et al., 2006). Nevertheless, we may hope to prove concrete results in restricted settings whose

structure is reminiscent of the structure present in practical settings.

In this work, we consider learning bounded degree polynomials by neural networks. Polynomials are an expressive class of functions since they can be used to approximate many "reasonable" functions, for example, any Lipshitz function can be well-approximated by a bounded degree polynomial (see, e.g., Andoni et al. (2014)). We establish provable guarantees for gradient descent on two-layer neural networks (i.e., networks with one hidden layer of neurons) for learning bounded degree polynomials. Our main result shows this setting can learn any bounded degree polynomial, provided the neural network is sufficiently large. Specifically, suppose the target function f is a degree d polynomial over an n-dimensional variable $x \in \mathbb{C}^n$, with x distributed according to a product distribution (for our main results, the distribution can be Gaussian or uniform distributions over the reals). The two-layer network with m hidden units outputs a function $g(x) = \sum_{i=1}^{m} \alpha_i \phi(w_i \cdot x)$, where $\alpha_i \in \mathbb{C}, w_i \in \mathbb{C}^n$ are network parameters, and ϕ is a non-linear activation function (e.g., a sigmoid). It has been known (Barron, 1993) that, when $m \approx n^d$, there exists some choice of parameters w_i, α_i so that the network g closely approximates the function f. We show that, in fact, with high probability, even if the bottom layer (w_i's) is set to be *random*, there is a choice for top layer (α_i's) such that the neural network approximates the target function f. Hence, the perceptron algorithm, when run on *only the top layer* while keeping the bottom layer fixed, will learn the correct weights α_i.

Learning polynomials. Analyzing gradient descent on the entire network turns out to be more challenging as, unlike when the bottom layer is fixed, the error function is no longer convex. We show that even if gradient descent is run on the entire network, i.e. on both top and bottom layers, the neural network will still find the network parameters α_i and w_i, for which the network approximates the target function f. This can be interpreted as saying that the effect of learning the bottom layer does not negatively affect the overall learning of the target function. Indeed, we show that the learning of the top layer (α_i's) happens sufficiently fast, in particular before the bottom layer (w_i's) significantly changes. This insight may be useful in analyzing gradient descent for the neural networks with multiple layers.

We further explore the landscape of the gradient descent by showing a conceptually compelling, though incomparable result: we show that for sufficiently large networks, in a large region of the parameter space, there are *no* robust local optima. That is, for *any* point in the parameter space that satisfies some mild conditions on the weights, with constant probability, a random perturbation will reduce the error by a non-negligible factor. Hence even when the gradient descent is stuck at a local minimal, a random perturbation would take it out. Because of the conditions on the weights of the neural network, we can not use this theorem to conclude that gradient descent plus random perturbation will always converge to the global optima from any initial neural network initialization; nevertheless, this provides a rigorous, and conceptually compelling explanation for why the existence of local optima do not deter the usage of neural networks in practice.

We stress that the random perturbation on the weights is *complex-valued* but the input can still be real valued. In contrast, the same "robustness" result does *not* hold when the random perturbation is strictly real; in this case there are robust local minima such that a random perturbation will increase the error. Intuitively, the complex-valued weights give extra dimensions, and hence nicer geometry, to the local structure of the space of neural networks, which allows it to escape from a local minimum. We find this result intriguing and hope it can further motivate the construction of new types of neural networks.

Learning *sparse* polynomials. It is natural to ask whether, in the case of a target function that is "simpler" (e.g., because it can be represented by a small number of hidden units), the gradient descent can actually learn it *more efficiently*. "More efficiently" here can mean both: 1) with a smaller network, 2) at a faster convergence rate. To study this general question, we suggest the following concrete open problem. Define the sparsity of a polynomial as the number of its nonzero monomial terms. We know, thanks to Barron (1993), a k-sparse degree-d polynomial can be *represented* by a neural network with $nk \cdot d^{O(d)}$ hidden nodes, but can gradient descent learn such a polynomial by a neural network with only $(nk \cdot d^d)^{O(1)}$ hidden nodes, and in a similar amount of time?

We note here we require the target function to be a k-sparse degree-d polynomial, more restrictive than requiring the target to be representable by a small network as in Barron (1994). This might be an easier problem, especially given that recent work has shown that one can indeed learn, for Gaussian or uniform distribution, any sparse polynomial in $nk \cdot d^{O(d)}$ time (Andoni et al., 2014), albeit via an algorithm that does not resemble gradient descent. We should emphasize that it is important to consider well-behaved distribution such as Gaussian or uniform distributions. Otherwise, if we allow arbitrary distribution, the sparsity assumption may not be helpful. For example, in the boolean case (where x is distributed over the boolean cube), this is at least as hard as "learning parity with noise" and "learning juntas" – problems that we suspect do not admit any efficient algorithms and are even used as cryptographic primitives (Alekhnovich, 2003; Regev, 2005; Peikert, 2009; Ap-

plebaum et al., 2010).

While we do not have the answer to this natural but challenging question, we give some indications why this may indeed be possible. In particular we show that, if the target function depends only on $k \ll n$ variables, then the neural network will learn a function that also depends on these k variables. Additionally, we provide some strong empirical evidence that such small networks are capable of learning sparse polynomials.

With the recent success of neural networks in practice, there is an increasing demand of insights and tools for analyzing gradient descent for learning neural networks. Our work represents one such effort. We note that Saxe et al. (2013) is another recent work in this space.

We will only sketch proof ideas in this abstract. The full details can be found in the supplementary material.

2. Preliminaries

We will mostly use complex-valued inputs and network parameters, to simplify the description. Most results generalize to other distributions.

For a complex number $c \in \mathbb{C}$, write $\bar{c}$ as its conjugate, and define its norm as $|c| = \sqrt{c\bar{c}}$. For a matrix (or a vector) P, write P^* as the adjoint of P, i.e., $(P^*)_{ij} = \overline{P_{ji}}$.

We denote by $\mathbb{C}(r)$ the uniform distribution of complex number with norm exactly r, $\mathbf{N}(\sigma^2)$ the real Gaussian distribution with mean 0 and variance σ^2, and $\mathbf{U}(r)$ the uniform distribution on $[-r, r]$. For a distribution $\mathbf{P}$, denote by $\mathbf{P}^n$ the n-fold product distribution. For any given distribution D and two functions $f, g : \mathbb{C}^n \mapsto \mathbb{C}$, define the inner product $\langle f, g\rangle_D = \mathrm{E}_{x\sim D}[f(x)\overline{g(x)}]$ and the norm $\|f\|_D = \sqrt{\langle f, f\rangle_D} = \sqrt{\mathrm{E}_{x\sim D}|f(x)|^2}$.

Polynomials. For any $J = (J_1, \cdots, J_n) \in \mathbb{N}^n$, write a monomial $x^J = x_1^{J_1} \cdots x_n^{J_n}$. Define $|J| = \sum_k J_k$. For a polynomial $p(x) = \sum_J b_J x^J$ where $b_J \in \mathbb{C}$, its degree $\deg(p) \triangleq \max_{b_J \neq 0} |J|$. All the degree d polynomials form an n^d-dimensional linear space. For any given distribution D, a polynomial basis $P_1, P_2, \ldots$ is orthonormal w.r.t. D if $\langle P_i, P_j\rangle_D = 1$ if $i = j$ and is zero otherwise.

We will use the following basic but important fact that monomials form an orthonormal basis for $D = \mathbb{C}(1)^n$.

Fact 2.1. *$\langle w^J, w^{J'}\rangle_{\mathbb{C}(r)^n} = r^{2|J|}$ if $J = J'$ and 0 otherwise.*

For Gaussian and uniform distributions, the corresponding orthonormal bases are the Hermite and Legendre polynomials respectively.

Neural networks. Assume $\phi : \mathbb{C} \mapsto \mathbb{C}$ is an analytic function without poles. Write $\phi(z) = \sum_{j\geq 0} a_j z^j$. For any $w \in \mathbb{C}^n$, define $\phi^w : \mathbb{C}^n \mapsto \mathbb{C}$ as

$$\phi^w(x) = \phi\left(\sum_k w_k x_k\right) = \sum_j a_j \left(\sum_k w_k x_k\right)^j := \sum_J a_J w^J x^J . \quad (2.1)$$

The (two-layer) neural network is defined as a function of the form $g(x) = \sum_i \alpha_i \phi^{w_i}(x)$. Here ϕ is called the activation function, and each ϕ^{w_i} — a hidden unit. We refer to α_i's as the "upper layer" and w_i's as the "lower layer".

In this paper, we will consider activation functions $\phi(z) = e^z$ and its truncated version $\phi_d(z) = \sum_{k=0}^d z^k/k!$, i.e. e^z with higher than degree d terms truncated (in this case, we call it a *truncated neural network*). While a more common ϕ is the sigmoid function, the particular form of ϕ is not that important as long as it has "good" coverage of all the degrees in its Taylor series (see, for example, the discussion in Barron (1993)), and for analysis convenience, it is defined everywhere on the complex plane. Our particular choice is just for the sake of a cleaner analysis.

Learning. Here learning is defined to construct the representation of an unknown (or target) function, from some function family $\mathcal{F}$, by given random samples. In this paper, we focus on learning low degree polynomials, where the input distribution is drawn from $\mathbb{C}(1)^n$. We will comment when the results can be extended to the other distributions. Learning the neural network via gradient descent proceeds by minimizing the error $\|f - g\|_D^2$ as a function of weights w_s and α_s's of ϕ. To focus on the main ideas, we will assume that there are enough samples such that the empirical estimate of gradient is sufficiently accurate. So we will be working with the model where we do have the access to the gradient.

3. Random Initialization of Neural Network

In this section we prove that any polynomial can be represented using a linear combination of a sufficient number of hidden units with weights w_i initialized randomly. This is a strengthening of (Barron, 1993), who showed that there exist *some* weights w_i, for which a linear combination of the hidden units yields a given polynomial. In addition, we show that the weights at the top node has small norm. Compared to the representability result in Barron (1993), we require more units, $O(n^{2d})$, rather than $O(n^d)$ there.

Theorem 3.1. *[Representation Theorem] Let $\phi(z) = e^z$ and the distribution $D = \mathbb{C}(1)^n$. For $m \geq O(n^{2d}/\epsilon^2)$, we choose m random $w_1, w_2, \ldots, w_m$ from the distribution $\mathbb{C}(1/\sqrt{n})^n$. Then, with high probability, for any polynomial p of degree d and norm 1, there exist $\alpha_1, \ldots, \alpha_m$ where $\sum_i |\alpha_i|^2 = O(n^{2d}/m)$ such that $\|\sum_i \alpha_i \phi^{w_i} -$*

$p\|_D \le \epsilon$.

The result also holds whenever D is Gaussian distribution $\mathbf{N}(1)$, or the uniform distribution $\mathbf{U}(1)$.

To prove the theorem, we consider general ϕ and D first and then instantiate them by estimating the relevant parameters. The following is immediate from (2.1) and Fact 2.1.

Observation 3.2. *For any $x \in \mathbb{C}^n$ and any $J \in \mathbb{N}^n$, $\mathrm{E}_{w\sim\mathbb{C}(r)^n} \overline{w}^J \phi^w(x) = a_J r^{2|J|} x^J$.*

In the following, we will show, constructively, that for the given set of units, how to pick the top layer weights to approximate any given p. Define for any polynomial $p(x) = \sum_J b_J x^J$, $r \ge 0$, and weight vector w, $T_{p,r}(w) = \sum_J c_J(r) b_J \overline{w}^J$, where $c_J(r) \triangleq 1/(a_J r^{2|J|})$. By Observation 3.2, for any x,

$$\mathrm{E}_{w\sim\mathbb{C}(r)^n}[T_{p,r}(w)\phi^w(x)] = p(x)\,. \tag{3.1}$$

Hence $T_{p,r}(w)\phi^w$ is centered around p in the functional space. We can apply standard concentration bound to show that the average over m such terms, with w chosen independently and m large enough, one can approximate $p(x)$ arbitrarily close. More precisely, suppose that $w_1, \ldots, w_m$ are sampled from $\mathbb{C}(r)^n$. Let $\eta(x)$ denote the error $\frac{1}{m}\sum_{i=1}^m T_{p,r}(w)\phi^{w_i}(x) - p(x)$. We obtain,

$$\mathrm{E}_{w_1,\ldots,w_m} \|\eta\|_D^2 \le \frac{1}{m} \mathrm{E}_{w\sim\mathbb{C}(r)^n} |T_{p,r}(w)|^2 \|\phi^w\|_D^2\,. \tag{3.2}$$

Let $a(d) = \min_{|J|\le d} |a_J|$, and define $\|p\|_1 = \sum_J |b_J|$. When w is from $\mathbb{C}(r)^n$ for $r \le 1$, we have

$$\begin{aligned} |T_{p,r}(w)| &\le \sum_J |c_J(r) b_J w^J| = \sum_J |b_J w^J/(a_J r^{2|J|})| \\ &\qquad \text{by } \deg(p) = d \\ &= \sum_J |b_J/(a_J r^{|J|})| \le 1/(a(d) r^d) \sum_J |b_J| \\ &= \|p\|_1/(a(d) r^d)\,. \end{aligned} \tag{3.3}$$

Denote by $\beta_D(d,r) = \mathrm{E}_{w\sim\mathbb{C}(r)^n} \|\phi^w\|_D^2/r^{2d}$ and $\gamma_D(d) = \max_{\|p\|_D=1} \|p\|_1$. Plugging (3.3) into (3.2), we have

Lemma 3.3. *Whenever $m \ge \gamma_D(d)^2 \beta_D(d,r)/(\epsilon a(d))^2$, with high probability $\mathrm{E}_{w_1,\ldots,w_m} \|\eta\|_D \le \epsilon$ for any p where $\deg(p) \le d$ and $\|p\|_D = 1$.*

Note that the number of required hidden units is determined by various parameters dependent on ϕ and D.

Now consider $\phi(z) = e^z$. Then $a(d) \ge 1/d!$. Suppose that $D = \mathbb{C}(1)^n$. Then $\beta_D(d,r) = O(e^{2\sqrt{n}r}/r^{2d})$. Taking $r = O(1/\sqrt{n})$, we have $\beta_D(d, 1/\sqrt{n}) = O(n^d)$. Further, $\|p\|_D = \sum_J |b_J|^2$, so $\gamma_D(d) = O(\sqrt{n^d})$. Plugging these parameters into Lemma 3.3, we have $m = O(n^{2d}/\epsilon^2)$. In addition $\alpha_i = T_{p,r}(w_i)/m$, so

$$\sum_{i=1}^m |\alpha_i|^2 = \tfrac{1}{m^2} \sum_{i=1}^m |T_{p,r}(w_i)|^2 \le \tfrac{\|p\|_1^2}{m a(d) r^{2d}} = O(n^{2d}/m)\,.$$

The same bounds can be derived for the distributions such as standard Gaussian distribution in $\mathbf{R}^n$ and the uniform distribution in $[0,1]^n$. This proves Theorem 3.1.

4. Gradient Descent for Polynomials

In this section we show that gradient descent on a two-layer neural network can learn a polynomial function, given enough hidden units and small enough learning rate. The statement relies on the fact that the weights have been initialized randomly. The formal statement is in Theorem 4.2.

First we prove a warm up (simpler) statement: if we run gradient descent only in the upper layer (and leave intact the weights in the lower layer), then the gradient descent will converge to a network approximating the polynomial up to a small error. We use the representation theorem from the previous section. In this simpler statement, we also assume that we have access to the exact value of the gradient.

Then we show a more general statement, which reaches the same conclusion even if we run the generic gradient descent (i.e., on the entire network), assuming that the number of hidden units is sufficiently large. The main intuition behind the result is that the top-layer weights converge to a good state (as in the simplified theorem) before the modifications in the lower layer change enough to affect the overall representability. In particular, one step of a gradient descent will update the weights α_i of all the hidden units at once. Hence the "total speed" of convergence of weights α_i is essentially proportional to the number of hidden units, whereas the speed of change of each w_i is not. Once we have enough hidden units, namely $n^{O(d)}$, the speed of convergence of α_i is sufficiently high to overtake the changes to the weight w_i of any particular hidden unit.

Theorem 4.1. *Fix some degree-d polynomial f of norm 1, and desired error $\epsilon > 0$. Consider a two-layer neural network with $m = \Omega(n^{2d}/\epsilon^2)$ hidden units. We initialize the α_i's such that $\|\alpha\| \le 1$ (e.g., $\alpha = 0$ would do), and choose the weights w_i randomly from $\mathbb{C}(1/\sqrt{n})^n$. Consider the algorithm where we run the gradient descent on the weights in the upper layer (keeping weights in lower layer fixed), with access to exact gradient. Then, for a learning rate $\lambda < 1/m$, the algorithm will converge to a network g such that $\|g - f\| \le \epsilon$ in $O\left(\frac{n^{2d}}{\lambda\epsilon^2 m}\right)$ steps.*

Proof. As in preliminaries, g denotes the function produced by the neural network: $g(x) = \sum_{i=1}^m \alpha_i \phi(w_i^t \cdot x)$.

Abusing notation, we will think of all functions in the base of monomials x^J. In particular, g in the function space is a sum of m vectors $p_1 \dots p_m$, where $p_i = \phi^{w_i}$ depends on vector w_i: $g = \sum_{i=1}^m \alpha_i p_i$.

The gradient descent minimizes the quantity $E = \langle e, e\rangle = e^* \cdot e$, for the error function $e = f - g$. We would like to take gradient with respect to α_i, but E is not analytic with respect to e. Instead we rely on *Wirtinger calculus*, and consider the following gradient:[1]

$$\left(\frac{\partial e}{\partial \alpha_i}\right)^* \cdot e = -{p_i}^* e = -\langle p_i, e\rangle.$$

In one step of the gradient descent the new function g' is

$$g' = \sum_i p_i \cdot (\alpha_i + \lambda \langle p_i, e\rangle),$$

and the new error function:

$$e' = f - g' = e - \lambda \sum_i p_i \langle p_i, e\rangle = \left(I - \lambda \sum_i p_i p_i^*\right) e,$$

where I is the identity matrix. Let P be the matrix whose columns are p_i's. Then $\sum_i p_i p_i^* = PP^*$.

After l iterations of the gradient descent, the error is

$$e^{(l)} = (I - \lambda PP^*)^l e^{(0)},$$

where $e^{(0)}$ is the starting error function.

We apply the representation Theorem 3.1 to $e^{(0)}$ to obtain $e^{(0)} = x + r$, where $\|r\| \le \epsilon$, and x can be expressed as $x = \sum_i a_i p_i$, with $\|a\|^2 \le O(n^{2d}/m)$ for $a \triangleq (a_1, \dots a_m)^t$. Note that $x = Pa$.

Then we can rewrite the above as:

$$e^{(l)} = (I - \lambda PP^*)^l (r + Pa) = (I - \lambda PP^*)^l r + (I - \lambda PP^*)^l Pa.$$

Let's see what happens after one iteration to the second term: $(I - \lambda PP^*)Pa = Pa - \lambda PP^*Pa = P(I - \lambda P^*P)a$. Hence, $(I - \lambda PP^*)^l Pa = P(I - \lambda P^*P)^l a$.

Let $a^{(l)} \triangleq (I - \lambda P^*P)^l a$, i.e., $e^{(l)} = Pa^{(l)}$. Suppose $\|Pa^{(l)}\| \ge \epsilon$. Then, we have that

$$\begin{aligned}\|a^{(l+1)}\|^2 &= (a^{(l+1)})^* a^{(l+1)} \\ &= ((I - \lambda P^*P)a^{(l)})^* (I - \lambda P^*P)a^{(l)} \\ &= \|a^{(l)}\|^2 - 2\lambda \|Pa^{(l)}\|^2 + \lambda^2 \|P^*Pa^{(l)}\|^2. \quad (4.1)\end{aligned}$$

As we have that $\|P\| \le \sqrt{m}$ and $\lambda \|P\|^2 \le 1$, we conclude that $\|a^{(l+1)}\|^2 \le \|a^{(l)}\|^2 - \lambda \|Pa^{(l)}\|^2 \le \|a^{(l)}\|^2 - \lambda\epsilon^2$.

[1] We essentially consider the variables and their conjugates as independent variables, and then take derivative with respect to α_i^*.

Finally, note that we can have at most $\frac{\|a^{(0)}\|^2}{\lambda\epsilon^2}$ steps, as $\|a^{(l)}\| \ge 0$ always. Thus, after that many steps, we must have that $\|Pa^{(l)}\| \le \epsilon$, and hence $\|e^{(l)}\| \le \|r\| + \|Pa^{(l)}\| \le 2\epsilon$. Since $\|a^{(0)}\|^2 \le O(n^{2d}/m)$, we obtain a total of $O\left(\frac{n^{2d}}{\epsilon^2 \lambda m}\right)$ steps. □

We now continue to our main result, which shows that the gradient descent on the entire network will converge as well, given enough hidden units and small enough learning rate. The proof of the theorem appears in the supplementary material.

Theorem 4.2 (General gradient descent)**.** *Fix target error $\epsilon > 0$ and degree $d \ge 1$. Suppose the weights α are initialized to zero and w_i's are random $\mathbb{C}(1/\sqrt{n})^n$. Assume the number of hidden units is $m = \Omega(n^{6d}/\epsilon^3)$ and the learning rate is $\lambda \le 1/4m$. Then, given a degree-d polynomial $f(x)$ of unit norm, the gradient descent will converge to a net, which approximates f up to error ϵ. The number of steps required is $O\left(\frac{n^{2d}}{\lambda\epsilon^2 m}\right)$, and the number of samples required is $M = m^{O(1)}$.*

5. Random Perturbation at Local Minima

The results of the previous section show that for a sufficiently large neural network, with high probability over the random initialization of the weights, the gradient descent will learn a degree d polynomial. In this section, we prove a conceptually compelling, though incomparable result: we show that for sufficiently large networks, in a large region of the parameter space, while there may exist local minima, there are *no robust local minima*. That is, for *any* point in the specified parameter space, as long as the error is not vanishingly small, with constant probability a random perturbation will reduce the error by at least $1/n^{O(d)}$ factor (see Theorem 5.1). Because of the conditions on the parameters, we can not use this theorem to conclude that gradient descent will always converge to the global optima from any initial neural network initialization. Nevertheless, this provides a rigorous explanation for why local optima may not be so damaging for neural networks in practice. Furthermore, this perspective may prove useful for going beyond the results of the previous section, for example for addressing the harder questions posed in the later Section 6.

We stress that this result relies crucially on the fact that we allow complex valued weights. In fact we also show such a result is not true when the weights are real-valued.

We now state the main result of this section. Define the fourth-norm $\|\alpha\|_4 = (\sum_i |\alpha_i|^4)^{1/4}$.

Theorem 5.1. *There exist constant $c_1, c_2, c_3, c_4 > 0$ such that for a truncated neural network $g = \sum_i \alpha_i \phi_d^{w_i}$ where $m = \Omega(n^{c_1 d})$, $\|w_i\| = O(\log n)$, and $\|\alpha\|_4 =$*

$O(\|\alpha\|/n^{c_2 d})$, if $\|e\|_D = \Omega(\|\alpha\| n^{c_3 d})$, then a perturbation of each w_{sj} drawn from $\mathbb{C}(1/\sqrt{n})$ reduces the total error by a factor of at least $1+1/n^{c_4 d}$, with constant probability.

Note that by Theorem 3.1, when m is large enough, $m = n^{\Omega(d)}$, the conditions in the theorem can all be met.

We first sketch the proof idea. Under a given distribution D, for a target function f and a neural network g, consider the error function $\|e\|_D^2 = \|g - f\|_D^2$. For a local perturbation $g + \Delta g$ of g, $\|g + \Delta g - f\|_D^2 = \|e\|_D^2 + 2\,\mathrm{Re}(\langle \Delta g, e\rangle_D) + \|\Delta g\|_D^2$. Hence the change in error is $\Delta\|e\|_D^2 = 2\,\mathrm{Re}(\langle \Delta g, e\rangle_D) + \|\Delta g\|_D^2$. We shall show that, with constant probability, the linear term is negative and overwhelms the quadratic term if the perturbation is sufficiently small. The proof consists of two steps. First we consider a single hidden unit. We show that a local perturbation can create non-negligible correlation with any bounded degree polynomial. Secondly we show that, by the anti-concentration inequality[2], when we perturb many hidden units independently, the aggregated correlation is still large and, when the number of hidden units is large enough, exceeds the quadratic term, which can be bounded by the standard concentration bound.

Our claim applies when the error function $e = g - f$ has bounded degree d, which is the reason the result applies to networks with a *truncated* activation function ϕ_d where $\phi_d(z) = \sum_{0\le j\le d} a_j z^j$. For the simplicity of notation, we will simply write it as $\phi(z)$.

Here it is important that w is complex. The distribution D on x can be over the reals, for example, D can be standard Gaussian distribution $\mathbf{N}(1)^n$ in $\mathbf{R}^n$ or the uniform distribution $\mathbf{U}(1)^n$ over $[-1,1]^n$. We first show our statement for $D = \mathbb{C}(1)^n$. Then we extend it to the case when $D = \mathbf{N}(1)^n$ or $\mathbf{U}(1)^n$. We will only sketch the main steps of proofs in this section.

We give further details of the proof; the missing proofs are in the supplementary material.

5.1. Random perturbation of one hidden unit

We first show that for a single hidden unit, a random perturbation will create large correlation with any bounded degree polynomial. Recall that $\phi^w(x) = \sum_J a_J w^J x^J$, and $a(d) = \min_{|J|\le d} |a_J|$. For $x \in \mathbb{C}^n$, we define $\|x\|_\infty = \max_j |x_j|$. We denote by $\Delta_\delta \phi^w = \phi^{w+\delta} - \phi^w$ as the perturbation of a hidden unit ϕ^w by δ. We have

Theorem 5.2. *For any $x \in \mathbb{C}^n$, $\mathrm{E}_{\delta\in\mathbb{C}(r)^n}[\Delta_\delta \phi^w(x)] = 0$. For any η such that $\deg(\eta) \le d$, and $\|\eta\|_D \ge 1$, we have that for any $0 < r \le 1$ and $\|w\|_\infty \le rL$,*

$$\mathrm{E}_{\delta\in\mathbb{C}(r)^n}\left[\mathrm{Re}(\langle \Delta_\delta \phi^w, \eta\rangle_D)^2\right] = \Omega\left(\frac{r^{2d} a(d)^2}{n^d (L+1)^{2d}}\right).$$

[2]This is where we need the condition on $\|\alpha\|_4$.

Proof sketch. Clearly for any x, $\Delta_\delta \phi^w(x)$ can be written as a polynomial in δ without constant term. By Fact 2.1, $\mathrm{E}_{\delta\in\mathbb{C}(r)^n}[\Delta_\delta \phi^w(x)] = 0$. This implies that $\mathrm{E}_{\delta\in\mathbb{C}(r)^n}[\langle \Delta_\delta \phi^w, \eta\rangle_D] = 0$,

Write $B(w) = \langle \phi^w, \eta\rangle_D$. As η is a polynomial with degree d, so is $B(w)$. By the above, we have

$$\mathrm{E}_{\delta\in\mathbb{C}(r)^n}[B(w+\delta)] = B(w)\,. \tag{5.1}$$

We lower bound $\mathrm{E}_{\delta\in\mathbb{C}(r)^n}[|B(w+\delta) - B(w)|^2]$ as follows. We first show the case when $w = 0$. Then we apply the "shifting" lemma (Lemma 5.4) to complete the proof.

Lemma 5.3. *For $0 \le r \le 1$, $\mathrm{E}_{\delta\in\mathbb{C}(r)^n}[|B(\delta) - B(0)|^2] \ge r^{2d} a(d)^2$.*

Lemma 5.4. *Suppose that f is a degree d polynomial on n variables. Let $v = (v_1, \ldots, v_n)$ such that $\|v\|_\infty \le L$. Let $f_v(x) = f(v + x)$. Then $\|f_v\|^2 \le n^d (L+1)^{2d} \|f\|^2$.*

By the above two lemmas, we can show that

$$\mathrm{E}_{\delta\in\mathbb{C}(r)^n}[|\langle \Delta_\delta \phi^w, \eta\rangle_D|^2] = \Omega\left(r^{2d} a(d)^2/(n^d (L+1)^{2d})\right),$$

and further transfer this bound to $(\mathrm{Re}\,\langle \Delta_\delta \phi^w, \eta\rangle_D)^2$, to complete the proof. □

We note that the above theorem holds for large range of r and w. But to suppress the second order term, we will only need the theorem in the range where $r = O(1/\sqrt{n})$ and $\|w\| = O(\log n)$.

5.2. Random perturbation of many hidden units

Now consider a neural network $g(x) = \sum_{i=1}^m \alpha_i \phi^{w_i}(x)$, where each $\|w_i\| = O(\log n)$. Let $g'(x) = \sum_{i=1}^m \alpha_i \phi^{w_i+\delta_i}(x)$, where each δ_i is i.i.d. from $\mathbb{C}(1/\sqrt{n})^n$.

$$\begin{aligned}\|e'\|_D^2 - \|e\|_D^2 &= \|(g' - g) + e\|_D^2 - \|e\|_D^2 \\ &= \|g' - g\|_D^2 + 2\,\mathrm{Re}(\langle g' - g, e\rangle_D)\,. \end{aligned}\tag{5.2}$$

First consider $\|g' - g\|_D^2 = \|\sum_{i=1}^m \alpha_i \Delta_{\delta_i} \phi^{w_i}\|_D^2$. We can view $\Delta_\delta \phi^w$ as a vector in the functional space, so each $\Delta_{\delta_i} \phi^{w_i}$ is a random vector. We have shown that $\mathrm{E}_{\delta_i}[\Delta_{\delta_i} \phi^{w_i}] = 0$, and for $\|w_i\| = O(\log n)$ and $r = O(1/\sqrt{n})$, $\mathrm{E}_{\delta_i\sim\mathbb{C}(r)^n} \|\Delta_{\delta_i} \phi^{w_i}\|_D^2 = O(n^{O(1)})$. Since $\Delta_{\delta_i} \phi^{w_i}$'s are independent, by standard concentration bound, with high probability

$$\|g' - g\|_D^2 = O(n^{O(1)} \|\alpha\|^2)\,. \tag{5.3}$$

For the linear term $\mathrm{Re}(\langle g' - g, e\rangle_D)$, by using Theorem 5.2 and anti-concentration inequality, we can show that when $\|\alpha\|_4 \le \|\alpha\|/n^{cd}$, with constant probability, say $1/4$,

$$\mathrm{Re}(\langle g' - g, e\rangle_D) = -\Omega(\|\alpha\| \|e\|_D / n^{O(d)})\,. \tag{5.4}$$

Combining (5.2,5.3,5.4), we have whenever $\|\alpha\|_4 \leq n^{cd}\|\alpha\|$ and $\|\alpha\| \leq \|e\|_D/n^{O(d)}$, we have that $\|e'\|_D^2 \leq \|e\|_D^2 - \|\alpha\|\|e\|_D/n^{O(d)}$ with constant probability. Hence, we have proved the main theorem.

5.3. Extension to distributions on reals

The above proof can also be extended to the case where the x is not complex but chosen from a Gaussian distribution $\mathbf{N}(1)^n$ in $\mathbf{R}^n$ or uniform distribution $\mathbf{U}(1)^n$ on $[-1,1]^n$. This follows from the following observation that relates the norm of a polynomial under different distributions.

Observation 5.5. *Let $P(x)$ be a degree d polynomial. Then $\|P\|_D = \Omega(1/d^{d/2})\|P\|_{\mathbb{C}(1)^n}$ and $O(d^{d/2}\|P\|_{\mathbb{C}(1)^n})$, where $D = \mathbf{N}(1)^n$ or $\mathbf{U}(1)^n$.*

The above observation implies that if we replace $\mathbb{C}(1)^n$ by $\mathbf{N}(1)^n$ or $\mathbf{U}(1)^n$, the bound in Theorem 5.2 is only affected by a factor dependent on d only (d^d or 2^d). Since we assume d to be constant, we have:

Corollary 5.6. *The same statement in Theorem 5.1 holds when $D = \mathbf{N}(1)^n$ or $D = \mathbf{U}(1)^n$.*

5.4. Robust local minima for real weights

The perturbation theorem 5.1 uses random perturbation in the complex plane to escape a local minimum. It is natural to ask whether real-valued perturbation would be sufficient instead. We show that this is not the case: there are examples, where a real-valued perturbation does not improve the error. This suggest that using complex perturbations may be useful.

Lemma 5.7. *Consider a network with activation function $\phi(z) = \sum_{l=0}^{d} a_l z^l$, where $a_l \geq 0$, on a neural network with one layer of hidden units, with real weights; the input distribution is $x \in \mathbf{U}(1)^n$. There exist a set of network parameters where the gradient is zero and a random perturbation on the weights $\{w_i\}_i$ goes in the direction away from the target function, i.e., $\|e'\|_D > \|e\|_D$ with high probability.*

Proof. We will give a construction of a set of parameters, which are a local minimum for the gradient descent, and a real-valued random perturbation of weights is expected to increase the error. This point is where all hidden units have identical weights $w_i = 0$ and $\alpha_i = 1/m$ (for m hidden units), so that $\sum \alpha_i = 1$.

We show why local perturbation does not reduce error, except with a very small probability. Let δ_i be the perturbation in weight w_i; for concreteness, suppose each perturbation is uniform from $[-\lambda, \lambda]$ for some $\lambda \ll 1/m$. Let $\Delta g = g' - g$ denote the change in the output function of the neural net. We argue that $\mathrm{E}_{\delta_i}[\Delta g]$ is non-zero. Note that the change Δg can be written as a polynomial in the change in the weights δ_{ij}. Fix one particular hidden unit i, and fixed $j \in [n]$, and consider the term corresponding to second Legendre polynomial in x_j, $L_2(x_j)$ (remember that the Legendre polynomials are the basis for our input distribution, so we are considering one "coordinate" in the polynomial basis). We claim that its coefficient is positive: in fact it is a (positive) combination of even powers of $\delta_{i,j'}$ for all $j' \in [n]$. In particular, say in a term $a_l(\delta_i x)^l$, we have contribution to $L_2(x_j)$ only from terms of the form $\delta^J x^J$, where vector J is even (any other term has correlation 0 with $L_2(x_j)$).

We can now choose $e = g - f$ so that $\langle e, \Delta g\rangle$ is positive by choosing $f(x) = g(0) - \sum_j L_2(x_j)$. Then the change in $\langle e, e\rangle$ is: $\Delta\langle e, e\rangle = \langle e', e'\rangle - \langle e, e\rangle = 2\langle e, \Delta g\rangle + \langle \Delta g, \Delta g\rangle$. The error strictly increases with probability 1. It is also clear why the gradient is zero: $\frac{\partial g}{\partial w_{ij}}$, when all $w_i = 0$, is composed only of a linear in x terms, whereas $e = g - f$ is has no constant or linear terms. □

We remark that the above holds even if we perform a small random perturbation on the weights α_i as well (it suffices that α_i and perturbed versions remain positive).

6. Learning sparse polynomials

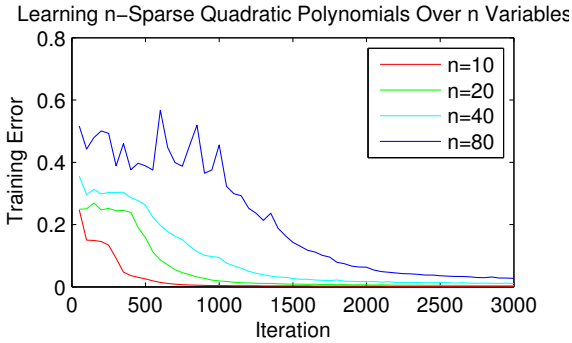
In this section we study whether smaller neural networks are sufficient for learning sparse polynomials (containing few monomials). As an intermediary step towards this goal, we will also consider the setting where the polynomial only depends on a small subset of the n variables (and hence is also sparse). These questions can be viewed as clean and potentially theoretically tractable special cases of the general question of understanding the relation between the representation complexity and the learning complexity of neural networks for some given class of functions.

6.1. Learning n-sparse polynomials

Can a neural network with $O(n)$ hidden units learn a quadratic or cubic polynomial that has $\approx n$ monomials? We provide strong empirical evidence (see Fig. 1) suggesting that, for the case of n-sparse polynomials over n variables, a neural network with $O(n)$ hidden units *can* learn the function. We train the net using $5n$ hidden units while varying n through the values $10, 20, 40$, and 80. The polynomial is constructed using randomly chosen n monomials. The plots show that the training error drops significantly after a reasonable number of iterations that depends on n.

6.2. Learning polynomials over few variables

As an intermediary step towards the sparse polynomial case, we investigate whether a small neural network suffices to learn a sparse polynomial which also depends only



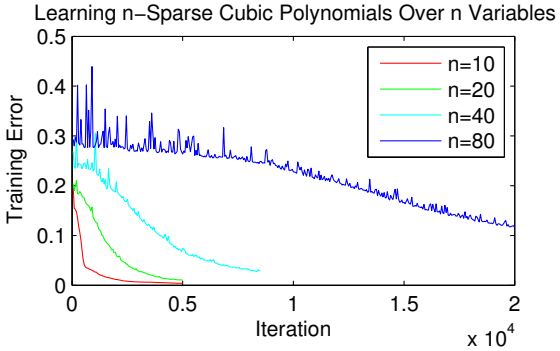


Figure 1. In the above plots the neural networks had $5n$ hidden units, and the polynomials were chosen by selecting each of the n monomials uniformly at random from the $O(n^2)$ (in the quadratic case), or $O(n^3)$ (in the cubic case) possible monomials.

on k variables. Here, a simpler goal may be to prove that gradient descent learns the polynomial using only $k^{O(d)}$ hidden units instead of $n^{O(d)}$. We are unable to prove this but provide some evidence in this direction. First we show (Lemma 6.1) that assuming $x \in \mathbb{C}(1)^n$, at the termination of the gradient descent, the final function output by the net will depend on the k relevant variables only. We show a similar result for the (more realistic) case where $x \in \mathbf{N}(1)^n$, for a specific transfer function ψ, built using Hermite polynomials. We will use $H_J(x)$ to denote the Hermite polynomial over x corresponding to a vector J of degrees in the n variables in x. (Some details are deferred to full version.)

Lemma 6.1. *If $x \in \mathbb{C}(1)^n$ and the target function f does not depend on a variable x_i, then: whenever the gradient descent converges to a point with zero gradient, the output g does not depend on x_i. This also holds for the input distribution $x \in \mathbf{N}(1)^n$, provided one uses a special activation function $\psi^w(x) = \sum_J a_J H_J x^J w^J$.*

Proof. The main idea is that if a variable, say x_1, is not used in f, then the error e can be written as a sum of two polynomials, one that does not involve x_1 and another that has x_1 in every monomial and these two are orthonormal.

Let $e = f - g = f - \sum_s \alpha_i \psi(w_i \cdot x)$. By expanding the polynomial ψ and gathering terms that are dependent on x_1 and others we get $e = q(x_2, x_3, .., x_n) + h(x)$ where $h(x) = \sum_i \alpha_i \sum_J c_J w_i^J x^J$ where each J has non zero degree in x_1 (that is J_1 is non zero) and q does not depend on either x_1 (or its weights $w_{i,1}$). All we need to show that $\sum_i |\langle \frac{\partial e}{\partial w_{i,1}}, e \rangle|^2$ is non-zero. So the partial derivative of the error with respect to $w_{i,1}$ is

$$\frac{\partial e}{\partial w_{i,1}} = \frac{\partial}{\partial w_{i,1}} (\sum_l \alpha_l \sum_J c_J w_k^J x^J) = \alpha_i \sum_J c_J J_1 w_i^J x^J / w_{i,1} \,.$$

Therefore

$$\sum_i w_{i,1} \frac{\partial e}{\partial w_{i,1}} = \sum_i \alpha_i \sum_J c_J J_1 w_i^J x^J = \sum_J J_1 \sum_i \alpha_i c_J w_i^J x^J$$

and hence

$$\begin{aligned} &\langle \sum_i w_{i,1} \frac{\partial e}{\partial w_{i,1}}, e \rangle \\ &= \langle \sum_J J_1 (\sum_i \alpha_i c_J w_i^J) x^J, \sum_J (\sum_i \alpha_i c_J w_i^J) x^J \rangle \\ &= \sum_J J_1 |\sum_i \alpha_i c_J w^J|^2 \,, \end{aligned}$$

which must be non-zero as $h = \sum_i \alpha_i \sum_J c_J w_i^J x^J = \sum_J (\sum_i \alpha_i c_J w_i^J) x^J$ is non-zero (each J_1 is non-zero).

This means that $\sum_i w_{i,1} \langle \frac{\partial}{\partial w_{i,1}} e, e \rangle$ is non-zero. Thus at least one of the $\langle \frac{\partial}{\partial w_{i,1}} e, e \rangle$ is non-zero, which means that the gradient descent has not converged yet.

A similar proof works for the case when x is real and we instead use a modified polynomial ψ where each monomial is replaced by a Hermite polynomial. The proof is based on the orthonormality of these polynomials over $\mathbf{N}(1)^n$. □

Further we point out that neural networks are able to learn polynomials based on their best possible sparsity under any orthonormal transform; i.e., the notion of sparsity is independent of the chosen coordinate system. This is because the neural net works with dot products of the input point.

Observation 6.2. *If a neural network can learn a k-sparse polynomial in time $T(k, d, n)$, then it can learn also learn any polynomial that is k-sparse under any orthonormal transform in the same time complexity.*

Proof. This follows from the rotational invariance of the gradient descent process when g is a function of $w_i^t.x$ over the different hidden units i. That is rotating the coordinate system does not change the gradient descent process. □

In order to show the gradient descent succeeds with $k^{O(d)}$ units, we need to assume a "High-Rank Condition" (a variant of the condition in Theorem 3.1) for similar analysis to Section 4 to hold. But we are currently unable to prove the "high-rank condition" and leave it as an open question.

References

Alekhnovich, Michael. More on average case vs approximation complexity. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2003.

Andoni, Alexandr, Panigrahy, Rina, Valiant, Gregory, and Zhang, Li. Learning sparse polynomial functions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.

Applebaum, Benny, Ishai, Yuval, and Kushilevitz, Eyal. Cryptography in nc^0. *SIAM Journal on Computing*, 36 (4):845–888, 2006.

Applebaum, Benny, Barak, Boaz, and Wigderson, Avi. Public-key cryptosystem from different assumptions. In *Proceedings of the Symposium on Theory of Computing (STOC)*, 2010.

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

Barron, Andrew R. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14: 115–133, 1994.

Bengio, Y. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2009.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

Bengio, Yoshua. Deep learning of representations: Looking forward. *arXiv preprint arXiv:1305.0445*, 2013.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. In *ICML*, 2013.

Hinton, G. E., Osinderoand, S., and Teh, Y. A fast learning algorithm for deep belief nets. *Neural Computation*, 18: 1527–1554, 2006.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Peikert, Chris. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the Symposium on Theory of Computing (STOC)*, 2009.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. Efficient learning of sparse representations with an energy-based model. *NIPS*, 2006.

Regev, Oded. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pp. 84–93, New York, NY, USA, 2005. ACM. ISBN 1-58113-960-8. doi: http://doi.acm.org/10.1145/1060590.1060603.

Saxe, Andrew M, McClelland, James L, and Ganguli, Surya. Dynamics of learning in deep linear neural networks. *NIPS Workshop on Deep Learning*, 2013.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, LeCun, Yann, and Fergus, Rob. Regularization of neural networks using dropconnect. In *ICML*, 2013.