

The power of deeper networks for expressing natural functions

David Rolnick*, Max Tegmark†

*Dept. of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 and
Dept. of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139*

(Dated: May 17, 2017)

It is well-known that neural networks are universal approximators, but that deeper networks tend to be much more efficient than shallow ones. We shed light on this by proving that the total number of neurons m required to approximate natural classes of multivariate polynomials of n variables grows only linearly with n for deep neural networks, but grows exponentially when merely a single hidden layer is allowed. We also provide evidence that when the number of hidden layers is increased from 1 to k , the neuron requirement grows exponentially not with n but with $n^{1/k}$, suggesting that the minimum number of layers required for computational tractability grows only logarithmically with n .

I. INTRODUCTION

Deep learning has lately been shown to be a very powerful tool for a wide range of problems, from image segmentation to machine translation. Despite its success, many of the techniques developed by practitioners of artificial neural networks (ANNs) are heuristics without theoretical guarantees. Perhaps most notably, the power of feedforward networks with many layers (*deep* networks) has not been fully explained. The goal of this paper is to shed more light on this question and to suggest heuristics for how deep is deep enough.

It is well-known [1–3] that nonlinear neural networks with a single hidden layer can approximate any function under reasonable assumptions, but it is possible that the networks required will be extremely large. Recent authors have shown that some functions can be approximated by deeper networks much more efficiently (i.e. with fewer neurons) than by shallower ones. However, many of the functions in question are complicated or arise from “existence proofs” without explicit constructions, and the results often apply only to types of network rarely used in practice.

Deeper networks have been shown to have greater representational power with respect to various notions of complexity, including piecewise linear decision boundaries [4] and topological invariants [5]. Recently, Poole et al. [6] and Raghu et al. [7] showed that the trajectories of input variables attain exponentially greater length and curvature with greater network depth. Work including [8] and [9] shows that there exist functions that require exponential width to be approximated by a shallow network. Mhaskar, Liao, and Poggio [10], in considering compositional functions with this property, inquire whether explicit examples must be pathologically complicated.

Various authors have also considered the power of deeper

network of types other than the standard feedforward model. The problem has also been posed for sum-product networks [11] and restricted Boltzmann machines [12]. Cohen, Sharir, and Shashua [13] showed, using tools from tensor decomposition, that shallow arithmetic circuits can express only a measure-zero set of the functions expressible by deep circuits. A weak generalization of this result to convolutional neural networks was shown in [14].

In summary, recent years have seen a wide variety of theoretical demonstrations of the power of deep neural networks. It is important and timely to extend this work to make it more concrete and actionable, by deriving resource requirements for approximating natural classes of functions using today’s most common neural network architectures. Lin, Tegmark, and Rolnick [15] recently proved that it is exponentially more efficient to use a deep network than a shallow network when approximating the product of input variables. In the present paper, we will greatly extend these results to include broad natural classes of multivariate polynomials, and to tackle the question of how resource use depends on the precise number of layers. Our results apply to standard feedforward ANNs with general nonlinearities and are borne out by empirical tests.

The rest of this paper is organized as follows. In §II B, we consider the complexity of approximating a multivariate polynomial $p(\mathbf{x})$ using a feedforward neural network with input $\mathbf{x} = (x_1, \dots, x_n)$. We show (Theorem II.4) that for general sparse p , exponentially many neurons are required if the network is shallow, but at most linearly many for a deep network. For monomials p , we calculate (Theorem II.1) exactly the minimum number of neurons required for a shallow network. These theorems apply for all nonlinear activation functions σ with nonzero Taylor coefficients; a slightly weaker result (Theorem II.2) holds for an even broader class of σ .

In §II C, we present similar results (Propositions II.5 and II.6) for approximating univariate polynomials. In this case, shallow networks require linearly many neurons while for deep networks it suffices to use only logarithmically many neurons. In §II D, we tie the difficulty of

*drolnick@mit.edu

†tegmark@mit.edu

approximating polynomials by shallow networks to the complexity of tensor decomposition (Proposition II.7).

In §III A, we consider networks with a constant number k of hidden layers. For input of dimension n , we show (Theorem III.1) that products can be approximated with a number of neurons exponential in $n^{1/k}$, and justify our theoretical predictions with empirical results. While still exponential, this shows that problems unsolvable by shallow networks can be tractable even for $k > 1$ of modest size.

Finally, in §III B, we compare our results on feedforward neural networks to prior work on the complexity of Boolean circuits. We conclude that these problems are independent, and therefore that established hard problems in Boolean complexity do not provide any obstacle to analogous results for standard deep neural networks.

II. THE INEFFICIENCY OF SHALLOW NETWORKS

In this section, we compare the efficiency of shallow networks (those with a single hidden layer) and deep networks at approximating multivariate polynomials.

A. Definitions

Let $\sigma(x)$ be a nonlinear function, k a positive integer, and $p(\mathbf{x})$ a multivariate polynomial of degree d . We define $m_k(p, \sigma)$ to be the minimum number of neurons (excluding input and output) required to approximate p with a neural net having k hidden layers and nonlinearity σ , where the error of approximation is of degree at least $d+1$ in the input variables. Thus, in particular, $m_1(p, \sigma)$ is the minimal integer m such that:

$$\sum_{j=1}^m w_j \sigma \left(\sum_{i=1}^n a_{ij} x_i \right) = p(\mathbf{x}) + \mathcal{O}(x_1^{d+1} + \dots + x_n^{d+1}).$$

Note that approximation up to degree d allows us to approximate any polynomial to high precision as long as the input variables are small enough. In particular, for homogeneous polynomials of degree d , we can adjust the weights so as to scale each variable by a constant $\lambda \ll 1$ before input, and then scale the output by $1/\lambda^d$, in order to achieve arbitrary precision.

We set $m(p, \sigma) = \min_{k \geq 0} m_k(p, \sigma)$. We will show that there is an exponential gap between $m_1(p, \sigma)$ and $m(p, \sigma)$ for various classes of polynomials p .

B. Multivariate polynomials

The following theorem generalizes a result of Lin, Tegmark, and Rolnick [15] to arbitrary monomials. By setting $r_1 = r_2 = \dots = r_n = 1$ below, we recover their result that the product of n numbers requires 2^n neurons in a shallow network but can be done with linearly many neurons in a deep network.

Theorem II.1. *Let $p(\mathbf{x})$ denote the monomial $x_1^{r_1} x_2^{r_2} \dots x_n^{r_n}$, with $N = \sum_{i=1}^n r_i$. Suppose that the nonlinearity $\sigma(x)$ has nonzero Taylor coefficients up to x^N . Then:*

$$m_1(p, \sigma) = \prod_{i=1}^n (r_i + 1), \quad (1)$$

$$m(p, \sigma) \leq \sum_{i=1}^n (7 \lceil \log_2(r_i) \rceil + 4), \quad (2)$$

where $\lceil x \rceil$ denotes the smallest integer that is at least x .

Proof. Without loss of generality, suppose that $r_i > 0$ for $i = 1, \dots, n$. Let X be the multiset¹ in which x_i occurs with multiplicity r_i .

We first show that $\prod_{i=1}^n (r_i + 1)$ neurons are *sufficient* to approximate $p(x)$. Appendix A in [15] demonstrates that for variables y_1, \dots, y_N , the product $y_1 \dots y_N$ can be approximated as a linear combination of the 2^N functions $\sigma(\pm y_1 \pm \dots \pm y_N)$.

Consider setting y_1, \dots, y_N equal to the elements of multiset X . Then, we conclude that we can approximate $p(\mathbf{x})$ as a linear combination of the functions $\sigma(\pm y_1 \pm \dots \pm y_N)$. However, these functions are not all distinct: there are $r_i + 1$ distinct ways to assign \pm signs to r_i copies of x_i (ignoring permutations of the signs). Therefore, there are $\prod_{i=1}^n (r_i + 1)$ distinct functions $\sigma(\pm y_1 \pm \dots \pm y_N)$, proving that $m_1(p, \sigma) \leq \prod_{i=1}^n (r_i + 1)$.

We now adapt methods introduced in [15] to show that this number of neurons is also *necessary* for approximating $p(\mathbf{x})$. Let $m \equiv m_1(p, \sigma)$ and suppose that $\sigma(x)$ has the Taylor expansion $\sum_{k=0}^{\infty} \sigma_k x^k$. Then, by grouping terms of each order, we conclude that there exist constants a_{ij} and w_j such that

$$\sigma_N \sum_{j=1}^m w_j \left(\sum_{i=1}^n a_{ij} x_i \right)^N = p(x) \quad (3)$$

$$\sigma_k \sum_{j=1}^m w_j \left(\sum_{i=1}^n a_{ij} x_i \right)^k = 0 \quad \text{for } 0 \leq k \leq N-1. \quad (4)$$

¹ That is, a collection of elements of X , in which each element is allowed to occur multiple times.

For each $S \subseteq X$, let us take the derivative of equations (3) and (4) by every variable that occurs in S , where we take multiple derivatives of variables that occur multiple times. This gives

$$\frac{\sigma_N \cdot N!}{|S|!} \sum_{j=1}^m w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^n a_{ij} x_i \right)^{N-|S|} = \prod_{i \notin S} x_i, \quad (5)$$

$$\frac{\sigma_k \cdot k!}{|S|!} \sum_{j=1}^m w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^n a_{ij} x_i \right)^{k-|S|} = 0 \quad (6)$$

for $|S| \leq k \leq N-1$. Observe that there are $\prod_{i=1}^n (r_i + 1)$ choices for S , since each variable x_i can be included anywhere from 0 to r_i times. Define \mathbf{A} to be the $(\prod_{i=1}^n (r_i + 1)) \times m$ matrix with entries $A_{S,j} = \prod_{h \in S} a_{hj}$. We claim that \mathbf{A} has full row rank. This would show that the number of columns m is at least the number of rows $\prod_{i=1}^n (r_i + 1)$, proving the desired lower bound on m .

Suppose towards contradiction that the rows $A_{S_\ell, \bullet}$ admit a linear dependence:

$$\sum_{\ell=1}^r c_\ell A_{S_\ell, \bullet} = \mathbf{0},$$

where the coefficients c_ℓ are nonzero and the S_ℓ denote distinct subsets of X . Set $s = \max_\ell |S_\ell|$. Then, take the dot product of each side of the above equation with the vector with entries (indexed by j) equal to $w_j (\sum_{i=1}^n a_{ij} x_i)^{N-s}$:

$$\begin{aligned} 0 &= \sum_{\ell=1}^r c_\ell \sum_{j=1}^m w_j \prod_{h \in S_\ell} a_{hj} \left(\sum_{i=1}^n a_{ij} x_i \right)^{N-s} \\ &= \sum_{\ell(|S_\ell|=s)} c_\ell \sum_{j=1}^m w_j \prod_{h \in S_\ell} a_{hj} \left(\sum_{i=1}^n a_{ij} x_i \right)^{N-|S_\ell|} \\ &\quad + \sum_{\ell(|S_\ell|<s)} c_\ell \sum_{j=1}^m w_j \prod_{h \in S_\ell} a_{hj} \left(\sum_{i=1}^n a_{ij} x_i \right)^{(N+|S_\ell|-s)-|S_\ell|}. \end{aligned}$$

We can use (5) to simplify the first term and (6) (with $k = N + |S_\ell| - s$) to simplify the second term, giving us:

$$\begin{aligned} 0 &= \sum_{\ell(|S_\ell|=s)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_N \cdot N!} \cdot \prod_{h \notin S_\ell} x_h + \sum_{\ell(|S_\ell|<s)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_k \cdot k!} \cdot 0 \\ &= \sum_{\ell(|S_\ell|=s)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_N \cdot N!} \cdot \prod_{h \notin S_\ell} x_h. \end{aligned}$$

Since the monomials $\prod_{x_i \in S_\ell} x_i$ are linearly independent, this contradicts our assumption that the c_ℓ are nonzero. We conclude that \mathbf{A} has full row rank, and therefore that $m_1(p, \sigma) = m \geq \prod_{i=1}^n (r_i + 1)$. This completes the proof of equation (1).

For equation (2), it follows from Proposition II.6 part (2) below that for each i , we can approximate $x_i^{r_i}$ using

$7\lceil \log_2(r_i) \rceil$ neurons arranged in a deep network. Therefore, we can approximate all of the $x_i^{r_i}$ using a total of $\sum_i 7\lceil \log_2(r_i) \rceil$ neurons. From [15], we know that these n terms can be multiplied using $4n$ additional neurons, giving us a total of $\sum_i (7\lceil \log_2(r_i) \rceil + 4)$. This completes the proof. \square

In order to approximate a degree- N polynomial effectively with a single hidden layer, we must naturally assume that σ has nonzero N th Taylor coefficient. However, if we do not wish to make assumptions about the other Taylor coefficients of σ , we can still prove the following weaker result:

Theorem II.2. *Once again, let $p(\mathbf{x})$ denote the monomial $x_1^{r_1} x_2^{r_2} \dots x_n^{r_n}$. Then, for $N = \sum_{i=1}^n r_i$, we have $m_1(p, \sigma) \leq \prod_{i=1}^n (r_i + 1)$ if σ has nonzero N th Taylor coefficient. For any σ , we have that $m_1(p, \sigma)$ is at least the maximum coefficient in the univariate polynomial $g(y) = \prod_i (1 + y + \dots + y^{r_i})$.*

Proof outline. The first statement follows as in the proof of Theorem II.1. For the second, consider once again the equation (5). The left-hand side can be written, for any S , as the linear combination of basis functions of the form:

$$\left(\sum_{i=1}^n a_{ij} x_i \right)^{N-|S|}. \quad (7)$$

Letting S vary over multisets of fixed size s , we see that the right-hand side of (5) attains every degree- s monomial that divides $p(x)$. The number of such monomials is the coefficient C_s of the term y^s in the polynomial $g(y)$. Since such monomials are linearly independent, we conclude that the number of basis functions of the form (7) above must be at least C_s . Picking s to maximize C_s gives us the desired result. \square

Corollary II.3. *If $p(x)$ is the product of the inputs (i.e. $r_1 = \dots = r_n = 1$), we conclude that $m_1(p, \sigma) \geq \binom{n}{\lfloor n/2 \rfloor} \sim 2^n / \sqrt{\pi n/2}$, under the assumption that the n th Taylor coefficient of σ is nonzero.*

It is natural now to consider the cost of approximating general polynomials. However, without further constraint, this is relatively uninformative because polynomials of degree d in n variables live within a space of dimension $\binom{n+d}{d}$, and therefore most require exponentially many neurons for *any* depth of network. We therefore consider polynomials of *sparsity* c : that is, those that can be represented as the sum of c monomials. This includes many natural functions.

The following theorem, when combined with Theorem II.1, shows that general polynomials p with subexponential sparsity have exponential large $m_1(p, \sigma)$, but subexponential $m(p, \sigma)$.

Theorem II.4. Suppose that $p(\mathbf{x})$ is a multivariate polynomial of degree N , with c monomials $q_1(\mathbf{x}), q_2(\mathbf{x}), \dots, q_c(\mathbf{x})$. Suppose that the nonlinearity $\sigma(x)$ has nonzero Taylor coefficients up to x^N . Then,

1. $m_1(p, \sigma) \geq \frac{1}{c} \max_j m_1(q_j, \sigma)$.
2. $m(p, \sigma) \leq \sum_j m(q_j, \sigma)$.

Proof outline. Our proof in Theorem II.1 relied upon the fact that all nonzero partial derivatives of a monomial are linearly independent. This fact is not true for general polynomials p ; however, an exactly similar argument shows that $m_1(p, \sigma)$ is at least the number of linearly independent partial derivatives of p , taken with respect to multisets of the input variables.

Consider the monomial q of p such that $m_1(q, \sigma)$ is maximized, and suppose that $q(\mathbf{x}) = x_1^{r_1} x_2^{r_2} \dots x_n^{r_n}$. By Theorem II.1, $m_1(q, \sigma)$ is equal to the number $\prod_{i=1}^n (r_i + 1)$ of distinct monomials that can be obtained by taking partial derivatives of q . Let Q be the set of such monomials, and let D be the set of (iterated) partial derivatives corresponding to them, so that for $d \in D$, we have $d(q) \in Q$.

Consider the set of polynomials $P = \{d(p) \mid d \in D\}$. We claim that there exists a linearly independent subset of P with size at least $|D|/c$. Suppose to the contrary that P' is a maximal linearly independent subset of P with $|P'| < |D|/c$.

Since p has c monomials, every element of P has at most c monomials. Therefore, the total number of distinct monomials in elements of P' is less than $|D|$. However, there are at least $|D|$ distinct monomials contained in elements of P , since for $d \in D$, the polynomial $d(p)$ contains the monomial $d(q)$, and by definition all $d(q)$ are distinct as d varies. We conclude that there is some polynomial $p' \in P \setminus P'$ containing a monomial that does not appear in any element of P' . But then p' is linearly independent of P' , a contradiction since we assumed that P' was maximal.

We conclude that some linearly independent subset of P has size at least $|D|/c$, and therefore that the space of partial derivatives of p has rank at least $|D|/c = m_1(q, \sigma)/c$. This proves part (1) of the theorem. Part (2) follows immediately from the definition of $m(p, \sigma)$. \square

C. Univariate polynomials

As with multivariate polynomials, depth can offer an exponential savings when approximating univariate polynomials. We show below (Proposition II.5) that a shallow network can approximate any degree- d univariate polynomial with a number of neurons at most linear in d . The monomial x^d requires $d+1$ neurons in a shallow network (Proposition II.6), but can be approximated with

only logarithmically many neurons in a deep network. Thus, depth allows us to reduce networks from linear to logarithmic size, while for multivariate polynomials the gap was between exponential and linear. The difference here arises because the dimensionality of the space of univariate degree- d polynomials is linear in d , which the dimensionality of the space of multivariate degree- d polynomials is exponential in d .

Proposition II.5. Let σ be a nonlinear function with nonzero Taylor coefficients. Then, we have $m_1(p, \sigma) \leq d+1$ for every univariate polynomial p of degree d .

Proof. Pick a_0, a_1, \dots, a_d to be arbitrary, distinct real numbers. Consider the Vandermonde matrix \mathbf{A} with entries $A_{ij} = a_i^j$. It is well-known that $\det(\mathbf{A}) = \prod_{i < i'} (a_{i'} - a_i) \neq 0$. Hence, \mathbf{A} is invertible, which means that multiplying its columns by nonzero values gives another invertible matrix. Suppose that we multiply the j th column of \mathbf{A} by σ_j to get \mathbf{A}' , where $\sigma(x) = \sum_j \sigma_j x^j$ is the Taylor expansion of $\sigma(x)$.

Now, observe that the i th row of \mathbf{A}' is exactly the coefficients of $\sigma(a_i x)$, up to the degree- d term. Since \mathbf{A}' is invertible, the rows must be linearly independent, so the polynomials $\sigma(a_i x)$, restricted to terms of degree at most d , must themselves be linearly independent. Since the space of degree- d univariate polynomials is $(d+1)$ -dimensional, these $d+1$ linearly independent polynomials must span the space. Hence, $m_1(p, \sigma) \leq d+1$ for any univariate degree- d polynomial p . In fact, we can fix the weights from the input neuron to the hidden layer (to be a_0, a_1, \dots, a_d , respectively) and still represent any polynomial p with $d+1$ hidden neurons. \square

Proposition II.6. Let $p(x) = x^d$, and suppose that $\sigma(x)$ is a nonlinear function with nonzero Taylor coefficients. Then:

1. $m_1(p, \sigma) = d+1$.
2. $m(x^d, \sigma) \leq 7 \lceil \log_2(d) \rceil$.

Proof. Part (1) follows from part (1) of Theorem II.1 above, by setting $n = 1$ and $r_1 = d$.

For part (2), observe that we can approximate the square x^2 of an input x with three neurons in a single layer:

$$\frac{1}{2\sigma''(0)} (\sigma(x) + \sigma(-x) - 2\sigma(0)) = x^2 + \mathcal{O}(x^4).$$

We refer to this construction as a *square gate*, and the construction of [15] as a *product gate*. We also use *identity gate* to refer to a neuron that simply preserves the input of a neuron from the preceding layer (this is equivalent to the *skip connections* in residual nets).

Consider a network in which each layer contains a square gate (3 neurons) and either a product gate or an identity gate (4 or 1 neurons, respectively), according to the

following construction: The square gate squares the output of the preceding square gate, yielding inductively a result of the form x^{2^k} , where k is the depth of the layer. Writing d in binary, we use a product gate if there is a 1 in the 2^{k-1} -place; if so, the product gate multiplies the output of the preceding product gate by the output of the preceding square gate. If there is a 0 in the 2^{k-1} -place, we use an identity gate instead of a product gate. Thus, each layer computes x^{2^k} and multiplies $x^{2^{k-1}}$ to the computation if the 2^{k-1} -place in d is 1. The process stops when the product gate outputs u^d .

This network clearly uses at most $7\lceil\log_2(d)\rceil$ neurons, with a worst case scenario where $d+1$ is a power of 2. \square

D. Tensor decomposition

We conclude this section by noting interesting connections between the value $m_1(p, \sigma)$ and the established hard problem of tensor decomposition. Specifically, we show (Proposition II.7) that the minimum number of neurons required to approximate a polynomial p equals the symmetric tensor rank of a tensor constructed from the coefficients of p .

Let \mathbf{T} be an order- d symmetric tensor of dimensions $n \times \dots \times n$. We say that \mathbf{T} is *symmetric* if the entry $T_{i_1 i_2 \dots i_d}$ is identical for any permutation of i_1, i_2, \dots, i_d . For \mathbf{T} symmetric, the *symmetric tensor rank* $R_S(\mathbf{T})$ is defined to be the minimum r such that \mathbf{T} can be written

$$\mathbf{T} = \sum_{i=1}^r \lambda_i \mathbf{a}_i \otimes \dots \otimes \mathbf{a}_i,$$

with $\lambda \in \mathbb{R}$ and $\mathbf{a}_i \in \mathbb{R}^n$ for each i [16]. Thus, for $d = 1$, \mathbf{T} is simply a real number and $R_S(\mathbf{T}) = 1$. For $d = 2$, \mathbf{T} is a symmetric matrix, and the symmetric tensor rank is simply the rank of \mathbf{T} , where one can take the values λ_i above to be eigenvalues of the matrix \mathbf{T} .

For a multiset S , we let $\pi(S)$ denote the number of distinct permutations of S . Thus, if q_1, \dots, q_s are frequencies for elements in S , we have

$$\pi(S) = \frac{q_1 + \dots + q_s}{q_1, \dots, q_s}.$$

For $p(\mathbf{u}) = p(u_1, \dots, u_n)$ a homogeneous multivariate polynomial of degree d , we define the *monomial tensor* $\mathbf{T}(p)$ as follows:

$$\mathbf{T}(p)_{j_1 \dots j_d} = [u_{j_1} \dots u_{j_d}]_p / \pi(\{j_1, \dots, j_d\}),$$

where $[\bullet]_p$ denotes the coefficient of a monomial in p .

Proposition II.7. *For $p(\mathbf{u}) = p(u_1, \dots, u_n)$ a homogeneous multivariate polynomial of degree d , we have $m_1(p) \geq R_S(\mathbf{T}(p))$.*

Proof. Suppose that we can approximate $p(\mathbf{u})$ using a neural net with m neurons in a single hidden layer. Then, there exist vectors $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m$ of weights from the input to the hidden layer such that

$$p(\mathbf{u}) \approx w_1 \sigma(\mathbf{a}_1 \cdot \mathbf{u}) + \dots + w_m \sigma(\mathbf{a}_m \cdot \mathbf{u}).$$

If we consider only terms of degree d , we obtain

$$p(\mathbf{u}) = \sigma_d \sum_{i=1}^m w_i (\mathbf{a}_i \cdot \mathbf{u})^d \quad (8)$$

The symmetric tensor

$$\mathbf{T} = \sum_{i=1}^m (\sigma_d w_i) \mathbf{a}_i \otimes \dots \otimes \mathbf{a}_i$$

has symmetric tensor rank at most m . Furthermore, by equation (8), we have

$$p(\mathbf{u}) = \sum_{1 \leq j_1, \dots, j_d \leq m} T_{j_1 \dots j_d} (u_{j_1} \dots u_{j_d}).$$

By the definition of \mathbf{T} , the entries $T_{j_1 \dots j_d}$ are equal up to permutation of j_1, \dots, j_d . Therefore, \mathbf{T} must equal the monomial tensor $\mathbf{T}(p)$, showing that $m \geq R_S(\mathbf{T}(p))$ as desired. \square

Corollary II.8. *Let $p(\mathbf{u}) = p(u_1, \dots, u_n)$ be a multivariate polynomial of degree d . For $c = 0, 1, \dots, d$, let $p_c(\mathbf{u})$ be the homogeneous polynomial obtained by taking all terms of $p(\mathbf{u})$ with degree c . If R_S is the maximum symmetric rank of $\mathbf{T}(p_c(\mathbf{u}))$, then $m_1(p) \geq R_S$.*

The proof of this statement closely follows that of Proposition II.7.

Various results are known for the symmetric rank of tensors over the complex numbers \mathbb{C} [16]. Notably, Alexander and Hirschowitz [17] showed in a highly non-trivial proof that the symmetric rank of a generic symmetric tensor of dimension n and order k over \mathbb{C} equals $\lceil \binom{n+k-1}{k} / n \rceil$, with the exception of a few small values of k and n . However, this result does not hold over the real numbers \mathbb{R} , and in fact there can be *several* possible generic ranks for symmetric tensors over \mathbb{R} [16].

III. HOW EFFICIENCY IMPROVES WITH DEPTH

We now consider how $m_k(p, \sigma)$ scales with k , interpolating between exponential in n (for $k = 1$) and linear in n (for $k = \log n$). In practice, networks with modest $k > 1$ are effective at representing natural functions. We explain this theoretically by showing that the cost of approximating the product polynomial drops off rapidly as k increases.

A. Networks of constant depth

By repeated application of the shallow network construction in Lin, Tegmark, and Rolnick [15], we obtain the following *upper* bound on $m_k(p, \sigma)$, which we conjecture to be essentially tight. Our approach is reminiscent of tree-like network architectures discussed e.g. in [10], in which groups of input variables are recursively processed in successive layers.

Theorem III.1. *For $p(\mathbf{x})$ equal to the product $x_1 x_2 \cdots x_n$, and for any σ with all nonzero Taylor coefficients, we have:*

$$m_k(p, \sigma) = \mathcal{O}\left(n^{(k-1)/k} \cdot 2^{n^{1/k}}\right). \quad (9)$$

Proof. We construct a network in which groups of the n inputs are recursively multiplied. The n inputs are first divided into groups of size b_1 , and each group is multiplied in the first hidden layer using 2^{b_1} neurons (as described in [15]). Thus, the first hidden layer includes a total of $2^{b_1} n / b_1$ neurons. This gives us n / b_1 values to multiply, which are in turn divided into groups of size b_2 . Each group is multiplied in the second hidden layer using 2^{b_2} neurons. Thus, the second hidden layer includes a total of $2^{b_2} n / (b_1 b_2)$ neurons.

We continue in this fashion for b_1, b_2, \dots, b_k such that $b_1 b_2 \cdots b_k = n$, giving us one neuron which is the product of all of our inputs. By considering the total number of neurons used, we conclude

$$m_k(p, \sigma) \leq \sum_{i=1}^k \frac{n}{\prod_{j=1}^i b_j} 2^{b_i} = \sum_{i=1}^k \left(\prod_{j=i+1}^k b_j \right) 2^{b_i}. \quad (10)$$

Setting $b_i = n^{1/k}$, for each i , gives us the desired bound (9). \square

In fact, we can solve for the choice of b_i such that the upper bound in (10) is minimized, under the condition $b_1 b_2 \cdots b_k = n$. Using the technique of Lagrange multipliers, we know that the optimum occurs at a minimum of the function

$$\mathcal{L}(b_i, \lambda) := \left(n - \prod_{i=1}^k b_i \right) \lambda + \sum_{i=1}^k \left(\prod_{j=i+1}^k b_j \right) 2^{b_i}.$$

Differentiating \mathcal{L} with respect to b_i , we obtain the con-

ditions

$$0 = -\lambda \prod_{j \neq i} b_j + \sum_{h=1}^{i-1} \left(\frac{\prod_{j=h+1}^k b_j}{b_i} \right) 2^{b_h} + (\log 2) \left(\prod_{j=i+1}^k b_j \right) 2^{b_i}, \text{ for } 1 \leq i \leq k \quad (11)$$

$$0 = n - \prod_{j=1}^k b_j. \quad (12)$$

Dividing (11) by $\prod_{j=i+1}^k b_j$ and rearranging gives us the recursion

$$b_i = b_{i-1} + \log_2(b_{i-1} - 1 / \log 2). \quad (13)$$

Thus, the optimal b_i are not exactly equal but very slowly increasing with i (see Figure 1).

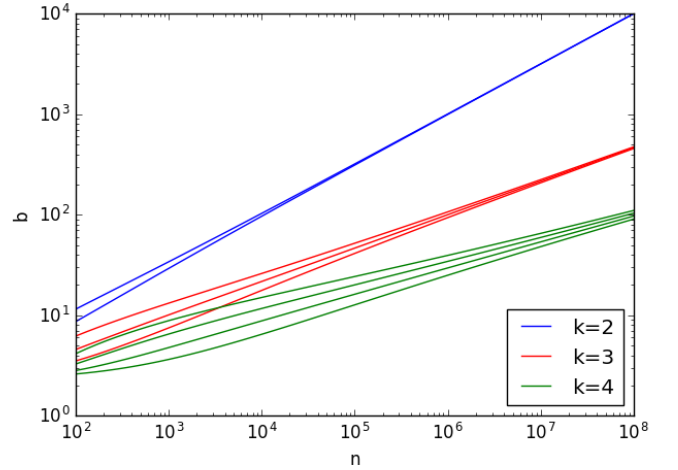


FIG. 1: The optimal settings for $\{b_i\}_{i=1}^k$ as n varies are shown for $k = 1, 2, 3$. Observe that the b_i converge to $n^{1/k}$ for large n , as witnessed by a linear fit in the log-log plot. The exact values are given by equations (12) and (13).

The following conjecture states that the bound given in Theorem III.1 is (approximately) optimal.

Conjecture III.2. *For $p(\mathbf{x})$ equal to the product $x_1 x_2 \cdots x_n$, and for any σ with all nonzero Taylor coefficients, we have*

$$m_k(p, \sigma) = 2^{\Theta(n^{1/k})}, \quad (14)$$

i.e., the exponent grows as $n^{1/k}$ as $n \rightarrow \infty$.

We empirically tested Conjecture III.2 by training ANNs to predict the product of input values x_1, \dots, x_n with

$n = 20$ (see Figure 2). The rapid interpolation from exponential to linear width aligns with our predictions.

In our experiments, we used feedforward networks with dense connections between successive layers, with nonlinearities instantiated as the hyperbolic tangent function. Similar results were also obtained for rectified linear units (ReLU) as the nonlinearity, despite the fact that this function does not satisfy our hypothesis of being everywhere differentiable. The number of layers was varied, as was the number of neurons within a single layer. The networks were trained using the AdaDelta optimizer [18] to minimize the absolute value of the difference between the predicted and actual values. Input variables x_i were drawn uniformly at random from the interval $[0, 2]$, so that the expected value of the output would be of manageable size.

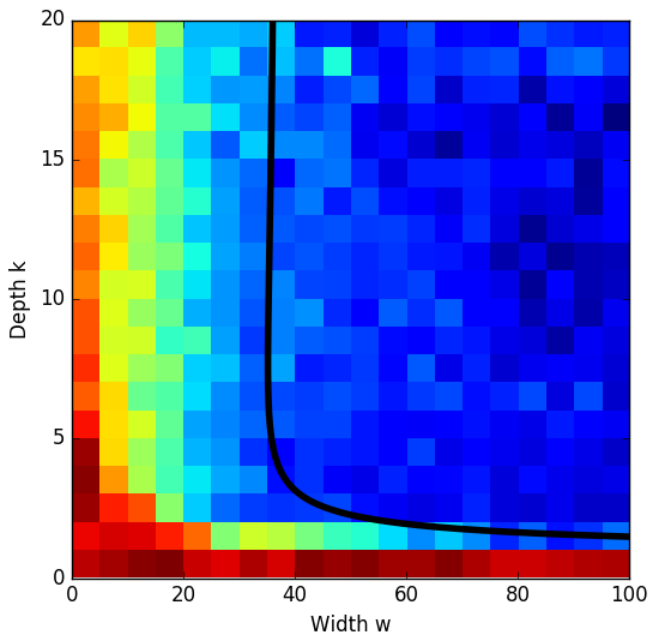


FIG. 2: Performance of trained networks in approximating the product of 20 input variables, ranging from red (high mean error) to blue (low mean error). The curve $w = n^{(k-1)/k} \cdot 2^{1/k}$ for $n = 20$ is shown in black. In the region above and to the right of the curve, it is possible to effectively approximate the product function (Theorem III.1).

Eq. (14) provides a helpful rule of thumb for how deep is deep enough. Suppose, for instance, that we wish to keep typical layers no wider than about a thousand ($\sim 2^{10}$) neurons. Eq. (14) then implies $n^{1/k} \lesssim 10$, *i.e.*, that the number of layers should be at least

$$k \gtrsim \log_{10} n.$$

B. Circuit complexity

It is interesting to consider how our results on the inapproximability of simple polynomials by polynomial-size neural networks compare to results for Boolean circuits. Recall that TC^0 is defined as the set of problems that can be solved by a Boolean circuit of constant depth and polynomial size, where the circuit is allowed to use AND, OR, NOT, and MAJORITY gates of arbitrary fan-in.² It is an open problem whether TC^0 equals the class TC^1 of problems solvable by circuits for which the depth is logarithmic in the size of the input.

In this section, we consider the feasibility of strong general no-flattening results. It would be very interesting if one could show that general polynomials p in n variables require a superpolynomial number of neurons to approximate for any constant number of hidden layers. That is, for each integer $k \geq 1$, we would like to prove a lower bound on $m_k(p, \sigma)$ that grows fast than polynomially in n .

Such a result might seem to address questions such as whether TC^0 and TC^1 are equal. However, Boolean circuits compute using 0/1 values, while the neurons of our artificial neural networks take on arbitrary real values. To preserve 0/1 values at all neurons, we can restrict inputs to such values and take the nonlinear activation to be the Heaviside step function:

$$\sigma(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases}$$

This gives us essentially a multi-layered perceptron, as inspired by McCulloch and Pitts [19].

We assume also that each neuron has access to a fixed bias constant: that is, a neuron receiving inputs x_1, x_2, \dots, x_n is of the form $\sigma(a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n)$ where $a_0, a_1, a_2, \dots, a_n$ are real constants. Such a neuron corresponds to a *weighted threshold gate* in Boolean circuits.

It follows from the work of [20] and [21] that such artificial neural nets (ANNs), with constant depth and polynomial size, have exactly the same power as TC^0 circuits. That is, weighted threshold gates can simulate and be simulated by constant-depth, polynomial-size circuits of AND, OR, NOT, and MAJORITY gates. The following are simple constructions for these four types of gates using weighted

² An AND gate evaluates **True** if *all* its inputs do so, an OR gate evaluates **True** if *any* of its inputs do so, a NOT gate evaluates **True** if its input is **False** and vice versa, and a MAJORITY gate evaluates **True** if *at least half* of its inputs do so.

thresholds.

$$\begin{aligned}\text{AND}(x_1, x_2, \dots, x_n) &= \sigma \left(\sum_{i=1}^n x_i - \left(n - \frac{1}{2} \right) \right) \\ \text{OR}(x_1, x_2, \dots, x_n) &= \sigma \left(\sum_{i=1}^n x_i - \frac{1}{2} \right) \\ \text{NOT}(x) &= \sigma \left(\frac{1}{2} - x \right) \\ \text{MAJORITY}(x_1, x_2, \dots, x_n) &= \sigma \left(\sum_{i=1}^n x_i - \frac{n-1}{2} \right)\end{aligned}$$

Thus, we should not hope easily to prove general no-flattening results for Boolean functions, but the case of polynomials in real-valued variables may be more tractable. Simply approximating a real value by a Boolean circuit requires arbitrarily many bits. Therefore, performing direct computations on real values is clearly intractable for TC^0 circuits. Moreover, related work such as [4, 8, 13] has already proven gaps in expressivity for real-valued neural networks of different depths, for which the analogous results remain unknown in Boolean circuits.

IV. CONCLUSION

We have shown how the power of deeper ANNs can be quantified even for simple polynomials. We have proven that there is an exponential gap between the width of shallow and deep networks required for approximating a given sparse polynomial. For n variables, a shallow network requires size exponential in n , while a deep network requires at most linearly many neurons. Networks with a constant number $k > 1$ of hidden layers appear to interpolate between these extremes, following a curve exponential in $n^{1/k}$. This suggests a rough heuristic for the number of layers required for approximating simple functions with neural networks. For example, if we want no layers to have more than 10^3 neurons, say, then the minimum number of layers required grows only as $\log_{10} n$.

It is worth noting that our constructions enjoy the property of *locality* mentioned in [13], which is also a feature

of convolutional neural nets. That is, each neuron in a layer is assumed to be connected only to a small subset of neurons from the previous layer, rather than the entirety of them (or some large fraction). In fact, we showed (e.g. Prop. II.6) that there exist natural functions that can be computed in a linear number of neurons, where each neuron is connected to at most *two* neurons in the preceding layer, which nonetheless cannot be computed with fewer than exponentially many neurons in a single layer, no matter how many connections are used. Our construction can also easily be framed with reference to the other properties mentioned in [13]: those of *sharing* (in which weights are shared between neural connections) and *pooling* (in which layers are gradually collapsed, as our construction essentially does with recursive combination of inputs).

This paper has focused exclusively on the resources (notably neurons and synapses) required to *compute* a given function. An important complementary challenge is to quantify the resources (e.g. training steps) required to *learn* the computation, i.e., to converge to appropriate weights using training data — possibly a fixed amount thereof, as suggested in [22]. There are simple functions that can be computed with polynomial resources but require exponential resources to learn [23]. It is quite possible that architectures we have not considered increase the feasibility of learning. For example, residual networks (ResNets) [24] and unitary nets (see e.g. [25, 26]) are no more powerful in representational ability than conventional networks of the same size, but by being less susceptible to the “vanishing/exploding gradient” problem, it is far easier to optimize them in practice. We look forward to future work that will help us understand the power of neural networks to learn.

V. ACKNOWLEDGMENTS

This work was supported by the Foundational Questions Institute <http://fqxi.org/>, the Rothberg Family Fund for Cognitive Science and NSF grant 1122374. We thank Scott Aaronson, Surya Ganguli, David Budden, and Henry Lin for helpful discussions and suggestions.

-
- [1] G. Cybenko, Mathematics of Control, Signals, and Systems (MCSS) **2**, 303 (1989).
 - [2] K. Hornik, M. Stinchcombe, and H. White, Neural networks **2**, 359 (1989).
 - [3] K.-I. Funahashi, Neural networks **2**, 183 (1989).
 - [4] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, in *Advances in Neural Information Processing Systems (NIPS)* (2014), pp. 2924–2932.
 - [5] M. Bianchini and F. Scarselli, IEEE transactions on neural networks and learning systems **25**, 1553 (2014).
 - [6] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, in *Advances In Neural Information Processing Systems (NIPS)* (2016), pp. 3360–3368.
 - [7] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, arXiv preprint arXiv:1611.08083 (2016).
 - [8] M. Telgarsky, Journal of Machine Learning Research (JMLR) **49** (2016).

- [9] R. Eldan and O. Shamir, in *29th Annual Conference on Learning Theory* (2016), pp. 907–940.
- [10] H. Mhaskar, Q. Liao, and T. Poggio, arXiv preprint arXiv:1603.00988v4 (2016).
- [11] O. Delalleau and Y. Bengio, in *Advances in Neural Information Processing Systems (NIPS)* (2011), pp. 666–674.
- [12] J. Martens, A. Chattopadhyaya, T. Pitassi, and R. Zemel, in *Advances in Neural Information Processing Systems (NIPS)* (2013), pp. 2877–2885.
- [13] N. Cohen, O. Sharir, and A. Shashua, *Journal of Machine Learning Research (JMLR)* **49** (2016).
- [14] N. Cohen and A. Shashua, in *International Conference on Machine Learning (ICML)* (2016).
- [15] H. W. Lin, M. Tegmark, and D. Rolnick, arXiv preprint arXiv:1608.08225v3 (2016).
- [16] P. Comon, G. Golub, L.-H. Lim, and B. Mourrain, *SIAM Journal on Matrix Analysis and Applications* **30**, 1254 (2008).
- [17] J. Alexander and A. Hirschowitz, *Journal of Algebraic Geometry* **4**, 201 (1995).
- [18] M. D. Zeiler, arXiv preprint arXiv:1212.5701 (2012).
- [19] W. S. McCulloch and W. Pitts, *The bulletin of mathematical biophysics* **5**, 115 (1943).
- [20] A. K. Chandra, L. Stockmeyer, and U. Vishkin, *SIAM Journal on Computing* **13**, 423 (1984).
- [21] N. Pippenger, *IBM journal of research and development* **31**, 235 (1987).
- [22] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, in *Proceedings of the International Conference on Learning Representations (ICLR)* (2017).
- [23] S. Shalev-Shwartz, O. Shamir, and S. Shammah, arXiv preprint arXiv:1703.07950 (2017).
- [24] K. He, X. Zhang, S. Ren, and J. Sun, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.
- [25] M. Arjovsky, A. Shah, and Y. Bengio, in *International Conference on Machine Learning* (2016), pp. 1120–1128.
- [26] L. Jing, Y. Shen, T. Dubček, J. Peurifoy, S. Skirlo, M. Tegmark, and M. Soljačić, arXiv preprint arXiv:1612.05231 (2016).