THE EXPRESSIVE POWER OF NEURAL NETWORKS
CPSC 490
MAY 3, 2018

David Brandfonbrener
Supervised by Andrew Barron and Dana Angluin

1. Introduction

Over the last ten years there has been a resurgence in the popularity of neural networks in machine learning. With better hardware, people have been able to implement increasingly complicated networks and achieve impressive results on tasks like image recognition and playing board games. The models are clearly flexible and powerful function approximators even though they are assembled from individually simple parts, namely linear transformations and simple componentwise nonlinear functions. Despite all this progress, the class of functions that can be easily represented by neural networks is still not well understood [14]. This project is an attempt to assess the current understanding of the function approximation capabilities of neural networks, to point towards the gaps in understanding, and to search for some theory that may be able to help fill these gaps. A better understanding of these capabilities could lead to better performance on real problems by formally understanding which network architectures can yield better performance.

The paper is organized as follows. Section 2 presents some preliminary definitions. Section 3 provides an in-depth review of the literature on function approximation by neural networks. Specific attention is given to universal approximation for fixed depth and fixed width networks. Then we examine the relationship between depth and expressivity in terms of depth gaps where deeper networks can represent certain functions more efficiently and an assortment of function complexity measures. Section 4 highlights some open questions in the field. Sections 5 presents techniques and a few preliminary results motivated by the open questions. Section 6 concludes.

1.1. **Acknowledgements.** I would like to thank Professors Andrew Barron and Dana Angluin for advising this project. They have both been extremely insightful and patient throughout the entire process. Moreover, I would like to thank all of the computer science and math professors I have had at Yale. Their teaching has given me the tools to be able to understand this often complicated topic and prepared me to not only complete my degree, but to leave Yale ready to begin conducting my own research.

2. Preliminaries

In this section, we briefly describe two notions of the feedforward neural network model and a few other useful definitions.

Definition 2.1. (Standard network) Let $\mathcal{A}$ be our network, so that the network defines a function $f_{\mathcal{A}} : \mathbb{R}^n \to \mathbb{R}$. The *network* consists of l hidden *layers* each of which defined by an affine linear transformation $T_i : \mathbb{R}^{w_{i-1}} \to \mathbb{R}^{w_i}$ for $i = 1, 2, \ldots, l+1$ and $w_0 = n, w_{l+1} = 1$. Then l is the *depth* of the network and the w_i are referred to as the *widths* of the layers. Let $w = \max\{w_i \mid 0 \leq i \leq l+1\}$ be the width of $\mathcal{A}$. Define a nonlinear *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$, which will often be the *ReLU* function $\sigma(x) = \max(0, x)$. For $x \in \mathbb{R}^k$ for any k we will overload notation and also define $\sigma : \mathbb{R}^k \to \mathbb{R}^k$ by applying σ to x component wise. Then, we can finally describe the function as:

$$f_{\mathcal{A}}(x) = T_{l+1} \circ \sigma \circ T_l \circ \ldots T_2 \circ \sigma \circ T_1(x).$$

One alternative way to think about this kind of network is to envision a directed graph from the inputs to the outputs where each component of a hidden layer is a node and each element of the matrix representing T_i is an edge. The following definition found in Bartlett et al. in [4] gives a more general notion of a network based on the graph representation. This definition includes networks used in practice, like convolutional networks, and if we let the graph contain cycles can even describe recurrent networks.

Definition 2.2. (General network) A network $\mathcal{A}$ is defined by a directed acyclic graph $G_{\mathcal{A}}$, an activation function σ, and a set of parameters of one weight for each edge and one bias for each node of $G_{\mathcal{A}}$. Then, if the longest path in the graph has length $l+1$ we say that the network has l layers. Then, the input layer (layer 0) is defined to be the set of nodes with in-degree 0. To create a function from $\mathbb{R}^n \to \mathbb{R}$ we assume that there are n such input nodes and that there is only one sink node, the unique node in layer l. Then for $1 \leq i \leq l$ we define layer i to be the set of nodes with a predecessor in layer $i-1$ and no predecessor in any layer $j \geq i$. Then, computation of the function $f_{\mathcal{A}}$ proceeds by computing the output of each subsequent layer i. For each node u in layer i, take the vector of outputs of units v with directed edges to u along with the vector w of weights associated to those edges and the bias b at node u. Then u outputs $\sigma(w^T x + b)$. The only exception is that the output node in layer l does not apply σ and rather outputs $w^T x + b$.

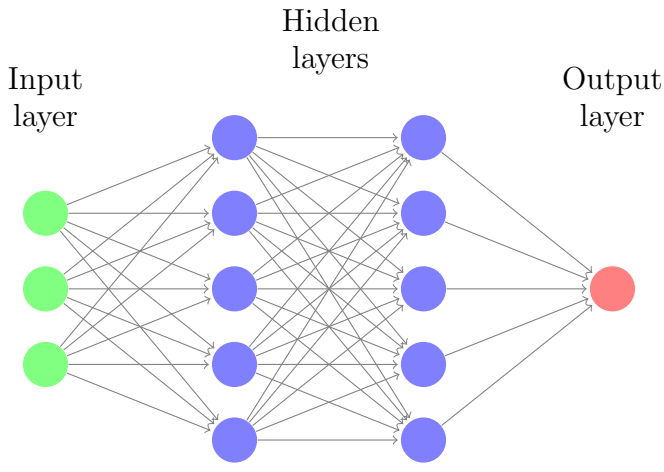

FIGURE 1
An example of a graph representing a neural network mapping $\mathbb{R}^3 \to \mathbb{R}$. Here we would have $l = 2$, $w = 5$.

Throughout this report we will use both the standard definition of the network as a composition of affine transformations as well as the graph definition. Unless specifically indicated, we will use the standard network architecture. However, it is sometimes useful to use the graph representation of that standard network in the analysis.

Now, we can define the main object of study in this report, the class of functions that are computable by a neural network of given architecture.

Definition 2.3. (Function class) Using the standard architecture, for a given depth l and width w define $\mathcal{F}_{l,w}$ to be the class of all functions represented by networks of l hidden layers, each of width w.

Since we care about how efficiently a network can represent a given function, it is useful to have a notion of network size.

Definition 2.4. (Network Size) Let the *size* of a network $\mathcal{A}$, denoted $|\mathcal{A}|$, be the total number of parameters, both weights and biases of the network. So, when considering the graph $G_{\mathcal{A}}$ this is the sum of the number of edges and number of nodes. Translating this to the standard model for a network with depth l and layers of widths w_i, this is $\sum_{i=1}^{l}(w_i w_{i-1} + w_i)$ since each layer is defined by a linear transformation that takes $w_i w_{i-1} + w_i$ weights to define since we can write $T_i x = Ax + b$ where $A \in \mathbb{R}^{w_{i-1} \times w_i}, b \in \mathbb{R}^{w_i}$. Note that when all of the widths are the same, the above definition of size just gives us $lw^2 + lw = O(lw^2)$. This is the case we will usually consider.

With these definitions in place, we can move on to summarizing the current state of the field of function approximation by neural networks.

3. Literature Review

This section provides a review of much of the literature on the relative approximation capabilities of networks of different sizes. We group the results into a few different types and explain the main theorems of each direction of research.

3.1. Universal Approximation.

The classical result about the approximation capabilities of neural networks is that of universal approximation. In the late 80's and early 90's a series of results by Barron, Cybenko, Funahashi, and Hornik demonstrated that a network with one hidden layer but an arbitrarily large width can approximate any reasonable function (integrable for example) to arbitrary precision (in L^1 or L^∞ norm for example) [3, 9, 13, 19]. In 2017, a new universal approximation result was proven by both Hanin and Lu et al. in [16, 23] showing that a deep network of fixed width $w = n + 2$ and arbitrary depth can also achieve arbitrary precision. This result shows that depth can also be used to achieve universal approximation, but does not prove any inherent advantage to depth instead of width. The result is proven as a corollary to the older work on networks with one hidden layer by constructing the same functions with different architecture. The technique relies on using a bounded input domain to guarantee that certain ReLU units are in the linear, positive regime. This construction is an important starting point for considering which classes of neural networks can represent which other classes. The full construction used to prove the following Theorem is presented below in the Lemma and follows Hanin [16].

Theorem 3.1. [16, Theorem 1] Let $\Omega \subseteq \mathbb{R}^n$ be compact. Then the set $\mathcal{R}$ of ReLU networks of width $w \leq n+2$ is dense in $C(\Omega)$, the continuous functions with L^∞ norm, and $\mathcal{R}$ is dense in $L^p(\Omega)$ for $1 \leq p < \infty$.

The Theorem follows directly from Lemma 3.2 and Hornik [19, Theorem 1].[1] Let $[\cdot]_+$ represent the ReLU function defined above.

Lemma 3.2. [16, Lemma 10] Let $\Omega \subset \mathbb{R}^n$ be compact and $f : \Omega \to \mathbb{R}$ be a function computed by a ReLU network with one hidden layer of width m. So, we have that $f(\vec{x}) = \sum_{j=1}^{m} c_j \left[\vec{a}_j \cdot \vec{x} + b_j\right]_+$, where for each j we have $\vec{a}_j \in \mathbb{R}^n$ and $b_j, c_j \in \mathbb{R}$. Then, there exists a ReLU network with $m+1$ hidden layers of width $n+2$ that computes f.

Proof. Since Ω is bounded we can find $T > 0$ such that $T + \sum_{j=1}^{k} c_j \left[\vec{a}_j \cdot \vec{x} + b_j\right]_+ > 0$ for all $k = 1, \ldots, m$ and $T + x_i > 0$ for all $i = 1, \ldots, d$. This value will allow us to avoid having inputs mapped to 0 by the repeated application of ReLU functions.

Now, note that for each j we can define $d_j \in \mathbb{R}$ such that $\vec{a}_j \cdot (\vec{x} + T\vec{1}) + d_j = \vec{a}_j \cdot \vec{x} + b_j$, just by distributing the dot product over the subtraction and letting $d_j = b_j - \vec{a}_j \cdot T\vec{1}$.

Let A_j be the function calculated by the j-th layer. So we have that $A_1 : \mathbb{R}^n \to \mathbb{R}^{n+2}$, $A_j : \mathbb{R}^{n+2} \to \mathbb{R}^{n+2}$ for $j = 1, \ldots, m+1$, and the output layer is $A_{m+2} : \mathbb{R}^{n+2} \to \mathbb{R}$. Take $\vec{x} \in \mathbb{R}^n$ and $y, z \in \mathbb{R}$, then:

$$\begin{aligned}
A_1(\vec{x}) &= \left[(\vec{x} + T\vec{1}, \vec{a}_1 \cdot \vec{x} + b_1, T)\right]_+ \\
A_2(\vec{x}, y, z) &= \left[(\vec{x}, \vec{a}_2 \cdot \vec{x} + d_2, z + c_1 y)\right]_+ \\
&\cdots \\
A_j(\vec{x}, y, z) &= \left[(\vec{x}, \vec{a}_j \cdot \vec{x} + d_j, z + c_{j-1} y)\right]_+ \\
&\cdots \\
A_{m+1}(\vec{x}, y, z) &= \left[(\vec{x}, 0, z + c_m y)\right]_+ \\
A_{m+2}(\vec{x}, y, z) &= z - T
\end{aligned}$$

Building a network by composing these layers gives us the desired result.

Note, we are using the first n components of each layer to store the input vector $\vec{x}$ (with some correction by T in case of negative components). Then we use the $n+1$-th component to calculate the ReLU of the j-th affine transformation and the $n+2$-th to calculate the m dimensional linear combination of ReLUs one term at a time (again with correction by T). In such a manner, we reconstruct the entirety of the original wide and shallow network that calculates the function f. □

Note that with Lemma 3.2, we can now translate any result about the approximation capability of ReLU networks with one hidden layer into a result about a ReLU network with fixed width $n+2$, as in the above Theorem.

[1] [19, Theorem 1] gives universal approximation for networks with one hidden layer and bounded, nonconstant activation functions. Since $[x]_+ - [x-1]_+$ is a bounded, nonconstant function representable by a linear combination of ReLU's applied to affine functions, the universal approximation result for bounded, nonconstant activation functions also extends to ReLU activation networks.

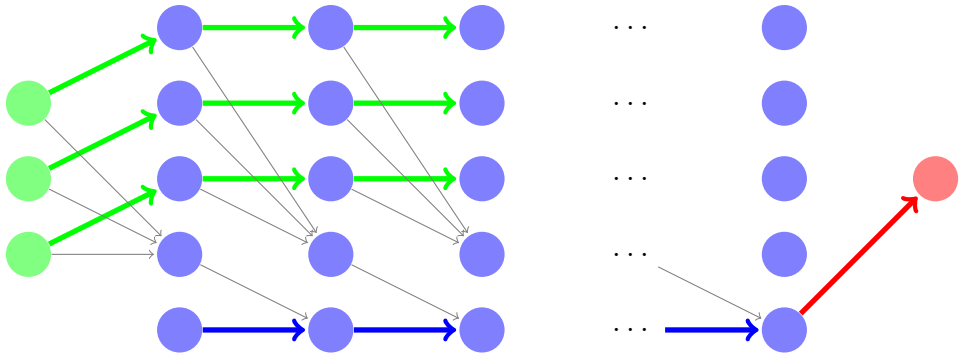

FIGURE 2
An illustration of the construction of Lemma 3.2 in terms of the network graphs. This construction transforms a network with one hidden layer into a network with many. The black edges represent the edges that would have been in the original network with one hidden layer. The green edges represent mapping the shifted inputs through the graph. The blue edges represent accumulating the shifted output. The red edge shifts the output back to the desired range.

These universal approximation results demonstrate how neural networks can be powerful. However, they say nothing in terms of the efficiency of the approximation determined by the size and architecture of the network needed to approximate a given function. Most of the research in this area is directed towards those questions of efficiency which are addressed in the following sections.

3.2. **Depth gaps.** Recent work has tried to prove that deep networks enjoy an exponential advantage in efficiency over their shallower counterparts. This means that there are certain functions which a deep architecture can represent that require a shallower network of size (number of parameters) exponential in the size of the deeper network. These results were initially demonstrated by Telgarsky in [33] in a constrained setting and then the ideas were generalized by Arora et al. and Telgarsky in [2, 34].

Specifically, [33] proves the following Theorem, which uses classification error as the measure of approximation capability. Classification error for a function $f : \mathbb{R} \to \mathbb{R}$ is defined over a set S of k points $(x_i, y_i) \in \mathbb{R} \times \{0, 1\}$ for $1 \leq i \leq k$ by $R_S(f) = k^{-1} \sum_i \mathbb{1}[y_i = \mathbb{1}[f(x_i) \geq 1/2]]$. This counts how often the function is at least $1/2$ away from the target function at the desired set of points. This result can be extended to one that uses L_1 or L_∞ norm instead of classification error with some added machinery [34], but we will only go through the proof of the simpler result here because it provides many useful insights into the understanding of the functions calculated by neural networks.

Theorem 3.3. [33, Theorem 1.1] There exists a network $\mathcal{A}$ of width 2 and depth $2h$ that computes a function $F_\mathcal{A} : \mathbb{R} \to \mathbb{R}$ and a set S of $k = 2^h$ points in $\mathbb{R} \times \{0, 1\}$ such that $R_S(F_\mathcal{A}) = 0$ and any network $\mathcal{B}$ of depth l and width w such that $w \leq 2^{(h-3)/l-1}$ has $R_S(F_\mathcal{B}) \geq 1/6$.

To prove this result, we first need to define a set S. So, let $S = \{(\frac{i}{k}, \mathbb{1}[i \; odd]) : 0 \leq i < k\}$, a set of uniformly spaced alternating labels in the unit interval. A function that hits all of the points in S must oscillate very quickly. The idea is to prove that shallow networks

cannot accomplish such oscillations and provide a construction of a deep network that can. First we need some lemmas.

Lemma 3.4. [33, Lemma 2.1] If the activation function σ is piecewise linear with t pieces, then a network $\mathcal{B}$ of depth l and width w with input dimension 1 calculates a function $F_{\mathcal{B}}$ that is piecewise linear with at most $(tw)^l$ pieces.

Proof. Take $f, g : \mathbb{R} \to \mathbb{R}$ piecewise linear with t_f and t_g pieces respectively. First we will show that $f + g$ has at most $t_f + t_g$ pieces and $f \circ g$ has at most $t_f t_g$ pieces. Let $\mathcal{I}_f, \mathcal{I}_g$ be the partitions of $\mathbb{R}$ corresponding to the pieces of the respective functions.

Now for any intervals $U_f \in \mathcal{I}_f$ and $U_g \in \mathcal{I}_g$, the function $f + g$ has a single slope on the interval $U_f \cap U_g$. So, $f+g$ has at most $|\mathcal{I}_f \cap \mathcal{I}_g|$ pieces, where the intersection of the partition is defined as the set of all piecewise intersections. Note that we can sort all of the intervals of $\mathcal{I}_f, \mathcal{I}_g$ by their left endpoint and that each endpoint defines at most one interval intersection. So, we have that the number of pieces is $|\mathcal{I}_f \cap \mathcal{I}_g| \leq t_f + t_g$.

Now take an interval $U_g \in \mathcal{I}_g$. Then the image $f(g(U_g))$ contains at most $|\mathcal{I}_f| = t_f$ pieces since g is linear on U_g so $g(U_g)$ is one continuous interval. Since U_g was arbitrary, we have that $f \circ g$ has at most $t_f t_g$ pieces.

Finally we can induct over the layers to show that $F_{\mathcal{B}}$ has at most $(tw)^l$ linear pieces. Specifically, the induction will be to show that the function of the input determined by the value at any node in layer i is piecewise linear with $(tw)^i$ pieces. Let the input be x. For the first layer, each node calculates $\sigma(a^T x + b)$ for some weight vector a and bias b. This function is piecewise linear with $t \leq (tw)^1$ pieces since the only nonlinearity is induced by the activation function. Assume inductively that each node at layer $i-1$ calculates a piecewise linear function g_j for $1 \leq j \leq w$ with at most $(tw)^{i-1}$ pieces. Then the function at a node in layer i is a combination of the g_j by $\sigma(\sum_j a_j g_j(x) + b)$ for some new weights a and bias b. By the above statements, this function will have at most $tw(tw)^{i-1} = (tw)^i$ pieces, proving the result. □

Lemma 3.5. [33, Lemma 2.2] Take $S = \{(\frac{i}{k}, \mathbb{1}[i \text{ odd}]) : 0 \leq i < k\}$, as defined above. Then for any piecewise linear $f : \mathbb{R} \to \mathbb{R}$ with at most t pieces, we have that $R_S(f) \geq \frac{k-4t}{3k}$.

Proof. Define $\tilde{f}(x) = \mathbb{1}[f(x) \geq 1/2]$. Then $R_S(f) = k^{-1} \sum_i \mathbb{1}[y_i \neq \tilde{f}(x_i)]$. Since f has at most t linear pieces, it is piecewise monotonic on each piece of a t-piece partition of $\mathbb{R}$, so f can cross $1/2$ at most $2t - 1$ times. That is 1 crossing for each interval, and one crossing for each right endpoint of an interval, except for the last one. So, $\tilde{f}$ is piecewise constant with at most $2t$ pieces. Then to place the k points into these $2t$ intervals, at least $k - 4t$ points must fall into intervals with at least 3 points by the pigeonhole principle. Now, since we defined S so that the labels are alternating 0 and 1, any interval with at least 3 points must misclassify at least $1/3$ of the points falling inside of it. Thus, we have that at least $(k - 4t)/3$ points are misclassified so the error is bounded below by $\frac{k-4t}{3k}$, as desired. □

Lemma 3.6. [33, Lemma 2.4] Let $S_j = \{(\frac{i}{2^j}, \mathbb{1}[i \text{ odd}]) : 0 \leq i < 2^j\}$, so that $S = S_h$. Then we can construct a ReLU network $\mathcal{A}_j$ of width 2 and depth j such that $F_{\mathcal{A}_j}(x_i) = y_i$ for all $(x_i, y_i) \in S_j$.

Proof. Define the "mirror" function

$$f_m(x) = \begin{cases} 2x & 0 \le x \le 1/2 \\ 2(1-x) & 1/2 < x \le 1 \\ 0 & \textit{otherwise.} \end{cases}$$

Let σ be the ReLU function $\sigma(x) = \max\{x, 0\}$. Then $f_m(x) = \sigma(2\sigma(x)+(-4)\sigma(x+(-1/2)))$, can easily be defined as a neural network of two hidden layers with width 2.

Now we want to induct over j to prove the following claim. For each $x \in [0,1]$ choose $i_j \in \{0, 1, \dots 2^{j-1}\}$ and $\bar{x}_j \in [0,1)$ such that $x = (i_j + \bar{x}_j)2^{1-j}$. Then

$$f_m^j(x) = \begin{cases} 2\bar{x}_j & 0 \le \bar{x}_j \le 1/2 \\ 2(1-\bar{x}_j) & 1/2 < \bar{x}_j < 1. \end{cases}$$

The base case of $j = 1$, was proven above by definition of f_m. Assume this holds for j we want to show it for $j+1$. Now note that for any function g, pre-composition $g \circ f_m$ calculates $g(2x)$ for $x \in [0, 1/2]$ and $g(2-2x)$ for $x \in (1/2, 1]$. In other words $g \circ f_m$ scales g horizontally by $1/2$ and reflects it about the vertical line $x = 1/2$. Using this reflection property and then the symmetry of f_m^j from the inductive hypothesis, we can note that for $x \in [0, 1/2]$:

$$(f_m^j \circ f_m)(x) = (f_m^j \circ f_m)(1-x) = (f_m^j \circ f_m)(x + 1/2).$$

So, we can consider the case where $x \in [0, 1/2]$ without loss of generality so that $(f_m^j \circ f_m)(x) = (f_m^j)(2x)$. Now in the $j+1$ case we have $i_{j+1}, \bar{x}_{j+1}$ with $i_{j+1} < 2^{j-1}$ since $x \le 1/2$ such that $2x = 2(i_{j+1} + \bar{x}_{j+1})2^{-j} = (i_{j+1} + \bar{x}_{j+1})2^{1-j}$. Applying the inductive hypothesis, we have that

$$f_m^{j+1}(x) = f_m^j(2x) = \begin{cases} 2\bar{x}_{j+1} & 0 \le \bar{x}_{j+1} \le 1/2 \\ 2(1-\bar{x}_{j+1}) & 1/2 < \bar{x}_{j+1} < 1. \end{cases}$$

We conclude the proof by setting $j = h$. Then at each of the 2^h values of $x_i = i/2^h$, we output 0 when i is even since then $x_i = (i/2 + 0)2^{1-h}$ and 1 when i is odd since $x_i = ((i-1)/2 + 1/2)2^{1-h}$. □

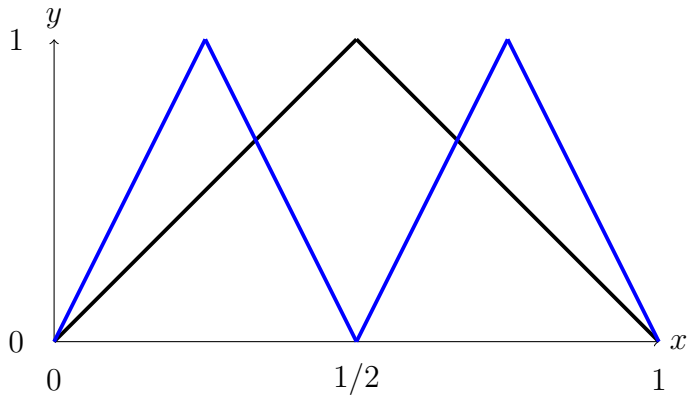

FIGURE 3
An illustration of the composition of the mirror function. In black, we plot f_m and in blue we plot f_m^2. Note that f_m^i will have 2^{i-1} oscillations.

Proof. (of Theorem 3.3) The construction of $\mathcal{A}$ follows immediately from Lemma 3.6.

To see that every network $\mathcal{B}$ must have error at least $1/6$, we combine Lemmas 3.4 and 3.5. Take l, w such that $w \leq 2^{(h-3)/l-1}$. Then $F_{\mathcal{B}}$ has at most

$$(2w)^l \leq (2^{(h-3)/l})^l = 2^{h-3}$$

pieces by Lemma 3.4. Thus, we can bound t in Lemma 3.5 to see that

$$R_S(F_{\mathcal{B}}) \geq \frac{k-4t}{3k} \geq \frac{2^h - 4(2^{h-3})}{3(2^h)} = \frac{2^{h-1}}{6(2^{h-1})} = 1/6.$$

□

This proof illustrates how composition is able to create many more linear regions than addition. This tool is used in the construction of the deep network as well as the proof that the shallow network cannot create enough oscillations. However, the result proven above is not optimal as the later extensions in [2, 34] have generalized and improved the same types of results. Explicitly, Telgarsky in [34] proves the following Theorem.

Theorem 3.7. [34, Theorem 1.1] For any positive integer k there exist neural networks of depth $\Theta(k^3)$ and width $\Theta(1)$ with $\Theta(1)$ distinct parameters that cannot be approximated with $O(k)$ layers unless they have $\Omega(2^k)$ width.

The proof follows similar lines to the one above. More care must be taken to be able to use a more general class of activation functions. However, the result is general enough to include piecewise polynomial functions, convolutional networks, and even applies to models like boosted decision trees, which are not usually thought of as neural networks. That said, the family of "hard" functions is still very limited and contrived. This is an issue that many subsequent papers have tried to remedy by creating more realistic settings, which will be discussed in a subsequent section. More closely related to this work, Arora et al. in [2] provide a smoothly parameterized class of "hard" functions and strengthens the result of [34] by using "hard" networks of non-constant width to prove a super-exponential rather than exponential depth gap.

3.2.1. *Single-layer depth gaps.* Another area of recent work has been to show efficiency gaps between networks with one and two hidden layers. These results often consider radial functions for their proofs. An older result of Cheang and Barron in [6] shows how depth-2 networks can represent high dimensional balls. In that paper, the authors use networks with threshold activation functions (the indicator of a half-space). They find that the networks can represent balls in $\mathbb{R}^n$ to ϵ error with width $\frac{n^2}{\epsilon^2}$ in the first layer and constant width in the second layer. The idea is to us the half spaces to efficiently represent a Gaussian function with the first layer, then to threshold the gaussian to get a ball with the second. The result suggests that using the second hidden layer to threshold the function calculated by the first leads to a dramatic improvement in efficiently approximating certain functions.

More recently, Eldan and Shamir in [12] construct a radial function that can be well approximated by a depth-2 network with width polynomial in the input dimension n, but a depth-1 network requires width exponential in n. This result was extended by Safran and Shamir in [31] to an exponential advantage for indicators of L_2 unit balls and to any piecewise

linear L_1-radial function. The proofs of these results are very involved and require many technical tools, such as tempered distributions. A similar gap was proved by Daniely in [10] for a more general class of functions and using somewhat simpler methods from harmonic analysis (although still beyond the scope of this report). For completeness we will state and consider an informal version of the main result.

Theorem 3.8. [10] Let $f : \mathbb{S}^{d-1} \times \mathbb{S}^{d-1} \to \mathbb{R}$ be a function of the form $f(x, y) = g(\langle x, y \rangle)$ for $g : [-1, 1] \to \mathbb{R}$ and $\langle \cdot, \cdot \rangle$ the inner product on the sphere. Then networks with only one hidden layer, width polynomial in d, and exponentially bounded weights cannot approximate f with respect to the uniform distribution on $\mathbb{S}^{d-1} \times \mathbb{S}^{d-1}$ whenever g cannot be approximated by a low degree polynomial. For many choices of g, such as $g(x) = \sin(\pi d^3 x)$, require $\Omega(2^{d \log d})$ width to represent f with one hidden layer.

An important difference between these single-layer gaps and the gaps explained above is that the single-layer results make great use of the input dimensionality. Above, we only consider function from $\mathbb{R} \to \mathbb{R}$, but here we are using all of the input dimension to strengthen the power of depth. Expanding upon the relationship between depth and high dimensional input could be an important avenue of research. Unfortunately, many of the complicated tools used in the proofs of these results do not seem to extend to greater depth. The tools are complicated enough when only dealing with addition, they cannot easily be used to prove statements about compositions of such functions. New tools may be needed to extend such results.

3.2.2. *Depth gaps on smooth functions.* Recently there has also been a series of depth gap type results on classes of smooth functions. Three papers [21, 31, 36] were all published independently and nearly concurrently that prove similar results. The work of Liand and Srikant in [21] relies on a network architecture that uses a non-standard mixture of ReLu and threshold activations. Since the other papers achieve essentially the same results by similar ideas without using this architecture, we will focus on those papers. The informal statement of the result is found below, with the more nuanced details following.

Theorem 3.9. (Informal) For some family of nonlinear but smooth functions, every function is approximated to ϵ accuracy by a network of width and depth $O(poly(\log(1/\epsilon)))$. However, fixed depth networks require $\Omega(poly(1/\epsilon))$ width to approximate the same function.

The nuances come from choosing the family of functions over which the result holds and the norm to judge approximation error. Safran and Shamir in [31] choose C^2 functions which are sufficiently nonlinear in a substantial portion of the domain and are ϵ approximable by polynomials. Moreover, they use L_2 for the lower bounds and L_∞ for the upper bounds (which are the stronger results in each case respectively). Yarotsky in [36] varies the setting fairly often across the paper, except for the use of L_∞ norm throughout. The upper bounds consider functions in the unit ball of the Sobolev space with integer k, and the construction creates a network of depth $O(\log(1/\epsilon))$ but width $O((1/\epsilon)^{-n/k} \log(1/\epsilon))$, weaker than the bound in [31] unless we take $k = \infty$. The lower bounds of [36] sometimes use this Sobolev setting and then sometimes in C^2. Overall, the results of [36] seem to attempt to find a more general setting, but achieve slightly weaker bounds than [31].

For clarity of setting, we will present the results of Safran and Shamir from [31] in more detail, recalling that many of the ideas are similar across the various papers. First we will

precisely state the nonlinearity condition on the class of functions being considered. This means we need to be able to quantify the amount that a function is nonlinear. We do this by taking the measure of the largest set on which the function has non-zero Hessian along some direction.

Definition 3.10. [31, Definition 1] Let μ_n be the uniform distribution on $[0,1]^n$, $f : [0,1]^n \to \mathbb{R}$, and $\lambda > 0$. Let $\mathcal{U}$ be the set of all connected and measurable sets in $[0,1]^n$ and $H(f)$ be the Hessian. Then

$$\sigma_\lambda(f) = \sup_{v \in \mathbb{S}^{n-1}, U \in \mathcal{U},\ s.t.\ v^T H(f)(x) v \geq \lambda \forall x \in U} \mu_n(U).$$

Generally we consider the set of C^2 functions with $\sigma_\lambda(f)$ bounded below by some constant for some $\lambda > 0$. Moreover the same results apply with $v^T H(f)(x) v \leq -\lambda$ instead. With the setting clarified, we can more precisely state the results and sketch the proofs. First we will present the lower bound.

Theorem 3.11. [31, Theorem 4] For any C^2 function $f : [0,1]^n \to \mathbb{R}$ and $\lambda > 0$ and ReLU network $\mathcal{A}$ of width w and depth l, then for some universal constant c:

$$\|f - F_{\mathcal{A}}\|_{L_2} \geq \frac{c\lambda^2 (\sigma_\lambda(f))^5}{(2w)^{4l}}.$$

The proof is very involved and relies on several technical lemmas. At a very high level, the authors use tools from Legendre polynomials to show that strictly curved functions cannot be well approximated in L_2 by piecewise linear functions unless the number of regions is very large. They begin by considering the one-dimensional case and the function x^2. Once they have the lower bound there they extend it to convex functions and then to input dimension n. To make the transition from piecewise linear functions without too many pieces to ReLU networks, they rely on a lemma that bounds the number of linear regions in a ReLU network along any 1-dimensional affine subspace of the input space.

One interesting takeaway from the theorem is the scaling of the lower bound, which is polynomial in width but exponential in depth. This follows the trend seen with the above depth gap and the bounds on counting linear regions discussed in the next section.

The upper bound relies first on the assumption that the function is easily approximated by a polynomial that does not require more that $O(poly(\log(1/\epsilon)))$ operations to compute. Under this assumption, the main technical lemma is to prove that ReLU networks can efficiently well approximate multiplication. Then, composing these operations into a polynomial is fairly easy. The Lemma is precisely stated below.

Lemma 3.12. [31, Theorem 5] Let $f : [-M, M]^2 \to \mathbb{R}$ be multiplication, $f(x,y) = xy$, And $\epsilon > 0$. Then there exists a ReLU network $\mathcal{A}$ of width $w = O(\log(M/\epsilon))$ and depth $l = O(\log(M/\epsilon))$ such that

$$\sup_{(x,y) \in [-M,M]^2} |f(x,y) - F_{\mathcal{A}}| \leq \epsilon.$$

The proof mainly relies on the insight that performing bitwise operations on the first k bits of the inputs yields a result accurate to a factor of $2^{1-k}M$. Then setting k to $O(\log(M/\epsilon))$ we can achieve error less than any constant multiple of ϵ. The bulk of the proof is to define

a construction that extracts these k bits and performs the appropriate bitwise operations for the multiplication. The construction actually relies on the above construction of the "mirror" function f_m to prove the first depth gap. Recall that f_m^k divides the unit interval into 2^{k-1} identical regions of size 2^{1-k}. Cleverly manipulating this function allows for bit extraction, see the appendix of [31] for the details. With this Lemma, the paper then proves Theorem 3.9 for the sufficiently nonlinear C^2 functions that are easily represented by polynomials.

In the end, while these results are appealing, they require a set of very careful and nuanced assumptions. That said, the results still extend to a fairly general set of functions, where if our usual assumptions about smoothness of naturally occurring functions hold these results are in fact quite strong.

3.3. **Function complexity.** Some other results take a different approach to how to represent the capacity of a network. Rather than prove which classes of functions can approximate each other, these results show that deeper networks can represent functions with more complexity, under some notion of complexity. In this section we will examine one such complexity measure in some detail and point towards a few more.

3.3.1. *Counting linear regions.* Networks with piecewise linear activation functions, such as the ReLU, define piecewise linear functions. One measure of the complexity of a piecewise linear function is the number of linear pieces needed to describe the function. To be more precise, we offer the following definition, noting that the functions we are concerned with are actually affine rather than linear.

Definition 3.13. A *linear region* of a piecewise linear function $F : \mathbb{R}^n \to \mathbb{R}$ is a maximal connected subset S of $\mathbb{R}^n$ such that $F : S \to \mathbb{R}$ is affine, a translation of a linear function.

Now we can say that a network architecture has greater capacity if it can represent a function with a greater number of linear regions. Recent work in Montufar et al. [26] has examined exactly this question and given a construction for a lower bound on the maximal number of linear regions defined by a standard feedforward network of given dimensions. Their bound is stated and proven below.

Theorem 3.14. [26, Theorem 4] Let $\mathcal{A}$ be a network of depth l and layer widths $w_i > n$ for $1 \leq i \leq l$ defining a function $F_{\mathcal{A}} : \mathbb{R}^n \to \mathbb{R}$. Then the maximal number of linear regions defined by $F_{\mathcal{A}}$ is at least

$$\left(\prod_{i=1}^{l-1} \left\lfloor \frac{w_i}{n} \right\rfloor^n\right) \sum_{j=0}^{n} \binom{w_l}{j}.$$

Proof. The construction proceeds by "folding" the input space component by component. Essentially, by mapping many regions of the input space of each layer to the same set, we can identify many regions of space. Composing these identifications will multiply the number of regions from the original input space which are identified. Then, with the last layer we break the identified set into as many linear regions as possible.

To be explicit, consider the first layer with input dimension n and width w with $w > n$. Let $p = \lfloor \frac{w}{n} \rfloor$. Now we define $\mathcal{I}$ to be a partition of the hidden nodes into n subsets of cardinality p, excluding any remaining nodes. Now consider the jth subset $\mathcal{I}_j$, which will be

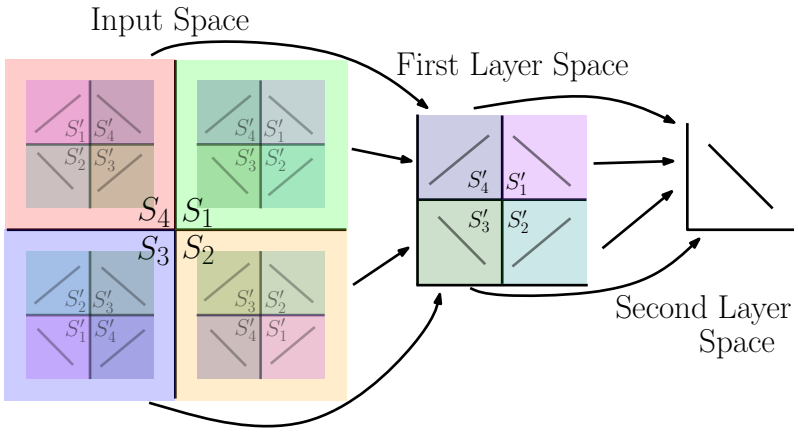

FIGURE 4

This figure, from [26], illustrates the motivation of this technique in the case where the input dimension is two and depth is three. We fold the space to identify S_1, S_2, S_3, S_4 by the first layer and then S'_1, S'_2, S'_3, S'_4 by the second. Then the line indicates the third layer splitting the space into linear regions by a hyperplane arrangement.

used to "fold" the jth component of the input space. Now we can define the functions h_i calculated by each node in $\mathcal{I}_j$ as follows:

$$\begin{aligned} h_1(x) &= \max\{0, e_j \cdot x\} \\ h_2(x) &= \max\{0, 2e_j \cdot x - 1\} \\ h_3(x) &= \max\{0, 2e_j \cdot x - 2\} \\ &\vdots \\ h_p(x) &= \max\{0, 2e_j \cdot x - (p-1)\}, \end{aligned}$$

where e_j is the standard basis vector that is 1 in the jth component and 0 elsewhere. Adding these functions with alternating signs, we construct the oscillating function g_j in the figure below, where

$$g_j(x) = (1, -1, 1, \ldots, (-1)^{p-1}) \cdot (h_1(x), h_2(x), \ldots, h_p(x)).$$

Note that since each h_i selects only the jth coordinate of x we are only really defining a function from $\mathbb{R} \to \mathbb{R}$ for each component. This function is linear on each of the intervals $[0, 1], [1, 2], \ldots, [p-1, p]$ and each interval is mapped to the unit interval $[0, 1]$. Now we can consider the function across all components: $g(x) = (g_1(x), g_2(x), \ldots, g_n(x))$. This function is locally symmetric about the hyperplanes $x_j = 0, x_j = 1, \ldots, x_j = p-1$ for every component j. These hyperplanes define a grid over the input space with p^n hypercube regions. Moreover, since each component is mapped to the unit interval on each linear component, g in fact identifies all p^n hypercubes into the unit hypercube of the input space.

Note that each g_j was defined by a single layer of a neural network of width p, the vector of h_i, composed with a linear function of alternating signs. To define a proper neural network, we can think of the linear transformation as being absorbed into the linear pre-activation function of the next layer in the network.

Inducting over the first $l-1$ layers, we see that we have identified precisely $\left(\prod_{i=1}^{l-1} \lfloor \frac{w_i}{n} \rfloor^n\right)$ regions of the original input space. For the last layer, we note that we want to divide the input, which is equivalent to the unit hypercube in n dimensions, into many regions. Now we consider the last layer in isolation as a function $f : [0,1]^n \to \mathbb{R}^{w_l}$ mapping $x \to \sigma(T_l(x))$. Let $T_l(x) = Ax + b$ and let A_i be the ith row of A. Then the linear regions are defined by the hyperplanes $H_i = \{x \in [0,1]^n : A_i x + b_i = 0\}$ for $1 \leq i \leq w_l$, since those define different orthants of $\mathbb{R}^{w_l}$ that the ReLU map will treat differently. So now the only question is how many regions the hyperplanes can divide the cube into. Now we note the classic result that an arrangement of w_l hyperplanes in general position in n dimensional space will partition the space into $\sum_{i=0}^{n} \binom{w_l}{i}$ regions [37]. Combining this with the previous construction, we now have a function that identifies $\left(\prod_{i=1}^{l-1} \lfloor \frac{w_i}{n} \rfloor^n\right)$ regions of space and then splits each one into $\sum_{i=0}^{n} \binom{w_l}{i}$ pieces, giving us the desired result. □

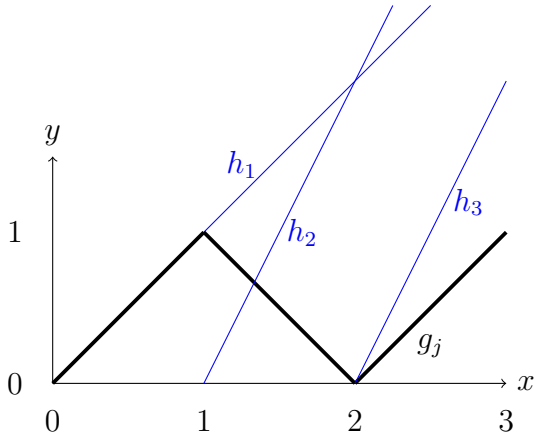

FIGURE 5
An illustration of the construction of one g_j in terms of $h_1, \cdots, h_p$ for $p = 3$. The h_i are pictured in blue and g_j (the linear combination of the h_i) in black.

To determine the strength of this bound, it is useful to consider the upper bound on the maximal number of linear regions. Recent work in [25, 29, 32] all prove such upper bounds. While Raghu et al. in [29] claim that their bound is asymptotically tight (only in terms of width of the network), it is actually weaker than the updated bound by Montufar in [25]. Here we will briefly explain the upper bound in [25] and note that the bound by Serra et al. in [32] is slightly stronger, but asymptotically equivalent and more involved to prove.

Theorem 3.15. [25, Proposition 3] Let $\mathcal{A}$ be a network of depth l and layer widths $w_i > n$ for $1 \leq i \leq l$ defining a function $F_{\mathcal{A}} : \mathbb{R}^n \to \mathbb{R}$. Then the maximal number of linear regions defined by $F_{\mathcal{A}}$ is at most

$$\prod_{i=1}^{l} \left(\sum_{j=0}^{n} \binom{w_i}{j} \right).$$

Proof. The proof follows from the above observation that a single layer network of width w_i can generate at most $\sum_{j=0}^{n} \binom{w_i}{j}$ regions as defined by a hyperplane arrangement. Then, at

the subsequent layer, a linear region defined by the ith layer of the network can be viewed as the input space and divided up into at most $\sum_{j=0}^{n} \binom{w_{i+1}}{j}$ sections. The bound follows immediately. Note that the bound is potentially quite loose since it is not at all clear that every region can be divided into the maximal number of pieces, and in fact this seems unlikely. □

Now we can briefly consider the asymptotics of these bounds. Assume that all of the w_i are the same, so the width of every layer is $w > n$. Then the upper bound becomes $\prod_{i=1}^{l} \left(\sum_{j=0}^{n} \binom{w}{j} \right) = O((\frac{ew}{n})^{nl})$. And the lower bound becomes $\left(\prod_{i=1}^{l-1} \lfloor \frac{w}{n} \rfloor^n \right) \sum_{j=0}^{n} \binom{w}{j} = \Omega((\frac{w}{n})^{n(l-1)}(\frac{w}{n})^n) = \Omega((\frac{w}{n})^{nl})$. The bounds are fairly close, but considering their ratio, the best bound on the ratio we can easily get is:

$$\frac{\prod_{i=1}^{l} \left(\sum_{j=0}^{n} \binom{w}{j} \right)}{\left(\prod_{i=1}^{l-1} \lfloor \frac{w}{n} \rfloor^n \right) \sum_{j=0}^{n} \binom{w}{j}} = \frac{\left(\sum_{j=0}^{n} \binom{w}{j} \right)^{l-1}}{\lfloor \frac{w}{n} \rfloor^{n(l-1)}} = \frac{O((\frac{ew}{n})^{n(l-1)})}{\lfloor \frac{w}{n} \rfloor^{n(l-1)}} = O(e^{n(l-1)}).$$

There may be an exponential difference between the bounds, but this is an upper bound on this ratio. There is room to improve both the upper and lower bounds to make this tight.

Regardless, what these bounds do show is a clear benefit to depth in terms of creating larger numbers of linear regions. Since the number of regions grows exponentially with depth, but only polynomially with width, there seems to be an efficiency benefit to deeper rather than wider networks.

3.3.2. *VC dimension.* Another way to measure the complexity of the functions that can be represented by a given architecture is to measure the VC dimension. The central idea of the VC dimension is to count how many points a class of functions can classify into any of the exponentially many possible classifications of those points. This type of idea stretches back to Cover's work on linear machines in [8] and was explicitly introduced by Vapnik and Chernovenkis in [35] (hence the name VC dimension). Here we consider some recent work of Bartlett et al. in [4] that proves some nearly tight bounds on the VC dimensions of neural networks based on depth. To present the result, first we need to be explicit about what the VC dimension is.

Definition 3.16. [4, Definition 1] Let $\mathcal{F}$ be a class of functions from $\mathbb{R}^n \to \mathbb{R}$. Then to translate this to the classification setting, define $H = \text{sgn}(\mathcal{F}) = \{\mathbb{1}[f > 0] : f \in \mathcal{F}\}$ to be the class of sign functions of functions in $\mathcal{F}$. Then a set of points $\{x_i\}_{i=1}^{m} \in \mathbb{R}^n$ is shattered by H if $|\{(h(x_1), \ldots, h(x_m)) : h \in H\}| = 2^m$. Now we define the VC dimension $VCdim(\mathcal{F}) = VCdim(H)$ to be the largest m such that some set of m points in $\mathbb{R}^n$ is shattered by H.

This measure of complexity is fairly natural since it corresponds to a practical problem to which the network will be applied. This notion of complexity is also often used in generalization theory to prove certain bounds on the generalization of a learned network. The recent work of [4] provides the following nearly tight bounds on the VC dimension.

Theorem 3.17. [4, Theorems 3 and 6] The class $\mathcal{F}_{w,l}$ of networks with piecewise linear activation functions of width w and depth l (and size $|\mathcal{A}| = w^2 l$) has VC dimension bounded

by:

$$\Omega(w^2l^2\log(w^2)) = \Omega(|\mathcal{A}|l\log(|\mathcal{A}|/l)) = VCdim(\mathcal{F}_{w,l}) = O(|\mathcal{A}|l\log(|\mathcal{A}|)) = O(w^2l^2\log(w^2l)).$$

The lower bound is proven with a construction that uses the "bit extraction" technique that uses a network to extract some lower order bits of the input to create a highly oscillatory function, in somewhat the same vein as the constructions above. The proof of the upper bound is more interesting, so we will present that here. Note that in our proof, we will restrict to the case where the activation is the ReLU function.

Proof. The proof relies on the following lemma due to Anthony and Bartlett [1, Theorem 8.3]:

Lemma 3.18. Let $p_1, \ldots, p_m$ be polynomials of at most degree d in $r \leq m$ variables. Define K to be the number of possible sign vectors given by the polynomials, then

$$K = |\{\text{sgn}(p_1(x)), \ldots, \text{sgn}(p_m(x)) : x \in \mathbb{R}^r\}| \leq 2\left(\frac{2emd}{r}\right)^r.$$

Let $W = |\mathcal{A}|$ be the total number of parameters in our network and let a be the vector in $\mathbb{R}^W$ of all of the parameters of $\mathcal{A}$. For any input $x \in \mathbb{R}^n$, define $f(x, a) = F_{\mathcal{A}}(x)$, to make clear that we are considering a function of the parameters. Then the class of functions we are considering is $\mathcal{F} = \{x \to f(x, a) : a \in \mathbb{R}^W\}$.

Fix some $x_1, \ldots, x_m \in \mathbb{R}^n$. We want to bound the possible number of sign patterns the network can have on the $\{x_i\}$. To do this, we define:

$$K = \left|\left\{\text{sgn}(f(x, a)), \ldots, \text{sgn}(f(x, a)) : a \in \mathbb{R}^W\right\}\right|.$$

Now note that for any partition $\mathcal{S} = \{P_1, \ldots, P_N\}$, we have

$$K \leq \sum_{i=1}^{N} |\{\text{sgn}(f(x, a)), \ldots, \text{sgn}(f(x, a)) : a \in P_i\}|.$$

The general idea of the proof is to partition $\mathbb{R}^W$, the parameter space, inductively layer by layer in such a way that $f(x, a)$ is a non-piecewise polynomial over each partition. Then, by bounding the number of partitions and applying the above Lemma, we can bound the VC dimension, ie bound K. To do this, we construct a sequence of partitions $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{l-1}$ with the following properties:

(1) Let W_i be the total number of weights up through layer i. Then we have $|\mathcal{S}_0| = 1$ and for each $i \in \{1, \ldots, l-1\}$

$$\frac{|\mathcal{S}_i|}{|\mathcal{S}_{i-1}|} \leq 2\left(\frac{2emw_ii}{W_i}\right)^{W_i}.$$

(2) For each $i \in \{1, \ldots, l-1\}$, each $S \in \mathcal{S}_{i-1}$, each $j \in \{1, \ldots, m\}$, and each unit h in the ith layer of the network, as a varies over S the input to h is a fixed polynomial in W_i variables of a with total degree no more than i.

Setting $\mathcal{S}_0 = \mathbb{R}^W$ satisfies the base case since $|\mathcal{S}_0| = 1$ and then the inputs to the first layer are affine (degree 1 polynomial) functions $T_1(x)$ of the weights of the "0th" layer in

this notation, ie we calculate the linear transformations as the inputs to the next layer. Now we just need to prove the inductive step for the ith layer.

Now for any unit $h \in \{1, \ldots, w_i\}$, any input $x_j \in \{x_1 \, dots, x_m\}$, and $S \in \mathcal{S}_{i-1}$ let $p_{h,x_j,S}$ be the function describing the net input to the hth unit of the ith layer from the input x_j for weights $a \in S$. By the inductive hypothesis of (2) we have that $p_{h,x_j,S}$ is a polynomial of degree at most i of W_i variables. Now by the lemma we have that the collection $\{p_{h,x_j,S} : h \in \{1, \ldots, w_i\}, x_j \in \{x_1 \, dots, x_m\}\}$ attains at most $2\left(\frac{2emw_i i}{W_i}\right)^{W_i}$ sign patterns on $\mathbb{R}^W$, which upper bounds the number of patterns on S which is the number of sets we partition S into for $\mathcal{S}_i$. Iterating over all $S \in \mathcal{S}_{i-1}$ gives us property (1). Now for any $S' \in \mathcal{S}_i$ note that the polynomials $\{p_{h,x_j,S} : h \in \{1, \ldots, w_i\}, x_j \in \{x_1 \, dots, x_m\}\}$ all have constant sign as a varies over S'. So, as a varies over S' the output of each ith layer unit in response to each x_j is a fixed polynomial of W_i variables of degree no more than i, so every input to every unit in the next layer is a degree $i+1$ polynomial of W_{i+1} variables, giving us property (2).

Proceeding like this we get a partition $\mathcal{S}_{l-1}$ of $\mathbb{R}^W$ so that the Lemma gives us that for $S \in \mathcal{S}_{l-1}$:

$$|\{\text{sgn}(f(x,a)), \ldots, \text{sgn}(f(x,a)) : a \in S\}| \leq 2\left(\frac{2eml}{W_l}\right)^{W_l}.$$

And property (1) gives us:

$$|\mathcal{S}_{l-1}| \leq \prod_{i=1}^{l-1} 2\left(\frac{2emw_i i}{W_i}\right)^{W_i}.$$

Combining this, we get that

$$K \leq \prod_{i=1}^{l} 2\left(\frac{2emw_i i}{W_i}\right)^{W_i} \leq 2^l \left(\frac{2eml\sum_{i=1}^{l} w_i}{\sum_{i=1}^{l} W_i}\right)^{\sum_{i=1}^{l} W_i}.$$

The desired result then requires the following technical lemma:

Lemma 3.19. Suppose that $2^m \leq 2^t(mr/k)^k$ for some $r \geq 16$ and $m \geq k \geq t \geq 0$. Then $m \leq t + k\log(2r\log(r))$.

Note that if the VC dimension is m we have that $2^{VCdim(\mathcal{F})} \leq K$ and $\sum_{i=1}^{l} W_i \leq lW$, so applying the lemma yields the result. □

The important difference between this complexity measure and the previous one that just counts linear regions is that the VC dimension has polynomial dependence on both l and w, rather than exponential dependence on l. This suggests that while deep networks are able to create highly oscillatory networks as in the constructions in the previous sections, the geometric constraints on these oscillations do not allow them to shatter a set of points. For example, in the construction of the depth gap, the deep network classifies alternating points into different classes but could not do the same for an arbitrary ordering of classes on the points. This result potentially shows some limitation of depth compared to the more optimistic results about depth from previous sections.

3.3.3. *Other complexity measures.* It is not clear which complexity measures will be the most useful to quantify how much more expressivity a given architecture has in practice. As such, people have developed several other notions of complexity to show the benefits of depth as well as to aid in model selection for real-world problems. Here we point towards a few of these alternative techniques for evaluating the expressive capabilities of a network.

The work of Bianchini and Scarselli in [5] define a notion of complexity with topological motivations related to Betti numbers. For a given network $\mathcal{A}$ they define $S_{\mathcal{A}} = \{x \in \mathbb{R}^n : F_{\mathcal{A}}(x) \geq 0\}$. Then the complexity measure is defined as $B(S_{\mathcal{A}}) = \sum_{i=0}^{n-1} b_i(S_{\mathcal{A}})$, where $b_i(S_{\mathcal{A}})$ is the ith Betti number, ie. the number of $i+1$ dimensional holes in $S_{\mathcal{A}}$. While the authors only consider differentiable activation functions like hyperbolic tangent, they show that for networks with one hidden layer the Betti complexity is bounded by $O(w^n)$, but for deep networks $\mathcal{A}$, one can achieve functions with complexity $\Omega(2^{wl})$. However, the upper bounds are not as good for networks with small numbers of hidden layers greater than one. That said, this work does give a different approach to demonstrating the advantage of depth which allows access to some more powerful mathematical tools.

Still in the vein of algebraic topology, Guss and Salakhutdinov in [15] define a notion of complexity by persistent homology. The idea is similar to the one above, except rather than summing the Betti numbers, the authors consider which homologies (very roughly which sets of Betti numbers) are achievable by a given architecture. They focus mainly on how this should inform model selection, but also explain that while achieving better bounds on the expressivity would require solving difficult open problems in algebraic topology there is an efficient computational method for testing the homology of trained networks in practice. Through their experiments, they claim to find important benefits of depth.

There are still more ways to measure network capacity. For example, using notions from Riemannian geometry and chaos theory, Poole et al. in [28] define a notion of complexity. They show that curvature grows exponentially with depth for one dimensional inputs and prove some preliminary results about how their metric behaves on random networks. On the other hand, with an eye towards generalization theory, Liang et al. in [22] define the Fisher-Rao metric inspired by information geometry (and Fisher information) to have specific geometric invariances. However, they do not prove any relationship between depth and the capacity of the network under this characterization of capacity.

Overall, these various techniques demonstrate how this field is still searching for the correct notions of complexity. It is very possible that the best notions of complexity will vary across different applications. As such, finding the appropriate notions of expressive power for given problems and constraints is a promising direction of research.

3.4. **Towards function approximation in realistic settings.** In practice, neural networks are used on real-world datasets to describe naturally occurring functions. This means that there are only certain types of more "realistic" functions that we need to be able to approximate. Adding some assumptions about the functions being approximated can yield different approximation efficiency results since certain architectures may be better suited to describe certain functions. So, practical results may also consider restricting the class of functions being approximated as well as restricting the architectures to those used in practice (like convolutional networks for example). Much recent work has considered these ideas.

3.4.1. *Inherent depth of real data.* One type of functions that depth seems inherently helpful to approximate are functions that are themselves naturally compositional. Poggio et al. in [27] argue that deep convolutional networks efficiently represent such a class of compositional functions. They argue that the local hierarchy of convolutions in subsequent layers can more easily approximate the natural hierarchy in image data for example. They add a mathematical formalism to this notion to prove some guarantees when the compositional structure of the network matches some structure in the desired function. Moreover, they empirically demonstrate their results using simple convolutional networks on real image data.

3.4.2. *Depth and alternative architectures.* Finally, there is a class of related results for non-standard network architectures. Above, we only consider feed-forward networks with traditional non-linear (usually ReLU) activation functions. For example, Delalleau and Bengio as well as Martens and Medabalimi in [11, 24] consider sum-product networks where the non-linear activations of traditional feed-forward networks are replaced by nodes that compute products of their inputs rather than linear combinations. They find families of functions where deep networks enjoy an exponential efficiency advantage. Another type of network considered by Cohen et al. in [7] is a convolutional network using product pooling. This looks somewhat like the incredibly successful convolutional networks used in practice, but uses products where the practical networks usually use a max or average. The paper proves via results in tensor decomposition that for randomly chosen weights in deep network, the network is almost surely not well approximated by a shallower network unless the shallow network has exponential width.

3.5. **Some comments on the literature.** There are a few trends to note across much of this body of literature. One observation across much of the literature is that many if not most high-dimensional constructions rely on defining some function in the one-dimensional case and then copying it across all components. Constructions like this seem to be missing out on some of the power of the high-dimensional setting. However, building high dimensional constructions will likely require a better understanding of the underlying structure of nonlinear composition.

Another trend is that of overwhelming optimism about depth. Nearly every result attempts to prove the advantage of depth over width. Depth seems to be exceptionally good at creating highly oscillatory functions. However, there also seems to be some inherent structure and symmetry to these functions (like the ones constructed for the depth gap and linear region lower bound) that is intimately tied to the increased oscillation. It is possible that this symmetry is a remnant of the one-dimensional constructions described above, but it also may be a more fundamental property of composition. Some of this limitation seems to be reflected in the VC dimension bounds which do not show much preference for depth over width. Better understanding the tradeoffs, not just benefits, of depth may be an important future direction.

4. Open questions

After reviewing the literature, there are clearly many directions of research that still have open questions. Here we formalize a few such questions, attempting to focus on areas that may be in reach and where an answer would likely need to provide some fundamental insight.

4.1. **Width gaps.** Based on the growing body of work about depth gaps, Lu et al. in [23] ask a sort of inverse question. Do some wide and shallow networks require exponentially large deep networks to represent the same functions? They conjecture that on the contrary a polynomial upper bound may exist, but offer no proof. Explicitly, such a result would need to show that for any network $\mathcal{A}$ of width w and depth l (potentially constant), there exists some network $\mathcal{B}$ with depth $poly(|\mathcal{A}|)$ and width $w' << w$ (maybe so that w' is near n) such that $\|F_{\mathcal{A}} - F_{\mathcal{B}}\|$ is small. The exact way that the approximation error comes into the bounds and how wide $\mathcal{B}$ are would likely depend upon the method of proof.

4.2. **Depth hierarchy.** There are many results proving a separation in expressivity between depth-1 and depth-2 networks, like those of [10, 12, 31] presented above. These results rely on defining functions that are inherently compositional (generally radial functions like the one that computes some function of an inner product). It remains open whether such a result exists for general l and $l+1$ depth networks. Proving such a result would be a major breakthrough in the understanding of function approximation by neural networks. Such a result has been proven in the domain of circuit complexity in theoretical computer science by Hastad in [17], but that discrete setting does not easily translate to the continuous case of neural networks (as will be discussed in a later section).

4.3. **Tight bounds on maximum number of linear regions.** As seen in the previous section, the current bounds on the number of linear regions are not quite tight. While the bounds may not seem that far apart, there is a fundamental problem preventing the creation of tight bounds. Namely, the construction that yields the lower bound relies on using one-dimensional ideas copied across all the input components while the upper bound considers general hyperplane arrangements without using the inherent structure of the composition of ReLU nonlinearities. Bridging this gap will likely require new geometric or algebraic insight into the fundamental qualities of piecewise linear composition.

5. Attempted Approaches

This section examines some possible, but not yet viable, approaches to the open questions posed in the previous section. Each subsection provides the rationale behind an approach and then some of the difficulties that would need to be surmounted to make the approach useful.

5.1. **A Geometric Interpretation.** There is a certain geometric interpretation of the function that a neural network represents that is not found in the literature, but may be a fruitful way to think about the problem. A network calculates a function by a series of operations in $\mathbb{R}^{w_i}$ and usually we just keep track of the vector of activations after each iterative application of a linear transformation and activation function. Instead, it is also possible to keep track of the global geometry of the function (as parameterized by the input space $\mathbb{R}^n$). This gives us

a piecewise linear n-dimensional manifold-like object in $\mathbb{R}^w$ whose parameterization defines the function of the network. This interpretation makes explicit that while depth clearly represents composition, width represents the dimension of the ambient space that we work inside of.

We can make this interpretation more explicit in the case where each layer has width w:

(1) (Embed) The transformation T_1 embeds a linear manifold, $M_1 = T_1(\mathbb{R}^n)$, into $\mathbb{R}^w$.
(2) (Fold) The ReLU function, σ, defines a piecewise linear function on $\mathbb{R}^w$ where the linear regions are precisely the orthants of the space. Moreover, note that in each orthant O, the map σ is in fact an orthogonal projection onto some subset of the boundary of O. So, $S_1 = \sigma(M_1)$ is now a piecewise linear manifold, a sort of "folded" version of M_1 into the positive orthant.
(3) (Transform) The transformation T_2 performs some affine transformation and shift on S_1, to give us $M_2 = T_2(S_1)$. This still yields a piecewise linear manifold. The process of iteratively calculating M_i and S_i then continues through the l layers (but note that the objects may not be manifolds anymore as the iteration may cause self-intersection).
(4) (Project) The final affine transformation, T_{l+1} is a map from $\mathbb{R}^w \to \mathbb{R}$ define by projection onto the vector $v \in \mathbb{R}^w$ of weights of the transformation followed by a shift by some $b \in \mathbb{R}$. So, the output of the overall function is $P_v(S_l) + b$.

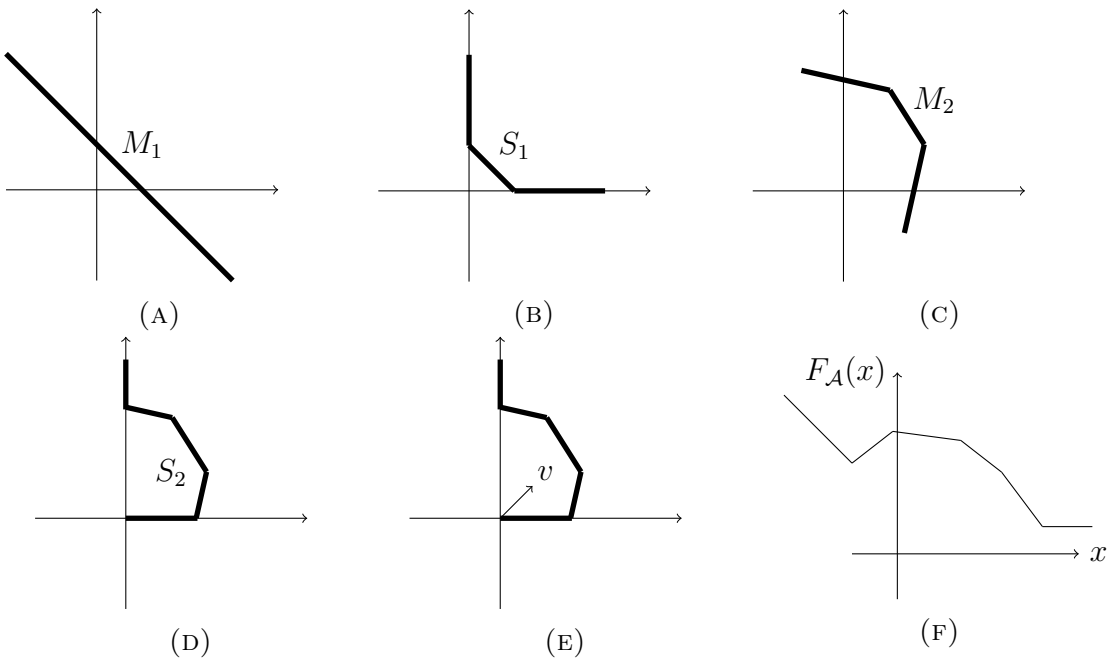

FIGURE 6
An illustration of this geometric interpretation for a network with $n = 1$, $w = 2$, $l = 2$. Going through each sketch: (A) demonstrates $\mathbb{R} = \mathbb{R}^n$ embedded in $\mathbb{R}^2 = \mathbb{R}^w$, (B) folds the embedding, (C) applied another affine transformation, (D) folds again, (E) shows the output vector v on which to project, (F) shows the resulting function.

One thing to note about this interpretation is that is is not enough to only maintain information about the successive M_i, S_i as sets since some points from the input space may

be mapped to the same place. As such, what we are really keeping is the parameterization of each M_i, S_i in terms of the inputs to the network. Also note that we can let $\|v\| = 1$ and calculate the any function by multiplying the weights in the previous layer by $1/\|v\|$. This makes P_v an orthogonal projection.

We attempted to use this setting to gain some traction on either a width gap bound or a better count of linear regions, but neither produced the desired results. However, we can walk through a bit of the thinking in these directions without proving results.

First consider the case of the width gap. We would like to take a network $\mathcal{A}$ of width w and depth l, and efficiently (in terms of network size) create a narrower but deeper network $\mathcal{B}$ to represent the same function. Let V be a subspace of $\mathbb{R}^w$ containing v from step (4) of the interpretation. Now, note that $P_v(P_V(S_l)) = P_v(S_l)$, where P_V is the orthogonal projection of S_l onto V, since they are both orthogonal projections and $\text{span}\{v\} \subseteq V$. If V is a w' dimensional subspace with $w' < w$, then $P_V(S_l)$ defines a piecewise linear object in $\mathbb{R}^{w'}$ that when projected onto v yields the same function as $\mathcal{A}$. The question is whether, or under what conditions, can we define a network $\mathcal{B}$ that uses width w' and not too much depth but calculates, or closely approximates, $\mathcal{A}$ by computing $P_V(S_l)$ as its last layer before the output. This reformulation may be easier to approach than the original way the width gap question was posed, but it is not clear that this is the case.

Now consider counting the maximum number of linear regions defined by a network $\mathcal{A}$ of width w and depth l. In the upper bound proved above, we saw that a network can achieve no more than $\sum_{i=0}^{n} \binom{w}{i}$ linear regions with any single layer by considering how the hyperplanes $\{x_i = 0 : x \in \mathbb{R}^w, 1 \leq i \leq w\}$ which define the nonlinearities of the ReLU intersect the embedding of an n-dimensional translated subspace of $\mathbb{R}^w$. The upper bound is obtained by just composing this bound on every linear region of every subsequent layer and thus taking the product of this bound across layers. However, this seems to ignore some of the potential geometric constraints. Using the above geometric interpretation, one could try to bound the number of regions that the w axis hyperplanes can divide a piecewise linear manifold M_i into. However, it is not easy to see what the constraints on each M_i are. Finding a useful description of how each subsequent M_i can intersect the axes hyperplanes would be very useful in this direction.

5.2. **Using Network Graphs.** Using much the same idea as the construction of the universal fixed-width approximator in Lemma 3.2, we can try to make some progress on finding an upper bound on any width gap as described in Section 4.1. Essentially, the strategy is to take the graph of any network of width w and depth l and to make a narrower but deeper network that calculates the same function by taking advantage of our ability to map the identity function along certain edges if we assume that we have a compact input space. Unfortunately, using this technique we were not able to find a polynomial upper bound on width gaps, as has been conjectured. This technique can prove an exponential upper bound by using one accumulation neuron per layer (so the network has width on the order of $n + l$) and using recursion to calculate the overall function. But, this is a weak result.

Moreover, we do not expect this technique to be able to prove any sort of polynomial upper bound on width gaps. This technique can be seen as a sort of directed graph minor problem. Essentially we are asking if there is a narrow network $\mathcal{B}$ with depth polynomial in the width of the original network $\mathcal{A}$ such that the graph of $\mathcal{A}$ is contained in the graph of $\mathcal{B}$

as a "minor" by deleting and contracting (using the identity map) edges. However, defining graph minors on directed acyclic graphs like our networks and attempting to maintain a valid network graph does not work so simply (see [20] and references therein for a more detailed review of directed graph minors). Moreover, finding such graph minors is in general a computationally hard problem, and the structure of the network graphs does not seem to solve this. Thus, it seems unlikely that this strategy will be s fruitful direction.

5.3. **Connections to Circuit Lower Bounds.** The desired generalization of the single layer depth gaps to arbitrary depths would look much like some of the depth hierarchy results that have been obtained in circuit complexity theory. In this section we will consider why these results do not directly extend and what insights may or may not be transferable from circuits to neural networks.

A classic paper of Hastad [17] proves two key results. First, there do not exist constant depth, polynomial size circuits to compute parity since such circuits must have depth $\log n / \log \log n$. Second, for any $l \geq 2$ there exist functions f_l^n on n inputs that are computable with polynomial size circuits of depth l, but any circuit of depth $l-1$ would require exponential size to compute the same function. This result was then strengthened in 2015 by Hastad, Rossman, Servedio, and Tan [18] to say that for any $l \geq 2$ there exist functions g_l^n that are computable by linear size circuits of depth l such that any depth $l-1$ circuit that agrees with g_l^n on $(1/2 + o_n(1))$ fraction of inputs must have size $\exp(n^{\Omega(1/d)})$. The functions used in both papers are based on Sipser functions, which have alternating layers of AND and OR gates of equal fan-in where the fan-in is $n^{1/l}$ in the original version and varies across layers in the more recent paper. The technique used in the older proof relies on random restrictions. This means that the values at given nodes in the circuit graph are restricted to either 0 or 1. Unfortunately, this does not transfer well to the continuous setting. The more recent result uses a more intricate version of this idea, which also does not seem to transfer to the continuous setting very readily.

In fact, it seems likely that there will be no way to use the circuit depth hierarchy results to prove any sort of neural network hierarchy. Immediately, we can see that the continuous setting is different since we can actually calculate the parity function with a simple network with only one hidden layer. This is noted by Eldan and Shamir in [12] where they show that the network:

$$\sum_{i=1}^{n+1} (-1)^{i+1} \sigma \left(\sum_{j=1}^{n} x_j - i + 1/2 \right)$$

where σ is the threshold activation indicating positive numbers. However, the construction extends to ReLU activations since it only requires $\sigma(z) = 1$ when $z > 1/2$ and $\sigma(z) = 0$ when $z < -1/2$ which can be computed by $\sigma(2z) - \sigma(2z-1)$ if σ is the ReLU. Moreover, Rumelhart et al. in the classic paper [30] even train a network with one hidden layer using backpropagation to calculate the parity function. The essential problem is that each computational unit of a neural network is given access to real numbers instead of bits and can perform scalar multiplication and addition as well as addition of outputs within single computational units while these operations would require larger circuits to compute. However, it may be possible to gain some inspiration from the randomization techniques since a ReLU activation does have only two regimes, somewhat like a boolean circuit. But large

modifications to the classes of difficult functions and proof techniques would likely be needed to extend circuit results to neural networks.

6. Conclusion

The goal of studying the approximation capabilities of neural networks is a rigorous understanding of their capabilities. There has clearly been much progress in recent years in better understanding the role that the depth of the network architecture can play in expanding the function approximation capabilities of a network. However, there are still some fundamental and simple questions about the expressive power of simple feedforward networks in terms of width and depth, as reviewed here. Moreover, the questions of approximation capabilities do not stop with just these simple architectures. While not the focus of this paper, networks used in practice often have more elaborate structure built in, so more work will be required to understand how those structural decisions effect approximation capabilities. In the end, progress in understanding the expressivity costs and benefits of various architectures will be able to inform more powerful and efficient computation in the future.

References

[1] M. Anthony and P. Bartlett, *Neural network learning: theoretical foundations*, Cambridge University Press, 1999.

[2] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, *Understanding deep neural networks with rectified linear units*, Electronic Colloquium on Computational Complexity, (2017).

[3] A. R. Barron, *Approximation and estimation bounds for artificial neural networks*, Machine Learning, 1 (1994), pp. 115–133.

[4] P. Bartlett, N. Harvey, C. Liaw, and A. Mehrabian, *Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks*, COLT, (2017).

[5] M. Bianchini and F. Scarselli, *On the complexity of neural network classifiers: a comparison between shallow and deep architectures*, IEEE Transactions on Neural Networks and Learning Systems, 25 (2014), pp. 1553–1565.

[6] G. Cheang and A. R. Barron, *A better approximation for balls*, Journal of Approximation Theory, (2000), pp. 183–203.

[7] N. Cohen, O. Sharir, and A. Shashua, *On the expressive power of deep learning: a tensor analysis*, JMLR: workshop and Conference Proceedings, 49 (2016), pp. 1–31.

[8] T. Cover, *Capacity problems for linear machines*, Pattern Recognition, (1968).

[9] G. Cybenko, *Approximating by superpositions of a sigmoidal function*, Mathematics of control, signals, and systems, 2 (1989), pp. 303–314.

[10] A. Daniely, *Depth separation for neural networks.* arxiv:1702.08489.

[11] O. Delalleau and Y. Bengio, *Shallow vs. deep sum-product networks*, Advances in Neural Information Processing Systems, (2011), pp. 666–674.

[12] R. Eldan and O. Shamir, *The power of depth for feedforward neural networks.* arxiv:1512:03965, 2016.

[13] K.-I. Funahashi, *On the approximate realization of continuous mapping by neural networks*, Neural Networks, 2 (1989), pp. 183–192.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[15] W. Guss and R. Salakhutdinov, *On characterizing the capacity of neural networks using algebraic topology.* arXiv:1802.04443v1, 2018.

[16] B. Hanin, *Universal function approximation by deep neural nets with bounded width and ReLU activations.* https://arxiv.org/abs/1708.02691, August 2017.

[17] J. Hastad, *Almost optimal lower bounds for small depth circuits*, Proceedings of the ACM, (1986).

[18] J. Hastad, B. Rossman, R. Servedio, and L. yang Tan, *An average-case depth hierarchy theorem for boolean circuits*, Journal of the ACM, 64 (2017).

[19] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, 4 (1991), pp. 251–257.

[20] S. Kreutzer and S. Tazari, *Directed nowhere dense classes of graphs*, SODA, (2011).

[21] S. Liang and R. Srikant, *Why deep neural networks?* arXiv:1610.04161, 2016.

[22] T. Liang, T. Poggio, A. Rakhlin, and J. Stokes, *Fisher-Rao metric, geometry and complexity of neural networks.* arXiv:1711.01530v1, 2017.

[23] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, *The expressive power of neural networks: A view from the width.* https://arxiv.org/abs/1709.02540, accepted NIPS 2017, September 2017.

[24] J. Martens and V. Medabalimi, *On the expressive efficiency of sum product networks*. arxiv:1411.7717, 2015.
[25] G. Montufar, *Notes on the nummber of linear regions of deep neural networks*, SampTA, (2017).
[26] G. Montufar, R. Pascanu, K. Cho, and Y. Bengio, *On the number of linear regions of deep neural networks*, NIPS, (2014).
[27] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, *Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review*, International Journal of Automation and Computing, 14 (2017), pp. 503–519.
[28] B. Poole, S. Lahiri, M. Raghu, J. Dickstein, and S. Ganguli, *Exponential expressivity in deep neural networks through transient chaos*, (2016).
[29] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Dickstein, *On the expressive power of deep neural networks*, ICML, (2017).
[30] D. Rumelhart, G. Hinton, and R. Williams, *Learning internal representations by error propagation*, MIT Press, 1986.
[31] I. Safran and O. Shamir, *Depth-width tradeoffs in approximating natural functions with neural networks*, International Conference on Machine Learning, (2017).
[32] T. Serra, C. Tjandraatmadja, and S. Ramalingam, *Bounding and counting linear regions of deep neural networks*. arXiv:1711.02114v2, 2018.
[33] M. Telgarsky, *Representation benefits of deep feedforward networks*. arXiv:1509.08101, 2015.
[34] ——, *Benefits of depth in neural networks*, in JMLR: Workshop and Conference Proceedings, vol. 49, 2016, pp. 1–23.
[35] V. Vapnik and A. Chervonenkis, *On the uniform convergence of relative frequenciees of events to their probabilities*, Theory of probability and its applications, 16 (1971).
[36] D. Yarotsky, *Error bounds for approximations with deep ReLU networks*. arXiv:1610.01145, 2017.
[37] T. Zaslavsky, *Facing up to arrangements: face-count formulas for partitions of space by hyperplanes*, Memoirs of the American Mathematical Society, 1 (1975).