CBMM Memo No. 058 **June 28, 2021**

Theory I: Why and When Can Deep Networks Avoid the Curse of Dimensionality?

by

Tomaso Poggio[1] Hrushikesh Mhaskar[2] Lorenzo Rosasco[1] Brando Miranda[1] Qianli Liao[1]

[1]Center for Brains, Minds, and Machines, McGovern Institute for Brain Research,
Massachusetts Institute of Technology, Cambridge, MA, 02139.
[2]Department of Mathematics, California Institute of Technology, Pasadena, CA 91125;
Institute of Mathematical Sciences, Claremont Graduate University, Claremont, CA 91711

Abstract: The paper characterizes classes of functions for which deep learning can be exponentially better than shallow learning. Deep convolutional networks are a special case of these conditions, though weight sharing is not the main reason for their exponential advantage. This 2021 version deletes Proposition 4 of the previous version which was wrong. Thanks to Eran Malach for pointing this out.

This work was supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF - 1231216. H.M. is supported in part by ARO Grant W911NF-15-1-0385.

International Journal of Automation and Computing X(X), X X, X-X
DOI: XXX

Theory I: Why and When Can Deep Networks Avoid the Curse of Dimensionality

Tomaso Poggio[1] Hrushikesh Mhaskar[2] Lorenzo Rosasco[1] Brando Miranda[1] Qianli Liao[1]

[1]Center for Brains, Minds, and Machines, McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA, 02139.
[2]Department of Mathematics, California Institute of Technology, Pasadena, CA 91125; Institute of Mathematical Sciences, Claremont Graduate University, Claremont, CA 91711

Abstract: The paper characterizes classes of functions for which deep learning can be exponentially better than shallow learning. Deep convolutional networks are a special case of these conditions, though weight sharing is not the main reason for their exponential advantage. This 2021 version deletes Proposition 4 of the previous version which was wrong. Thanks to Eran Malach for pointing this out.

Keywords: Deep and Shallow Networks, Convolutional Neural Networks, Function Approximation, Deep Learning

1 A theory of deep learning

1.1 Introduction

There are at three main theory questions about Deep Neural Networks. The first set of questions is about the power of the architecture – which classes of functions can it approximate and learn well? The second set of questions is about the learning process: what is the landscape of the empirical risk? The third question is about generalization. Overparametrization may explain why minima are easy to find during training but then why do the minima found by Stochastic Gradient Descent generalize so well?

In this paper we focus especially on the first set of questions, summarizing several theorems that have appeared online in 2015[1, 2, 3], and in 2016[4, 5]). We then describe additional results as well as a few conjectures and open questions. The main message is that deep networks have the theoretical guarantee, which shallow networks do not have, that they can avoid the *curse of dimensionality* for an important class of problems, corresponding to *compositional functions*, that is functions of functions. An especially interesting subset of such compositional functions are *hierarchically local compositional functions* where all the constituent functions are local in the sense of bounded small dimensionality. The deep networks that can approximate them without the curse of dimensionality are of the deep convolutional type (though weight sharing is not necessary).

Implications of the theorems likely to be relevant in practice are:

1. Certain *deep convolutional architectures* have a theoretical guarantee that they can be *much better* than one layer architectures such as kernel machines;

2. the problems for which certain deep networks are guaranteed to avoid the curse of dimensionality (see for a nice review [6]) correspond to input-output mappings that are *compositional*. Since in a very specific sense all computable functions are compositional, we focus on certain specific types of compositionality. The most interesting class of functions are composed of a hierarchy of constituent functions that are local: an example is $f(x_1, \cdots, x_8) = h_3(h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8)))$. The compositional function f requires only "local" computations (here with just dimension 2) in each of its constituent functions h;

3. the key aspect of convolutional networks that can give them an exponential advantage is *not weight sharing* but *locality* at each level of the hierarchy.

2 Previous theoretical work

Deep Learning references start with Hinton's backpropagation and with Lecun's convolutional networks (see for a nice review [7]). Of course, multilayer convolutional networks have been around at least as far back as the optical processing era of the 70s. The Neocognitron[8] was a convolutional neural network that was trained to recognize characters. The property of *compositionality* was a main motivation for hierarchical models of visual cortex such as HMAX which can be regarded as a pyramid of AND and OR layers[9], that is a sequence of conjunctions and disjunctions. Several papers in the '80s focused on the approximation power and learning properties of one-hidden layer networks (called shallow networks here). Very little appeared on multilayer networks, (but see [10, 11, 12]), mainly because one hidden layer nets performed empirically as well as deeper networks. On the theory side, a review by Pinkus in 1999[13] concludes that "...there seems to be reason to conjecture that the two hidden layer model may be significantly more promising than the single hidden layer model...". A version of the questions about the importance of hierarchies was asked in [14] as follows: "*A comparison with real brains offers another, and probably related, challenge to learning theory. The "learning algorithms" we have described in this paper correspond to one-layer architectures. Are hierarchical architectures with more layers justifiable in terms of learning theory? It seems that the learning theory of the type we have outlined does not offer any general argument in favor of hierarchical learning machines for regression or classification. This is somewhat of a puzzle since the organization of cortex – for instance visual cortex – is strongly hierarchical. At the same time, hierarchical learning systems show superior performance in several engineering applications.*" Because of the great empirical success of deep learning over the last three years, several papers addressing the question of why hierarchies have appeared. Sum-Product networks, which are equivalent to polynomial networks (see [15, 16]), are a simple case of a hierarchy that was analyzed[17] but did not provide particularly useful insights. Montufar and Bengio[18] showed that the number of linear regions that can be synthesized by a deep network with ReLU nonlinearities is much larger than by a shallow network. The meaning of this result in terms of approximation theory and of our results is at the moment an open question[1]. Relevant to the present review is the work on hierarchical quadratic networks[16], together with function approximation results[19, 13]. Also relevant is the conjecture by Shashua (see [20]) on a connection between deep learning networks and the hierarchical Tucker representations of tensors. In fact, our theorems describe formally the class of functions for which the conjecture holds. This paper describes and extends results pre-

[1]We conjecture that the result may be similar to other examples in section 4.2. It says that among the class of functions that are piecewise linear, there exist functions that can be synthesized by deep networks with a certain number of units but require a much large number of units to be synthesized by shallow networks

sented in[21, 22, 23] and in[24, 4] which derive new upper bounds for the approximation by deep networks of certain important classes of functions which avoid the curse of dimensionality. The upper bound for the approximation by shallow networks of general functions was well known to be exponential. It seems natural to assume that, since there is no general way for shallow networks to exploit a compositional prior, lower bounds for the approximation by shallow networks of compositional functions should also be exponential. In fact, examples of specific functions that cannot be represented efficiently by shallow networks have been given very recently by Telgarsky [25] and by Shamir [26]. We provide in theorem 5 another example of a class of compositional functions for which there is a gap between shallow and deep networks.

3 Function approximation by deep networks

In this section, we state theorems about the approximation properties of shallow and deep networks.

3.1 Degree of approximation

The general paradigm is as follows. We are interested in determining how complex a network ought to be to *theoretically guarantee* approximation of an unknown target function f up to a given accuracy $\epsilon > 0$. To measure the accuracy, we need a norm $\|\cdot\|$ on some normed linear space $\mathbb{X}$. As we will see the norm used in the results of this paper is the sup norm in keeping with the standard choice in approximation theory. Notice, however, that from the point of view of machine learning, the relevant norm is the L_2 norm. In this sense, several of our results are stronger than needed. On the other hand, our main results on compositionality require the sup norm in order to be independent from the unknown distribution of the inputa data. This is important for machine learning.

Let V_N be the be set of all networks of a given kind with complexity N which we take here to be the total number of units in the network (e.g., all shallow networks with N units in the hidden layer). It is assumed that the class of networks with a higher complexity include those with a lower complexity; i.e., $V_N \subseteq V_{N+1}$. The *degree of approximation* is defined by

$$\mathsf{dist}(f, V_N) = \inf_{P \in V_N} \|f - P\|. \tag{1}$$

For example, if $\mathsf{dist}(f, V_N) = \mathcal{O}(N^{-\gamma})$ for some $\gamma > 0$, then a network with complexity $N = \mathcal{O}(\epsilon^{-\frac{1}{\gamma}})$ will be sufficient to guarantee an approximation with accuracy at least ϵ. Since f is unknown, in order to obtain theoretically proved upper bounds, we need to make some assumptions on the class of functions from which the unknown target function is chosen. This a priori information is codified by the statement that $f \in W$ for some subspace $W \subseteq \mathbb{X}$. This subspace is usually a smoothness class characterized by a smoothness parameter m. Here it will be generalized to a smoothness and compositional class, characterized by the parameters m and d ($d = 2$ in the example of Figure 1; in general is the size of the kernel in a convolutional network).

3.2 Shallow and deep networks

This section characterizes conditions under which deep networks are "better" than shallow network in approximating functions. Thus we compare shallow (one-hidden layer) networks with deep networks as shown in Figure 1. Both types of networks use the same small set of operations – dot products, linear combinations, a fixed nonlinear function of one variable, possibly convolution and pooling. Each node in the networks we consider usually corresponds to a node in the graph of the function to be approximated, as shown in the Figure. In particular each node in the network contains a certain number of units. A unit is a neuron which computes

$$(\langle x, w \rangle + b)_+, \tag{2}$$

where w is the vector of weights on the vector input x. Both t and the real number b are parameters tuned by learning. We assume here that each node in the networks computes the linear combination of r such units

$$\sum_{i=1}^{r} c_i(\langle x, t_i \rangle + b_i)_+ \tag{3}$$

Notice that for our main example of a deep network corresponding to a binary tree graph, the resulting architecture is an idealized version of the plethora of deep convolutional neural networks described in the literature. In particular, it has only one output at the top unlike most of the deep architectures with many channels and many top-level outputs. Correspondingly, each node computes a single value instead of multiple channels, using the combination of several units (see Equation 3). Our approach and basic results apply rather directly to more complex networks (see third note in section 6). A careful analysis and comparison with simulations will be described in future work.

The logic of our theorems is as follows.

- *Both shallow (a) and deep (b) networks are universal*, that is they can approximate arbitrarily well any continuous function of n variables on a compact domain. The result for shallow networks is classical. Since shallow networks can be viewed as a special case of deep networks, it clear that for any continuous function of n variables, there exists also a deep network that approximates the function arbitrarily well on a compact domain.

- We consider a special class of functions of n variables on a compact domain that are a hierarchical compositions of local functions such as

$$f(x_1, \cdots, x_8) = h_3(h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), \\ h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8))) \tag{4}$$

  The structure of the function in equation 4 is represented by a graph of the binary tree type. This is the simplest example of compositional functions, reflecting dimensionality $d = 2$ for the constituent functions h. In general, d is arbitrary but fixed and independent of the dimensionality n of the compositional function f. In our results we will often think of n increasing while d is fixed. In section 4 we will consider the more general compositional case.

- The approximation of functions with a *compositional structure* – can be achieved with the same degree of accuracy by deep and shallow networks but that the number of parameters are much smaller for the deep networks than for the shallow network with equivalent approximation accuracy. It is intuitive that a hierarchical network matching the structure of a compositional function should be "better" at approximating it than a generic shallow network but universality of shallow networks asks for non-obvious characterization of "better". Our result makes clear that the intuition is indeed correct.

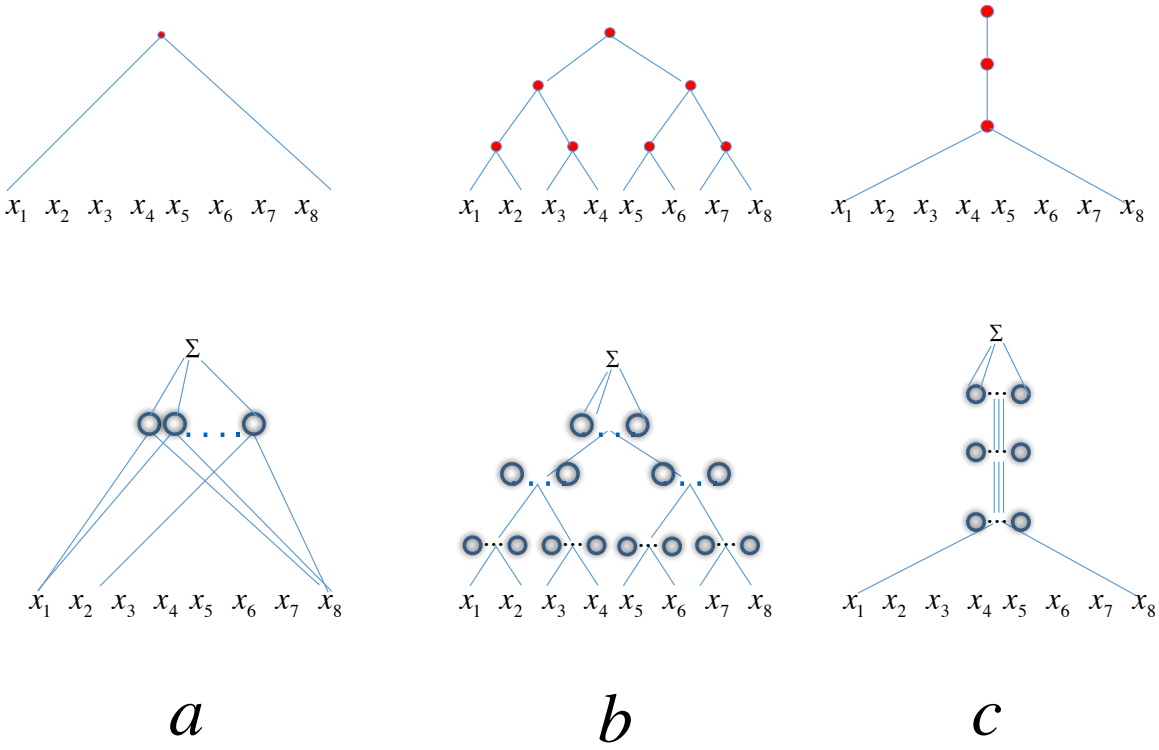

Figure 1 The top graphs are associated to *functions*; each of the bottom diagrams depicts the ideal *network* approximating the function above. In a) a shallow universal network in 8 variables and N units approximates a generic function of 8 variables $f(x_1, \cdots, x_8)$. Inset b) shows a binary tree hierarchical network at the bottom in $n = 8$ variables, which approximates well functions of the form $f(x_1, \cdots, x_8) = h_3(h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8)))$ as represented by the binary graph above. In the approximating network each of the $n - 1$ nodes in the graph of the function corresponds to a set of $Q = \frac{N}{n-1}$ ReLU units computing the ridge function $\sum_{i=1}^{Q} a_i(\langle \mathbf{v}_i, \mathbf{x} \rangle + t_i)_+$, with $\mathbf{v}_i, \mathbf{x} \in \mathbb{R}^2$, $a_i, t_i \in \mathbb{R}$. Each term in the ridge function corresponds to a unit in the node (this is somewhat different from todays deep networks, but equivalent to them, see text and note in 6). In a binary tree with n inputs, there are $log_2 n$ levels and a total of $n - 1$ nodes. Similar to the shallow network, a hierarchical network is universal, that is, it can approximate any continuous function; the text proves that it can approximate a compositional functions exponentially better than a shallow network. No invariance – that is weight sharing – is assumed here. Notice that the key property that makes convolutional deep nets exponentially better than shallow for compositional functions is the locality of the constituent functions – that is their low dimensionality. Weight sharing corresponds to all constituent functions at one level to be the same ($h_{11} = h_{12}$ etc.). Inset c) shows a different mechanism that can be exploited by the deep network at the bottom to reduce the curse of dimensionality in the compositional function at the top: leveraging different degrees of smoothness of the constituent functions, see Theorem 6 in the text. Notice that in c) the input dimensionality must be ≥ 2 in order for deep nets to have an advantage over shallow nets. The simplest examples of functions to be considered for a), b) and c) are polynomials with a structure corresponding to the graph at the top.

In the perspetive of machine learning, we assume that the shallow networks do not have any structural information on the function to be learned (here its compositional structure), because they cannot represent it directly and cannot exploit the advantage of a smaller number of parameters. In any case, in the context of approximation theory, we will exhibit and cite lower bounds of approximation by shallow networks for the class of compositional functions. Deep networks with standard architectures on the other hand *do represent* compositionality in their architecture and can be adapted to the details of such prior information.

We approximate functions of n variables of the form of Equation (4) with networks in which the activation nonlinearity is a smoothed version of the so called ReLU, originally called *ramp* by Breiman and given by $\sigma(x) = x_+ = max(0, x)$. The architecture of the deep networks reflects Equation (4) with each node h_i being a ridge function, comprising one or more neurons.

Let $I^n = [-1, 1]^n$, $\mathbb{X} = C(I^n)$ be the space of all continuous functions on I^n, with $\|f\| = \max_{x\in I^n} |f(x)|$. Let $\mathcal{S}_{N,n}$ denote the class of all shallow networks with N units of the form

$$x \mapsto \sum_{k=1}^{N} a_k \sigma(\langle w_k, x\rangle + b_k),$$

where $w_k \in \mathbb{R}^n$, $b_k, a_k \in \mathbb{R}$. The number of trainable parameters here is $(n+2)N \sim n$. Let $m \geq 1$ be an integer, and W_m^n be the set of all functions of n variables with continuous partial derivatives of orders up to $m < \infty$ such that $\|f\| + \sum_{1\leq|\mathbf{k}|_1\leq m} \|D^{\mathbf{k}} f\| \leq 1$, where $D^{\mathbf{k}}$ denotes the partial derivative indicated by the multi-integer $\mathbf{k} \geq 1$, and $|\mathbf{k}|_1$ is the sum of the components of $\mathbf{k}$.

For the hierarchical binary tree network, the analogous spaces are defined by considering the compact set $W_m^{n,2}$ to be the class of all compositional functions f of n variables with a binary tree architecture and constituent functions h in W_m^2. We define the corresponding class of deep networks $\mathcal{D}_{N,2}$ to be the set of all deep networks with a binary tree architecture, where each of the constituent nodes is in $\mathcal{S}_{M,2}$, where $N = |V|M$, V being the set of non–leaf vertices of the tree. We note that in the case when n is an integer power of 2, the total number of parameters involved in a deep network in $\mathcal{D}_{N,2}$ – that is, weights and biases, is $4N$.

Two observations are critical to understand the meaning of our results:

- compositional functions of n variables are a subset of functions of n variables, that is $W_m^n \supseteq W_m^{n,2}$. Deep networks can exploit in their architecture the special structure of compositional functions, whereas shallow networks are blind to it. Thus from the point of view of shallow networks, functions in $W_m^{n,2}$ are just functions in W_m^n; this is not the case for deep networks.

- the deep network does not need to have exactly the same compositional architecture as the compositional function to be approximated. It is sufficient that the acyclic graph representing the structure of the function is *a subgraph of the graph representing the structure of the deep network*. The degree of approximation estimates depend on the graph associated with the network and are thus an upper bound on what could be achieved by a network exactly matched to the function architecture.

The following two theorems estimate the degree of approximation for shallow and deep networks.

3.3 Shallow networks

The first theorem is about shallow networks.

Theorem 1. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be infinitely differentiable, and not a polynomial. For $f \in W_m^n$ the complexity of shallow networks that provide accuracy at least ϵ is*

$$N = \mathcal{O}(\epsilon^{-n/m}) \text{ and is the best possible.} \tag{5}$$

Notes In [27, Theorem 2.1], the theorem is stated under the condition that σ is infinitely differentiable, and there exists $b \in \mathbb{R}$ such that $\sigma^{(k)}(b) \neq 0$ for any integer $k \geq 0$. It is proved in [28] that the second condition is equivalent to σ not being a polynomial. The proof in [27] relies on the fact that under these conditions on σ, the algebraic polynomials in n variables of (total or coordinatewise) degree $< q$ are in the uniform closure of the span of $\mathcal{O}(q^n)$ functions of the form $\mathbf{x} \mapsto \sigma(\langle w, x\rangle + b)$ (see Appendix 4.1). The estimate itself is an upper bound on the degree of approximation by such polynomials. Since it is based on the approximation of the polynomial space contained in the ridge functions implemented by shallow networks, one may ask whether it could be improved by using a different approach. The answer relies on the concept of nonlinear n–width of the compact set W_m^n (cf. [29, 4]). The n-width results imply that the estimate in Theorem (1) is *the best possible* among *all* reasonable [29] methods of approximating arbitrary functions in W_m^n. □ The estimate of Theorem 1 is the best possible if the only a priori information we are allowed to assume is that the target function belongs to $f \in W_m^n$. The exponential dependence on the dimension n of the number $\epsilon^{-n/m}$ of parameters needed to obtain an accuracy $\mathcal{O}(\epsilon)$ is known as the *curse of dimensionality*. Note that the constants involved in $\mathcal{O}$ in the theorems will depend upon the norms of the derivatives of f as well as σ.

A simple but useful corollary follows from the proof of Theorem 1 about polynomials (which are a smaller space than spaces of Sobolev functions). Let us denote with P_k^n the linear space of polynomials of degree at most k in n variables. Then

Corollary 1. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be infinitely differentiable, and not a polynomial. Every $f \in P_k^n$ can be realized with an arbitrary accuracy by shallow network with r units, $r = \binom{n+k}{k} \approx k^n$.*

3.4 Deep hierarchically local networks

Our second and main theorem is about deep networks with smooth activations and is recent (preliminary versions appeared in [3, 2, 4]). We formulate it in the binary tree case for simplicity but it extends immediately to functions that are compositions of constituent functions of a fixed number of variables d instead than of $d = 2$ variables as in the statement of the theorem (in convolutional networks d corresponds to the size of the kernel).

Theorem 2. *For $f \in W_m^{n,2}$ consider a deep network with the same compositonal architecture and with an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ which is infinitely differentiable, and not a polynomial. The complexity of the network to provide approximation with accuracy at least ϵ is*

$$N = \mathcal{O}((n-1)\epsilon^{-2/m}). \tag{6}$$

Proof To prove Theorem 2, we observe that each of the constituent functions being in W_m^2, (1) applied with $n = 2$ implies that each of these functions can be approximated from $\mathcal{S}_{N,2}$ up to accuracy $\epsilon = cN^{-m/2}$. Our assumption that $f \in W_m^{N,2}$ implies that each of these constituent functions is Lipschitz continuous. Hence, it is easy to deduce that, for example, if P, P_1, P_2 are approximations to the

constituent functions h, h_1, h_2, respectively within an accuracy of ϵ, then since $\|h - P\| \leq \epsilon$, $\|h_1 - P_1\| \leq \epsilon$ and $\|h_2 - P_2\| \leq \epsilon$, then $\|h(h_1, h_2) - P(P_1, P_2)\| = \|h(h_1, h_2) - h(P_1, P_2) + h(P_1, P_2) - P(P_1, P_2)\| \leq \|h(h_1, h_2) - h(P_1, P_2)\| + \|h(P_1, P_2) - P(P_1, P_2)\| \leq c\epsilon$ by Minkowski inequality. Thus

$$\|h(h_1, h_2) - P(P_1, P_2)\| \leq c\epsilon,$$

for some constant $c > 0$ independent of the functions involved. This, together with the fact that there are $(n-1)$ nodes, leads to (6). □

Also in this case the proof provides the following corollary about the subset T_k^n of the space P_k^n which consists of compositional polynomials with a binary tree graph and constituent polynomial functions of degree k (in 2 variables)

Corollary 2. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be infinitely differentiable, and not a polynomial. Let $n = 2^l$. Then $f \in T_k^n$ can be realized by a deep network with a binary tree graph and a total of r units with $r = (n-1)\binom{2+k}{2} \approx (n-1)k^2$.*

It is important to emphasize that the assumptions on σ in the theorems are not satisfied by the ReLU function $x \mapsto x_+$, but they are satisfied by smoothing the function in an arbitrarily small interval around the origin. This suggests that the result of the theorem should be valid also for the non-smooth ReLU. Section 4.1 provides formal results. Stronger results than the theorems of this section (see [5]) hold for networks where each unit evaluates a Gaussian non–linearity; i.e., Gaussian networks of the form

$$G(x) = \sum_{k=1}^{N} a_k \exp(-|x - w_k|^2), \qquad x \in \mathbb{R}^d \tag{7}$$

where the approximation is on the entire Euclidean space.

In summary, when the only a priori assumption on the target function is about the number of derivatives, then to *guarantee* an accuracy of ϵ, we need a shallow network with $\mathcal{O}(\epsilon^{-n/m})$ trainable parameters. If we assume a hierarchical structure on the target function as in Theorem 2, then the corresponding deep network yields a guaranteed accuracy of ϵ with $\mathcal{O}(\epsilon^{-2/m})$ trainable parameters. Note that Theorem 2 applies to all f with a compositional architecture given by a graph which correspond to, or is a subgraph of, the graph associated with the deep network – in this case the graph corresponding to $W_m^{n,d}$. Theorem 2 leads naturally to the notion of *effective dimensionality* that we formalize in the next section

Definition 1. *The effective dimension of a class W of functions (for a given norm) is said to be d if for every $\epsilon > 0$, any function in W can be recovered within an accuracy of ϵ (as measured by the norm) using an appropriate network (either shallow or deep) with ϵ^{-d} parameters.*

Thus, the effective dimension for the class W_m^n is n/m, that of $W_m^{n,2}$ is $2/m$. Notice that these results on deep networks imply that the function class they represent as a hypothesis space has low covering numbers.

4 General compositionality results: functions composed by a hierarchy of functions with bounded effective dimensionality

The main class of functions we considered in previous papers consists of functions as in Figure 1 b that we called *compositional functions*. The term "compositionality" was used with the meaning it has in language and vision, where higher level concepts are composed of a small number of lower level ones, objects are composed of parts, sentences are composed of words and words are composed of syllables. Notice that this meaning of compositionality is narrower than the mathematical meaning of composition of functions. The *compositional functions* we have described in previous papers may be more precisely called *functions composed of hierarchically local functions*.

Here we generalize formally our previous results to the broader class of compositional functions (beyond the hierarchical locality of Figure 1b to Figure 1c and Figure 2) by restating formally a few comments of previous papers. Let us begin with one of the previous examples. Consider

$$\begin{aligned} Q(x,y) \;=\; & (Ax^2y^2 + Bx^2y \\ & + Cxy^2 + Dx^2 + 2Exy \\ & + Fy^2 + 2Gx + 2Hy + I)^{2^{10}}. \end{aligned}$$

Since Q is nominally a polynomial of coordinatewise degree 2^{11}, [27, Lemma 3.2] shows that a shallow network with $2^{11} + 1$ units is able to approximate Q arbitrarily well on I^2. However, because of the hierarchical structure of Q, [27, Lemma 3.2] shows also that a hierarchical network with 9 units can approximate the quadratic expression, and 10 further layers, each with 3 units can approximate the successive powers. Thus, a hierarchical network with 11 layers and 39 units can approximate Q arbitrarily well. We note that even if Q is nominally of degree 2^{11}, each of the monomial coefficients in Q is a function of only 9 variables, $A, \cdots, I$.

A different example is

$$Q(x, y) = |x^2 - y^2|. \tag{8}$$

This is obviously a Lipschitz continuous function of 2 variables. The effective dimension of this class is 2, and hence, a shallow network would require at least $c\epsilon^{-2}$ parameters to approximate it within ϵ. However, the effective dimension of the class of univariate Lipschitz continuous functions is 1. Hence, if we take into account the fact that Q is a composition of a polynomial of degree 2 in 2 variables and the univariate Lipschitz continuous function $t \mapsto |t|$, then it is easy to see that the same approximation can be achieved by using a two layered network with $\mathcal{O}(\epsilon^{-1})$ parameters.

To formulate our most general result that includes the examples above as well as the constraint of hierarchical locality, we first define formally a compositional function in terms of a directed acyclic graph. Let $\mathcal{G}$ be a directed acyclic graph (DAG), with the set of nodes V. A $\mathcal{G}$–function is defined as follows. Each of the source node obtains an input from $\mathbb{R}$. Each in-edge of every other node represents an input real variable, and the node itself represents a function of these input real variables, called a *constituent function*. The out-edges fan out the result of this evaluation. We assume that there is only one sink node, whose output is the $\mathcal{G}$-function. Thus, ignoring the compositionality of this function, it is a function of n variables, where n is the number of source nodes in $\mathcal{G}$.

Theorem 3. *Let $\mathcal{G}$ be a DAG, n be the number of source nodes, and for each $v \in V$, let d_v be the number of in-edges of v. Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a compositional $\mathcal{G}$-function, where each of the constitutent function is in $W_{m_v}^{d_v}$. Consider shallow and deep networks with infinitely smooth activation function as in Theorem 1. Then deep networks – with an associated graph that corresponds to the graph of f – avoid the curse of dimensionality in approximating f for increasing n, whereas shallow networks cannot directly avoid the curse. In particular, the complexity of the best approximating shallow network is exponential in n*

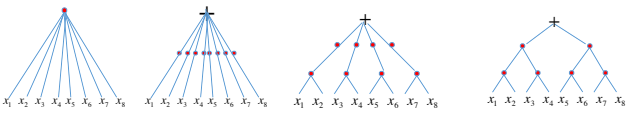

Figure 2 The figure shows the graphs of functions that may have small effective dimensionality, depending on the number of units per node required for good approximation.

$$N_s = \mathcal{O}(\epsilon^{-\frac{n}{m}}), \tag{9}$$

where $m = \min_{v \in V} m_v$*, while the complexity of the deep network is*

$$N_d = \mathcal{O}(\sum_{v \in V} \epsilon^{-d_v/m_v}). \tag{10}$$

Following definition 1 we call d_v/m_v the *effective dimension* of function v. Then, deep networks can avoid the curse of dimensionality if the constituent functions of a compositional function have a small effective dimension; i.e., have fixed, "small" dimensionality or fixed, "small" "roughness. A different interpretation of Theorem 3 is the following.

Proposition 1. *If a family of functions* $f : \mathbb{R}^n \mapsto \mathbb{R}$ *of smoothness* m *has an effective dimension* $< n/m$*, then the functions are compositional in a manner consistent with the estimates in Theorem 3.*

Notice that the functions included in this theorem are functions that are either local or the composition of simpler functions or both. Figure 2 shows some examples in addition to the examples at the top of Figure Figure 1.

As before, there is a simple corollary for polynomial functions:

Corollary 3. *Let* $\sigma : R \to R$ *be infinitely differentiable, and not a polynomial. With the set up as in Theorem 3, let* f *be DAG polynomial; i.e., a DAG function, each of whose constituent functions is a polynomial of degree* k*. Then* f *can be represented by a deep network with* $\mathcal{O}(|V_N|k^d)$ *units, where* $|V_N|$ *is the number of non-leaf vertices, and* d *is the maximal indegree of the nodes.*

For example, if $\mathcal{G}$ is a full binary tree with 2^n leaves, then the nominal degree of the $\mathcal{G}$ polynomial as in Corollary 3 is k^{k^n}, and therefore requires a shallow network with $\mathcal{O}(k^{2k^n})$ units, while a deep network requires only $\mathcal{O}(nk^2)$ units.

Notice that polynomials in S_k^n are *sparse* with a number of terms which is not exponential in n, that is it is not $O(k^n)$ but linear in n (that is $O(nk)$) or at most polynomial in n.

4.1 Approximation results for shallow and deep networks with (non-smooth) ReLUs

The results we described so far use smooth activation functions. We already mentioned why relaxing the smoothness assumption should not change our results in a fundamental way. While studies on the properties of neural networks with smooth activation abound, the results on non-smooth activation functions are much more sparse. Here we briefly recall some of them.

In the case of shallow networks, the condition of a smooth activation function can be relaxed to prove density (see [13], Proposition 3.7):

Proposition 2. *Let* $\sigma =: \mathbb{R} \to \mathbb{R}$ *be in* $\mathcal{C}^0$*, and not a polynomial. Then shallow networks are dense in* $\mathcal{C}^0$*.*

In particular, ridge functions using ReLUs of the form $\sum_{i=1}^r c_i(\langle w_i, x\rangle + b_i)_+$, with $w_i, x \in \mathbb{R}^n$, $c_i, b_i \in \mathbb{R}$ are dense in $\mathcal{C}$.

Networks with non-smooth activation functions are expected to do relatively poorly in approximating smooth functions such as polynomials in the sup norm. "Good" degree of approximation rates (modulo a constant) have been proved in the L_2 norm. Define $\mathcal{B}$ the unit ball in $\mathbb{R}^n$. Call $C^m(B^n)$ the set of all continuous functions with continuous derivative up to degree m defined on the unit ball. We define the Sobolev space W_p^m as the completion of $C^m(B^n)$ with respect to the Sobolev norm p (see for details [13] page 168). We define the space $\mathcal{B}_p^m = \{f : f \in W_p^m, \|f\|_{m,p} \leq 1\}$ and the approximation error $E(B_2^m; H; L_2) = \inf_{g \in H} \|f - g\|_{L_2}$. It is shown in [13, Corollary 6.10] that

Proposition 3. *For* $M^r : f(x) = \sum_{i=1}^r c_i(\langle w_i, x\rangle + b_i)_+$ *it holds* $E(B_2^m; M_r; L_2) \leq Cr^{-\frac{m}{n}}$ *for* $m = 1, \cdots, \frac{n+3}{2}$*.*

These approximation results with respect to the L^2 norm cannot be applied to derive bounds for compositional networks. Indeed, in the latter case, as we remarked already, estimates in the uniform norm are needed to control the propagation of the errors from one layer to the next, see Theorem 2. Results in this direction are given in [30], and more recently in [31] and [5] (see Theorem 3.1). In particular, using a result in [31] and following the proof strategy of Theorem 2 it is possible to derive the following results on the approximation of Lipshitz continuous functions with deep and shallow ReLU networks that mimics our Theorem 2:

Theorem 4. *Let* f *be a* L*-Lipshitz continuous function of* n *variables. Then, the complexity of a network which is a linear combination of ReLU providing an approximation with accuracy at least* ϵ *is*

$$N_s = \mathcal{O}\left(\left(\frac{\epsilon}{L}\right)^{-n}\right),$$

wheres that of a deep compositional architecture is

$$N_d = \mathcal{O}\left((n-1)(\frac{\epsilon}{L})^{-2}\right).$$

Our general Theorem 3 can be extended in a similar way. Theorem 4 is an example of how the analysis of smooth activation functions can be adapted to ReLU. Indeed, it shows how deep compositional networks with standard ReLUs can avoid the curse of dimensionality. In the above results, the regularity of the function class is quantified by the magnitude of Lipshitz constant. Whether the latter is the best notion of smoothness for ReLU based networks, and if the above estimates can be improved, are interesting questions that we defer to a future work. A result that is more intuitive and may reflect what networks actually do is described in Appendix 4.3. Though the construction described there provides approximation in the L_2 norm but not in the sup norm, this is not a problem under any discretization of real number required for computer simulations (see Appendix).

Figures 3, 4, 5, 6 provide a sanity check and empirical support for our main results and for the claims in the introduction.

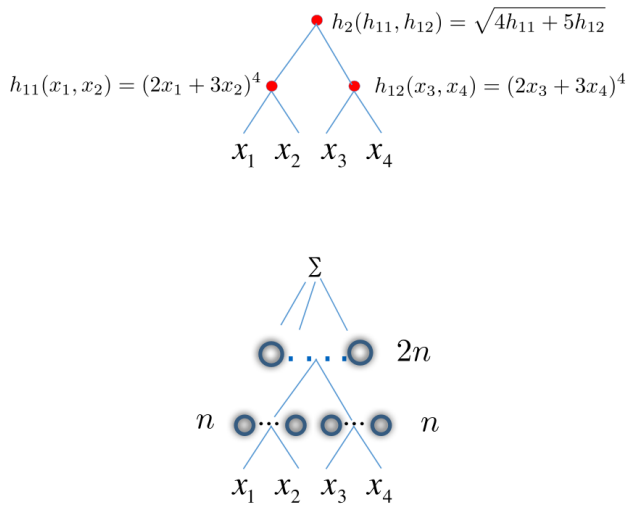

Figure 3 The figure shows on the top the graph of the function to be approximated, while the bottom part of the figure shows a deep neural network with the same graph structure. The left and right node inf the first layer has each n units giving a total of $2n$ units in the first layer. The second layer has a total of $2n$ units. The first layer has a convolution of size n to mirror the structure of the function to be learned. The compositional function we approximate has the form $f(x_1, x_2, x_3, x_4) = h_2(h_{11}(x_1, x_2), h_{12}(x_3, x_4))$ with h_{11}, h_{12} and h_2 as indicated in the figure.

4.2 Lower bounds and gaps

So far we have shown that there are deep networks – for instance of the convolutional type – that can *avoid the curse of dimensionality* if the functions they are learning are *blessed with compositionality*. There are no similar guarantee for shallow networks: for shallow networks approximating generic continuous functions the lower and the upper bound are both exponential[13]. From the point of view of machine learning, it is obvious that shallow networks, unlike deep ones, cannot exploit in their architecture the reduced number of parameters associated with priors corresponding to compositional functions. In past papers we listed a few examples, some of which are also valid lower bounds from the point of view of approximation theory:

- The polynomial considered earlier

  $$Q(x_1, x_2, x_3, x_4) = (Q_1(Q_2(x_1, x_2), Q_3(x_3, x_4)))^{1024},$$

  can be approximated by deep networks with a smaller number of parameters than shallow networks is based on polynomial approximation of functions of the type $g(g(g()))$. Here, however, a formal proof of the impossibility of good approximation by shallow networks is not available. For a lower bound we need at least one example of a compositional function which cannot be approximated by shallow networks with a non-exponential degree of approximation.

- Such an example, for which a proof of the lower bound exists since a few decades, consider a function which is a linear combination of n tensor product Chui–Wang spline wavelets, where each wavelet is a tensor product cubic spline. It is shown in [11, 12] that is impossible to implement such a function using a shallow neural network with a sigmoidal activation function using $\mathcal{O}(n)$ neurons, but a deep network with the activation function $(x_+)^2$ can do so. In this case, as we mentioned, there is a formal proof of a gap between deep and shallow networks. Similarly, Eldan and Shamir [35] show other cases with separations that are exponential in the input dimension.

- As we mentioned earlier, Telgarsky proves an exponential gap between certain functions produced by deep networks and their approximation by shallow networks. The theorem [25] can be summarized as saying that *a certain family of classification problems with real-valued inputs cannot be approximated well by shallow networks with fewer than exponentially many nodes whereas a deep network achieves zero error*. His upper bound can be proved directly from our main theorem by considering a different function – the real-valued polynomial $x_1 x_2 ... x_d$ defined on the cube $(-1, 1)^d$ which can be seen as a compositional function with a binary tree graph.

- We exhibit here another example to illustrate a limitation of shallow networks in approximating a compositional function. Let $n \geq 2$ be an integer, $B \subset \mathbb{R}^n$ be the unit ball of $\mathbb{R}^n$. We consider the class W of all compositional functions $f = f_2 \circ f_1$, where $f_1 : \mathbb{R}^n \to \mathbb{R}$, and $\sum_{|\mathbf{k}| \leq 4} \|D^{\mathbf{k}} f_1\|_\infty \leq 1$, $f_2 : \mathbb{R} \to \mathbb{R}$ and $\|D^4 f_2\|_\infty \leq 1$. We consider

  $$\Delta(\mathcal{A}_N) := \sup_{f \in W} \inf_{P \in \mathcal{A}_N} \|f - P\|_{\infty, B},$$

  where $\mathcal{A}_N$ is either the class $\mathcal{S}_N$ of all shallow networks with N units or $\mathcal{D}_N$ of deep networks with two layers, the first with n inputs, and the next with one input. The both cases, the activation function is a C^∞ function $\sigma : \mathbb{R} \to \mathbb{R}$ that is not a polynomial.

  Theorem 5. *There exist constants $c_1 > 0$ such that for $N \geq c_1$,*

  $$\Delta(\mathcal{S}_N) \geq \lceil 2^{-N/(n-1)} \rceil, \tag{11}$$

  In contrast, there exists $c_3 > 0$ such that

  $$\Delta(\mathcal{D}_N) \leq c_3 N^{-4/n}. \tag{12}$$

  The constants c_1, c_2, c_3 may depend upon n.

  PROOF. The estimate (12) follows from the estimates already given for deep networks. To prove (11), we use Lemma 3.2 in [12]. Let ϕ be a C^∞ function supported on $[0, 1]$, and we consider $f_N(x) = \phi(|4^N x|^2)$. We may clearly choose ϕ so that $\|f_N\|_\infty = 1$. Then it is clear that each $f_N \in W$. Clearly,

  $$\Delta(\mathcal{S}_N) \geq \inf_{P \in \mathcal{S}_N} \max_{x \in B} |f_N(x) - P(x)|. \tag{13}$$

  We choose $P^*(x) = \sum_{k=1}^N \sigma(\langle w_k^*, x \rangle + b_k^*)$ such that

  $$\inf_{P \in \mathcal{S}_N} \max_{x \in B} |f_N(x) - P(x)| \geq (1/2) \max_{x \in B} |f_N(x) - P^*(x)|. \tag{14}$$

  Since f_N is supported on $\{x \in \mathbb{R}^n : |x| \leq 4^{-N}\}$, we may imitate the proof of Lemma 3.2 in [12] with $g_k^*(t) = \sigma(t + b_k^*)$. Let $x_0 \in B$ be such that (without loss of generality) $f_N(x_0) = \max_{x \in B} |f_N(x)|$, and μ_0 be the Dirac measure supported at x_0. We group $\{w_k^*\}$ in $m = \lceil N/(n-1) \rceil$ disjoint groups of $n - 1$ vectors each. For each group, we take vectors $\{v_\ell\}$ such that v_ℓ is orthogonal to the w_k^*'s in group ℓ. The argument in the proof of Lemma 3.2 in [12] can be modified to get a measure μ with total variation 2^m such that

  $$\int_B f_N(x) d\mu(x) = \|f_N\|_\infty, \int_B g_k^*(x) d\mu(x) = 0, \; k = 1, \cdots, N.$$

  It is easy to deduce from here as in [12] using the duality principle that

  $$\max_{x \in B} |f_N(x) - P^*(x)| \geq c 2^{-m}.$$

  Together with (13) and (14), this implies (11). □

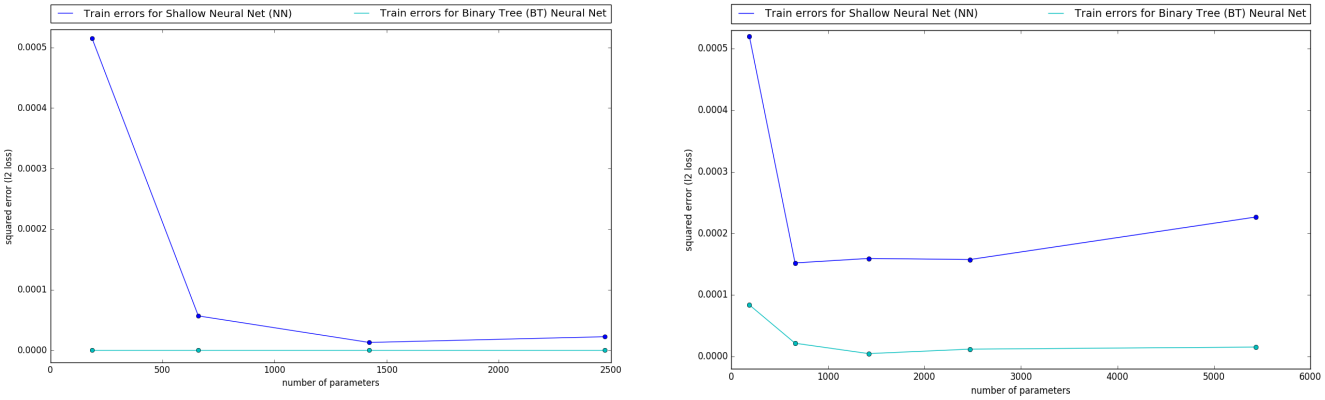

Figure 4 An empirical comparison of shallow vs 2-layers binary tree networks in the approximation of compositional functions. The loss function is the standard mean square error (MSE). There are several units per node of the tree. In our setup here the network with an associated binary tree graph was set up so that each layer had the same number of units and shared parameters. The number of units for the shallow and binary tree neural networks had the same number of parameters. On the left the function is composed of a single ReLU per node and is approximated by a network using ReLU activations. On the right the compositional function is $f(x_1, x_2, x_3, x_4) = h_2(h_{11}(x_1, x_2), h_{12}(x_3, x_4))$ and is approximated by a network with a smooth ReLU activation (also called softplus). The functions h_1, h_2, h_3 are as described in Figure 3. In order to be close to the function approximation case, a large data set of 60K training examples was used for both training sets. We used for SGD the Adam[32] optimizer. In order to get the best possible solution we ran 200 independent hyper parameter searches using random search [33] and reported the one with the lowest training error. The hyper parameters search was over the step size, the decay rate, frequency of decay and the mini-batch size. The exponential decay hyper parameters for Adam were fixed to the recommended values according to the original paper [32]. The implementations were based on TensorFlow [34].

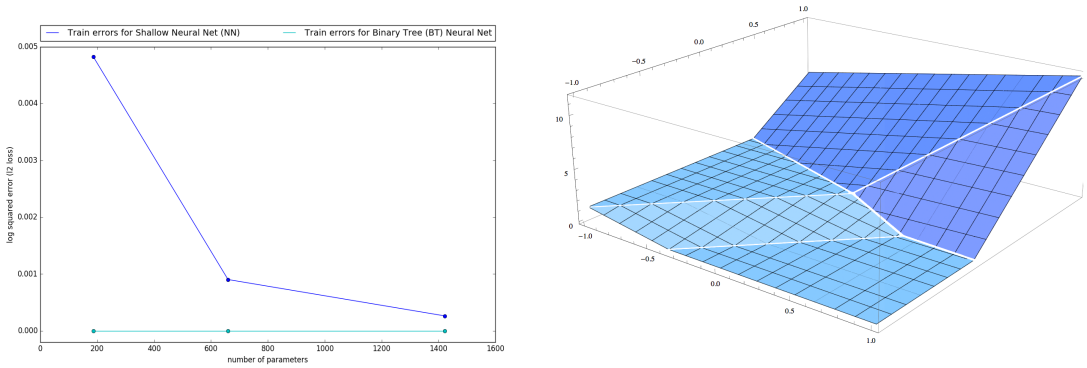

Figure 5 Another comparison of shallow vs 2-layers binary tree networks in the learning of compositional functions. The set up of the experiment was the same as in the one in figure 4 except that the compositional function had two ReLU units per node instead of only one. The right part of the figure shows a cross section of the function $f(x_1, x_2, 0.5, 0.25)$ in a bounded interval $x_1 \in [-1, 1], x_2 \in [-1, 1]$. The shape of the function is piecewise linear as it is always the case for ReLUs networks.

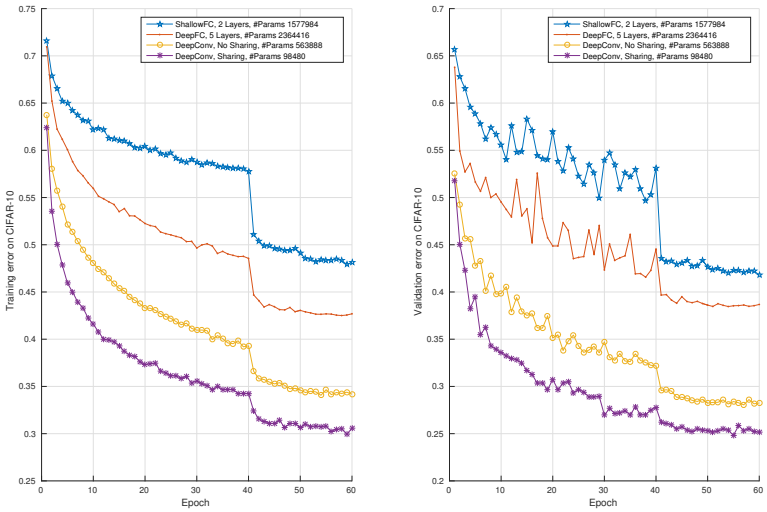

Figure 6 We show that the main advantage of deep Convolutional Networks (ConvNets) comes from "hierarchical locality" instead of weight sharing. We train two 5-layer ConvNets with and without weight sharing on CIFAR-10. ConvNet without weight sharing has different filter parameters at each spatial location. There are 4 convolutional layers (filter size 3x3, stride 2) in each network. The number of feature maps (i.e., channels) are 16, 32, 64 and 128 respectively. There is an additional fully-connected layer as a classifier. The performances of a 2-layer and 5-layer fully-connected networks are also shown for comparison. Each hidden layer of the fully-connected network has 512 units. The models are all trained for 60 epochs with cross-entropy loss and standard shift and mirror flip data augmentation (during training). The training errors are higher than those of validation because of data augmentation. The learning rates are 0.1 for epoch 1 to 40, 0.01 for epoch 41 to 50 and 0.001 for rest epochs. The number of parameters for each model are indicated in the legends. Models with hierarchical locality significantly outperform shallow and hierarchical non-local networks.

So by now plenty of examples of lower bounds exist showing a gap between shallow and deep networks. A particularly interesting case is the *product* function, that is the monomial $f(x_1, \cdots, x_n) = x_1 x_2 \cdots x_n$ which is, from our point of view, *the* prototypical compositional functions. Keeping in mind the issue of lower bounds, the question here has to do with the minimum integer $r(n)$ such that the function f is in the closure of the span of $\sigma(\langle w_k, x \rangle + b_k)$, with $k = 1, \cdots, r(n)$, and w_k, b_k ranging over their whole domains. Such a result has been claimed for the case of smooth ReLUs, using unusual group techniques and is sketched in the Appendix of [36].

Notice, in support of the conjecture, that assuming that a shallow network with (non-smooth) ReLUs has a lower bound of $r(q) = O(q)$ will lead to an apparent contradiction with Hastad theorem (which is about representation not approximation of Boolean functions) by restricting x_i from $x_i \in (-1, 1)$ to $x_i \in \{-1, +1\}$. Hastad theorem (see [37] and the earlier enabling work of [38]) establishes the inapproximability of the parity function by shallow circuits of non-exponential size.

4.3 Messy graphs and densely connected deep networks

As mentioned already, the approximating deep network does not need to exactly match the architecture of the compositional function as long as the graph or tree associated with the function is contained in the graph associated with the network. This is of course good news: the compositionality prior embedded in the architecture of the network does not to reflect exactly the graph of a new function to be learned. We have shown that for a given class of compositional functions characterized by an associated graph there exist a deep network that approximates such a function better than a shallow network. The same network approximates well functions characterized by subgraphs of the original class.

The proofs of our theorems show that linear combinations of compositional functions are *universal* in the sense that they can approximate any function and that deep networks with a number of units that increases exponentially with layers can approximate any function. Notice that deep compositional networks can interpolate if they are overparametrized with respect to the data, even if the data reflect a non-compositional function.

As an aside, note that the simplest compositional function – addition – is trivial in the sense that it offers no approximation advantage to deep networks. The key function is multiplication which is for us the prototypical compositional functions. As a consequence, *polynomial functions are compositional* – they are linear combinations of monomials which are compositional. However, their compositional structure does not confer any advantage in terms of approximation, because of the exponential number of compositional terms.

As we mentioned earlier, networks corresponding to graphs that include the graph of the function to be learned can exploit compositionality. The relevant number of parameters to be optimized, however, is the number of parameters r in the network and not the number of parameters $r*$ ($r* < r$) of the optimal deep network with a graph exactly matched to the graph of the function to be learned. As an aside, the price to be paid in using a non-optimal prior depend on the learning algorithm. For instance, under sparsity constraints it may be possible to pay a smaller price than r (but higher than r^*).

In this sense, some of the densely connected deep networks used in practice – which contain sparse graphs possibly relevant for the function to be learned and which are still "smaller" than the exponential number of units required to represent a generic function of n variables – may be capable in some cases of exploiting an underlying compositionality structure without paying an exhorbitant price in terms of required complexity.

5 Connections with the theory of Boolean functions

The approach followed in our main theorems suggest the following considerations (see Appendix 1 for a brief introduction). The structure of a deep network is reflected in polynomials that are best approximated by it – for instance generic polynomials or sparse polynomials (in the coefficients) in d variables of order k. The tree structure of the nodes of a deep network reflects the structure of a specific sparse polynomial. Generic polynomial of degree k in d variables are difficult to learn because the number of terms, trainable parameters and associated VC-dimension are all exponential in d. On the other hand, functions approximated well by sparse polynomials can be learned efficiently by deep networks with a tree structure that matches the polynomial. We recall that in a similar way several properties of certain Boolean functions can be "read out" from the terms of their Fourier expansion corresponding to "large" coefficients, that is from a polynomial that approximates well the function.

Classical results [37, 38] about the depth-breadth tradeoff in circuits design show that deep circuits are more efficient in representing certain Boolean functions than shallow circuits. Hastad proved that highly-variable functions (in the sense of having high frequencies in their Fourier spectrum), in particular the parity function cannot even be decently approximated by small constant depth circuits (see also [39]).

Notice that Hastad's results on Boolean functions have been often quoted in support of the claim that deep neural networks can represent functions that shallow networks cannot. For instance Bengio and LeCun [40] write *"We claim that most functions that can be represented compactly by deep architectures cannot be represented by a compact shallow architecture"."*.

Finally, we want to mention a few other observations on Boolean functions that shows an interesting connection with our approach. It is known that within Boolean functions the AC^0 class of polynomial size constant depth circuits is characterized by Fourier transforms where most of the power spectrum is in the low order coefficients. Such functions can be approximated well by a polynomial of low degree and can be learned well by considering only such coefficients. There are two algorithms [41] that allow learning of certain Boolean function classes:

1. the low order algorithm that approximates functions by considering their low order Fourier coefficients and
2. the sparse algorithm which learns a function by approximating its significant coefficients.

Decision lists and decision trees can be learned by the first algorithm. Functions with small L_1 norm can be approximated well by the second algorithm. Boolean circuits expressing DNFs can be approximated by the first one but even better by the second. In fact, in many cases a function can be approximated by a small set of coefficients but these coefficients do not correspond to low-order terms. All these cases are consistent with the notes about sparse functions in section 6.

6 Notes on a theory of compositional computation

The key property of the theory of compositional functions sketched here is that certain deep networks can learn them avoiding the curse of dimensionality because of the blessing of *compositionality* via a small *effective dimension*.

We state here several comments and conjectures.

1. *General comments*

   - *Properties of the compositionality type may have a more significant impact than smoothness properties in countering the curse of dimensionality in practical cases of learning and approximation.*
   - The estimates on the n–width imply that there is some function in either W_m^n (theorem 1) or $W_m^{n,2}$ (theorem 2) for which the approximation cannot be better than that suggested by the theorems.
   - The main question that may be asked about the relevance of the theoretical results of this paper and networks used in practice has to do with the many "channels" used in the latter and with our assumption that each node in the networks computes a scalar function – the linear combination of r units (Equation 3). The following obvious but interesting extension of Theorem 1 to vector-valued functions says that the number of hidden units required for a given accuracy in each component of the function is the same as in the scalar case considered in our theorems (of course the number of weigths is larger):

     Corollary 4. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be infinitely differentiable, and not a polynomial. For a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^q$ with components $f_i \in W_m^n$, $i = 1, \cdots, q$ the number of hidden units in shallow networks with n inputs, q outputs that provide accuracy at least ϵ in each of the components of f is*

     $$N = \mathcal{O}(\epsilon^{-n/m}) \ . \tag{15}$$

     The demonstration follows the proof of theorem 1, see Figure 7 and also Appendix 4.1. It amounts to realizing that the hidden units (or linear combinations of them) can be equivalent to the monomials of a generic polynomial of degree k in n variables that can be used by a different set of coefficients for each of the f_i. This argument of course does not mean that during learning this is what happens; it provides one way to perform the approximation and an associated upper bound. The corollary above leads to a simple argument that *generalizes our binary tree results to standard, multi-channel deep convolutional networks* by introducing a set of virtual linear units as ouputs of one layer and inputs of the next one. This in turn leads to the following *prediction*: for consistent approximation accuracy across the layers, the rank of the weights matrices between units in successive layers should be in the order of the number of the dimensionality in the first layer (inputs and outputs have to be defined wrt support of the convolution kernel). This suggests rank-deficient weight matrices in present networks.
   - We have used polynomials (but see Appendix 4.3) to prove results about complexity of approximation in the case of neural networks. Neural network learning with SGD may or may not synthesize polynomial, depending on the smoothness of the activation function and on the target. This is not a problem for theoretically establishing upper bounds on the degree of convergence because results using the framework on nonlinear width guarantee the "polynomial" bounds are optimal.
   - Both shallow and deep representations may or may not reflect invariance to group transformations of the inputs of the function ([42, 22]). Invariance – also called

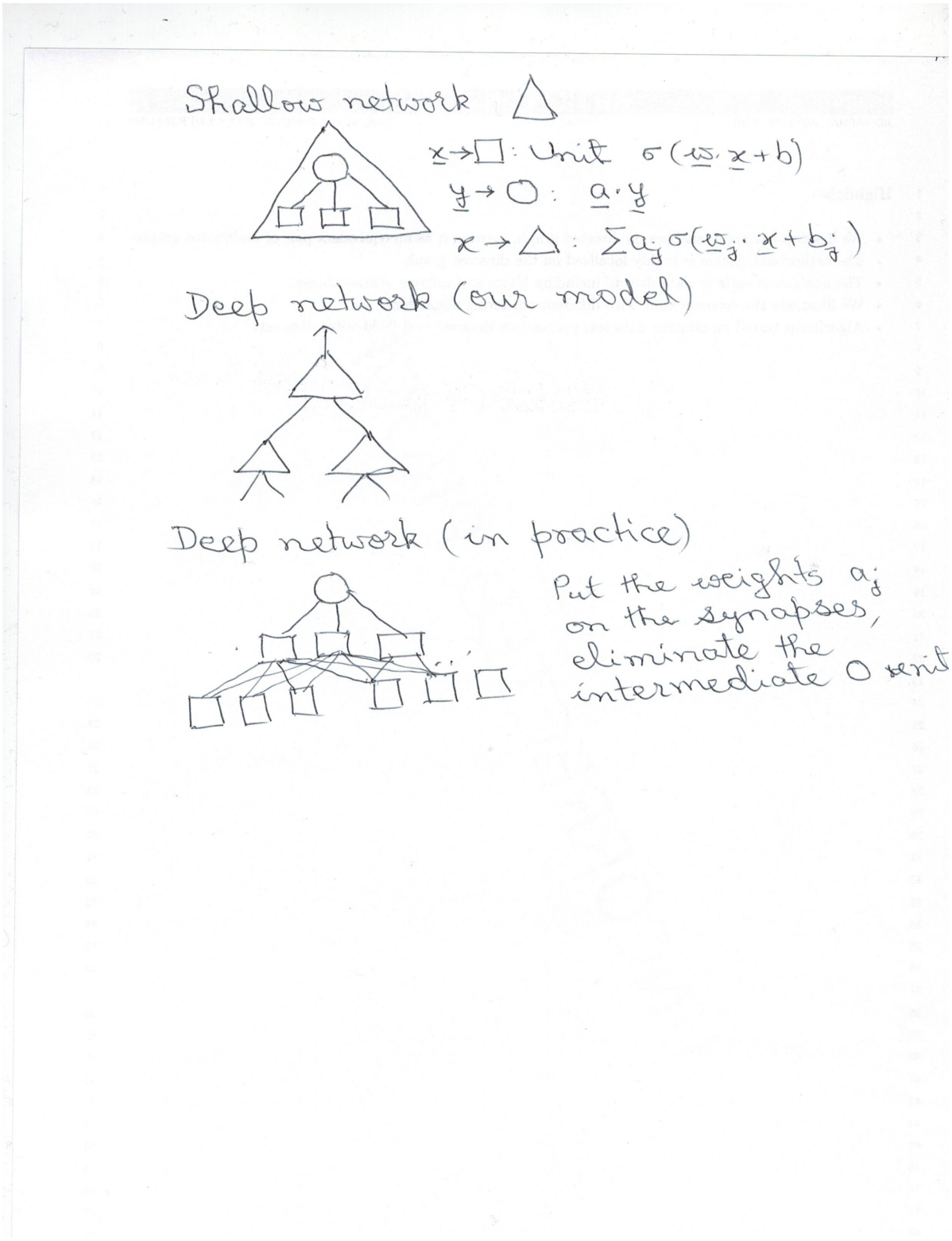

Figure 7 An explanation of the connection between the network used in the proofs in Theory I and the networks (bottom) used in practice.

weight sharing – decreases the complexity of the network. Since we are interested in the comparison of shallow vs deep architectures, we have considered the generic case of networks (and functions) for which invariance is not assumed. In fact, the key advantage of deep vs. shallow network – as shown by the proof of the theorem – is the associated hierarchical locality (the constituent functions in each node are local that is have a small dimensionality) and *not invariance* (which designates shared weights that is nodes at the same level sharing the same function). One may then ask about the relation of these results with i-theory[43]. The original core of i-theory describes how pooling can provide either shallow or deep networks with invariance and selectivity properties. Invariance of course helps but not exponentially as hierarchical locality does.

- There are several properties that follow from the theory here which are attractive from the point of view of neuroscience. A main one is the robustness of the results with respect to the choice of nonlinearities (linear rectifiers, sigmoids, Gaussians etc.) and pooling.
- In a machine learning context, minimization over a training set of a loss function such as the square loss yields an empirical approximation of the regression function $p(y/x)$. Our hypothesis of compositionality becomes an hypothesis about the structure of the conditional probability function.

2. *Spline approximations, Boolean functions and tensors*

   - Consider again the case of section 4 of a multivariate function $f : [0, 1]^d \to \mathbb{R}$. Suppose to discretize it by a set of piecewise constant splines and their tensor products. Each coordinate is effectively replaced by n boolean variables.This results in a d-dimensional table with $N = n^d$ entries. This in turn corresponds to a boolean function $f : \{0, 1\}^N \to \mathbb{R}$. Here, the assumption of compositionality corresponds to compressibility of a d-dimensional table in terms of a hierarchy of $d - 1$ 2-dimensional tables. Instead of n^d entries there are $(d - 1)n^2$ entries. This has in turn obvious connections with HVQ (Hierarchical Vector Quantization), discussed in Appendix 5.
   - As Appendix 4.3 shows, every function f can be approximated by an epsilon-close binary function f_B. Binarization of $f : \mathbb{R}^n \to \mathbb{R}$ is done by using k partitions for each variable x_i and indicator functions. Thus $f \mapsto f_B : \{0, 1\}^{kn} \to \mathbb{R}$ and $sup|f - f_B| \leq \epsilon$, with ϵ depending on k and bounded Df.
   - f_B can be written as a polynomial (a Walsh decomposition) $f_B \approx p_B$. It is always possible to associate a p_b to any f, given ϵ.
   - The binarization argument suggests a direct way to connect results on function approximation by neural nets with older results on Boolean functions. The latter are special cases of the former results.
   - One can think about tensors in terms of d-dimensional tables. The framework of hierarchical decompositions of tensors – in particular the *Hierarchical Tucker format* – is closely connected to our notion of compositionality. Interestingly, the hierarchical Tucker decomposition has been the subject of recent papers on Deep Learning (for instance see [20]). This work, as well more classical papers [44], does not characterize directly the class of functions for which these decompositions are effective. Notice that tensor decompositions *assume* that the sum of polynomial functions of order d is sparse (see eq. at top of page 2030 of [44]). Our results provide a rigorous grounding for the tensor work related to deep learning. There is obviously a wealth of interesting connections with approximation theory that should be explored. Notice that the notion of *separation rank* of a tensor is very closely related to the effective r in Equation 25.

3. *Sparsity*

   - We suggest to define binary sparsity of f, in terms of the sparsity of the boolean function p_B; binary sparsity implies that an approximation to f can be learned by non-exponential deep networks via binarization. Notice that if the function f is compositional the associated Boolean functions f_B is sparse; the converse is not true.
   - In many situations, Tikhonov regularization corresponds to *cutting high order Fourier coefficients*. *Sparsity* of the coefficients subsumes Tikhonov regularization in the case of a Fourier representation. Notice that as an effect the number of Fourier coefficients is reduced, that is trainable parameters, in the approximating trigonometric polynomial. *Sparsity of Fourier coefficients* is a general constraint for learning Boolean functions.
   - *Sparsity in a specific basis.* A set of functions may be defined to be *sparse in a specific basis* when the number of parameters necessary for its ϵ-approximation increases less than exponentially with the dimensionality. An open question is the appropriate definition of sparsity. The notion of sparsity we suggest here is the effective r in Equation 25. For a general function $r \approx k^n$; we may define sparse functions those for which $r << k^n$ in

     $$f(x) \approx P_k^*(x) = \sum_{i=1}^{r} p_i(\langle w_i, x \rangle). \qquad (16)$$

     where P^* is a specific polynomial that approximates $f(x)$ within the desired ϵ. Notice that the polynomial P_k^* can be a sum of monomials or a sum of, for instance, orthogonal polynomials with a total of r parameters. In general, sparsity depends on the basis and one needs to know the basis and the type of sparsity to exploit it in learning, for instance with a deep network with appropriate activation functions and architecture. There are function classes that are sparse in every bases. Examples are compositional functions described by a binary tree graph.

4. *Deep Networks as memories*

   Notice that independently of considerations of generalization, deep compositional networks are expected to be *very efficient memories* – in the spirit of hierarchical vector quantization – for associative memories reflecting compositional rules (see Appendix 5 and [45]). Notice that the advantage with respect to shallow networks from the point of view of memory capacity can be exponential (as in the example after Equation 31 showing $m_{shallow} \approx 10^{10^4} m_{deep}$).

5. *Theory of computation, locality and compositionality*

   - From the computer science point of view, feedforward multilayer networks are equivalent to finite state machines running for a finite number of time steps[46, 47]. This result holds for almost any fixed nonlinearity in each layer. Feedforward networks are equivalent to cascades without loops (with a finite number of stages) and all other forms of loop free cascades (i.e. McCulloch-Pitts nets without loops, perceptrons, analog perceptrons, linear threshold machines). Finite state machines, cascades with loops, and difference equation systems which are Turing equivalent, are more powerful than multilayer architectures with a finite number of layers.

The latter networks, however, are practically universal computers, since every machine we can build can be approximated as closely as we like by defining sufficiently many stages or a sufficiently complex single stage. Recurrent networks as well as differential equations are Turing universal.

In other words, all computable functions (by a Turing machine) are recursive, that is composed of a small set of primitive operations. In this broad sense all computable functions are compositional (composed from elementary functions). Conversely a Turing machine can be written as a compositional function $y = f^{(t}(x, p)$ where $f : Z^n \times P^m \mapsto Z^h \times P^k$, P being parameters that are inputs and outputs of f. If t is bounded we have a finite state machine, otherwise a Turing machine. in terms of elementary functions. As mentioned above, each layer in a deep network correspond to one time step in a Turing machine. In a sense, this is sequential compositionality, as in the example of Figure 1 c. The hierarchically local compositionality we have introduced in this paper has the flavour of compositionality in space.

Of course, since any function can be approximated by polynomials, and a polynomial can always be calculated using a recursive procedure and a recursive procedure can always be unwound as a "deep network", any function can always be approximated by a compositional function of a few variables. However, generic compositionality of this type does not guarantee good approximation properties by deep networks.

- Hierarchically local compositionality can be related to the notion of *local connectivity* of a network. Connectivity is a key property in network computations. Local processing may be a key constraint also in neuroscience. One of the natural measures of connectivity that can be introduced is the *order* of a node defined as *the number of its distinct inputs*. The *order of a network is then the maximum order among its nodes*. The term order dates back to the Perceptron book ([48], see also [47]). From the previous observations, it follows that a *hierarchical network of order at least* 2 *can be universal.* In the *Perceptron* book many interesting visual computations have low order (e.g. recognition of isolated figures). The message is that they can be implemented in a single layer by units that have a small number of inputs. More complex visual computations require inputs from the full visual field. A hierarchical network can achieve effective high order at the top using units with low order. The network architecture of Figure 1 b) has low order: each node in the intermediate layers is connected to just 2 other nodes, rather than (say) all nodes in the previous layer (notice that the connections in the trees of the figures may reflect linear combinations of the input units).

- Low order may be a key constraint for cortex. If it captures what is possible in terms of connectivity between neurons, it may determine by itself the hierarchical architecture of cortex which in turn may impose compositionality to language and speech.

- The idea of functions that are compositions of "simpler" functions extends in a natural way to recurrent computations and recursive functions. For instance $h(f^{(t)}g((x)))$ represents t iterations of the algorithm f (h and g match input and output dimensions to f).

7 Why are compositional functions so common?

Let us provide a couple of simple examples of compositional functions. Addition is compositional but the degree of approximation does not improve by decomposing addition in different layers of a network; all linear operators are compositional with no advantage for deep networks; multiplication as well as the AND operation (for Boolean variables) is the prototypical compositional function that provides an advantage to deep networks. So compositionality is not enough: we need certain sublasses of compositional functions (such as the hierarchically local functions we described) in order to avoid the curse of dimensionality.

It is not clear, of course, why problems encountered in practice should match this class of functions. Though we and others have argued that the explanation may be in either the physics or the neuroscience of the brain, these arguments (see Appendix 2) are not (yet) rigorous. Our conjecture at present is that compositionality is imposed by the wiring of our cortex and is reflected in language and the common problems we worry about. Thus compositionality of several – but not all – computations on images many reflect the way we describe and think about them.

Acknowledgment

This work was supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF – 1231216. HNM was supported in part by ARO Grant W911NF-15-1-0385. We thank O. Shamir for useful emails that prompted us to clarify our results in the context of lower bounds and for pointing out a number of typos and other mistakes. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the DGX-1 used for this research.

References

[1] F. Anselmi, L. Rosasco, C. Tan, and T. Poggio, "Deep convolutional network are hierarchical kernel machines," *Center for Brains, Minds and Machines (CBMM) Memo No. 35, also in arXiv*, 2015.

[2] T. Poggio, L. Rosasco, A. Shashua, N. Cohen, and F. Anselmi, "Notes on hierarchical splines, dclns and i-theory," tech. rep., MIT Computer Science and Artificial Intelligence Laboratory, 2015.

[3] T. Poggio, F. Anselmi, and L. Rosasco, "I-theory on depth vs width: hierarchical function composition," *CBMM memo 041*, 2015.

[4] H. Mhaskar, Q. Liao, and T. Poggio, "Learning real and boolean functions: When is deep better than shallow?," *Center for Brains, Minds and Machines (CBMM) Memo No. 45, also in arXiv*, 2016.

[5] H. Mhaskar and T. Poggio, "Deep versus shallow networks: an approximation theory perspective," *Center for Brains, Minds and Machines (CBMM) Memo No. 54, also in arXiv*, 2016.

[6] D. L. Donoho, "High-dimensional data analysis: The curses and blessings of dimensionality," in *AMS CONFERENCE ON MATH CHALLENGES OF THE 21ST CENTURY*, 2000.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, pp. 436–444, 2015.

[8] K. Fukushima, "Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[9] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, pp. 1019–1025, Nov. 1999.

[10] H. Mhaskar, "Approximation properties of a multilayered feedforward artificial neural network," *Advances in Computational Mathematics*, pp. 61–80, 1993.

[11] C. Chui, X. Li, and H. Mhaskar, "Neural networks for localized approximation," *Mathematics of Computation*, vol. 63, no. 208, pp. 607–623, 1994.

[12] C. K. Chui, X. Li, and H. N. Mhaskar, "Limitations of the approximation capabilities of neural networks with one hidden layer," *Advances in Computational Mathematics*, vol. 5, no. 1, pp. 233–243, 1996.

[13] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta Numerica*, vol. 8, pp. 143–195, 1999.

[14] T. Poggio and S. Smale, "The mathematics of learning: Dealing with data," *Notices of the American Mathematical Society (AMS)*, vol. 50, no. 5, pp. 537–544, 2003.

[15] B. B. Moore and T. Poggio, "Representations properties of multilayer feedforward networks," *Abstracts of the First annual INNS meeting*, vol. 320, p. 502, 1998.

[16] R. Livni, S. Shalev-Shwartz, and O. Shamir, "A provably efficient algorithm for training deep networks," *CoRR*, vol. abs/1304.7045, 2013.

[17] O. Delalleau and Y. Bengio, "Shallow vs. deep sum-product networks," in *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pp. 666–674, 2011.

[18] R. Montufar, G. F.and Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 27, pp. 2924–2932, 2014.

[19] H. N. Mhaskar, "Neural networks for localized approximation of real functions," in *Neural Networks for Processing [1993] III. Proceedings of the 1993 IEEE-SP Workshop*, pp. 190–196, IEEE, 1993.

[20] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: a tensor analysis," *CoRR*, vol. abs/1509.0500, 2015.

[21] F. Anselmi, J. Leibo, L. Rosasco, J. Mutch, A. Tacchetti, and T. Poggio, "Unsupervised learning of invariant representations with low sample complexity: the magic of sensory cortex or a new framework for machine learning?.," *Center for Brains, Minds and Machines (CBMM) Memo No. 1. arXiv:1311.4158v5*, 2014.

[22] F. Anselmi, J. Z. Leibo, L. Rosasco, J. Mutch, A. Tacchetti, and T. Poggio, "Unsupervised learning of invariant representations," *Theoretical Computer Science*, 2015.

[23] T. Poggio, L. Rosaco, A. Shashua, N. Cohen, and F. Anselmi, "Notes on hierarchical splines, dclns and i-theory," *CBMM memo 037*, 2015.

[24] Q. Liao and T. Poggio, "Bridging the gap between residual learning, recurrent neural networks and visual cortex," *Center for Brains, Minds and Machines (CBMM) Memo No. 47, also in arXiv*, 2016.

[25] M. Telgarsky, "Representation benefits of deep feedforward networks," *arXiv preprint arXiv:1509.08101v2 [cs.LG] 29 Sep 2015*, 2015.

[26] I. Safran and O. Shamir, "Depth separation in relu networks for approximating smooth non-linear functions," *arXiv:1610.09887v1*, 2016.

[27] H. N. Mhaskar, "Neural networks for optimal approximation of smooth and analytic functions," *Neural Computation*, vol. 8, no. 1, pp. 164–177, 1996.

[28] E. Corominas and F. S. Balaguer, "Condiciones para que una funcion infinitamente derivable sea un polinomio," *Revista matemática hispanoamericana*, vol. 14, no. 1, pp. 26–43, 1954.

[29] R. A. DeVore, R. Howard, and C. A. Micchelli, "Optimal nonlinear approximation," *Manuscripta mathematica*, vol. 63, no. 4, pp. 469–478, 1989.

[30] H. N. Mhaskar, "On the tractability of multivariate integration and approximation by neural networks," *J. Complex.*, vol. 20, pp. 561–590, Aug. 2004.

[31] F. Bach, "Breaking the curse of dimensionality with convex neural networks," *arXiv:1412.8690*, 2014.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[33] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[35] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," *arXiv preprint arXiv:1512.03965v4*, 2016.

[36] M. Lin, H.and Tegmark, "Why does deep and cheap learning work so well?," *arXiv:1608.08225*, pp. 1–14, 2016.

[37] J. T. Hastad, *Computational Limitations for Small Depth Circuits*. MIT Press, 1987.

[38] M. Furst, J. Saxe, and M. Sipser, "Parity, circuits, and the polynomial-time hierarchy," *Math. Systems Theory*, vol. 17, pp. 13–27, 1984.

[39] N. Linial, M. Y., and N. N., "Constant depth circuits, fourier transform, and learnability," *Journal of the ACM*, vol. 40, no. 3, p. 607–620, 1993.

[40] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards ai," in *Large-Scale Kernel Machines* (L. Bottou, O. Chapelle, and J. DeCoste, D.and Weston, eds.), MIT Press, 2007.

[41] Y. Mansour, "Learning boolean functions via the fourier transform," in *Theoretical Advances in Neural Computation and Learning* (V. Roychowdhury, K. Siu, and A. Orlitsky, eds.), pp. 391–424, Springer US, 1994.

[42] S. Soatto, "Steps Towards a Theory of Visual Information: Active Perception, Signal-to-Symbol Conversion and the Interplay Between Sensing and Control," *arXiv:1110.2053*, pp. 0–151, 2011.

[43] F. Anselmi and T. Poggio, *Visual Cortex and Deep Networks*. MIT Press, 2016.

[44] L. Grasedyck, "Hierarchical Singular Value Decomposition of Tensors," *SIAM J. Matrix Anal. Appl.*, no. 31,4, pp. 2029–2054, 2010.

[45] F. Anselmi, L. Rosasco, and T. Tan, C.and Poggio, "Deep Convolutional Networks are Hierarchical Kernel Machines," *Center for Brains, Minds and Machines (CBMM) Memo No. 35, also in arXiv*, 2015.

[46] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge eBooks, 2014.

[47] T. Poggio and W. Reichardt, "On the representation of multi-input systems: Computational properties of polynomial algorithms.," *Biological Cybernetics, 37, 3, 167-186.*, 1980.

[48] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge MA: The MIT Press, ISBN 0-262-63022-2, 1972.

[49] D. Ruderman, "Origins of scaling in natural images," *Vision Res.*, pp. 3385 – 3398, 1997.

[50] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *NeuralComputation*, vol. 7, pp. 219–269, 1995.

[51] T. Poggio and F. Girosi, "A theory of networks for approximation and learning," *Laboratory, Massachusetts Institute of Technology*, vol. A.I. memo n1140, 1989.

[52] J. Mihalik, "Hierarchical vector quantization. of images in transform domain.," *ELEKTROTECHN. CA5, 43, NO. 3. 92,94.*, 1992.

[53] F. Girosi and T. Poggio, "Representation properties of networks: Kolmogorov's theorem is irrelevant," *Neural Computation*, vol. 1, no. 4, pp. 465–469, 1989.

[54] F. Girosi and T. Poggio, "Networks and the best approximation property," *Biological Cybernetics*, vol. 63, pp. 169–176, 1990.

[55] M. Anthony and P. Bartlett, *Neural Network Learning - Theoretical Foundations*. Cambridge University Press, 2002.

Appendix: observations, theorems and conjectures

1 Boolean Functions

One of the most important tools for theoretical computer scientists for the study of functions of n Boolean variables, their related circuit design and several associated learning problems, is the Fourier transform over the Abelian group $\mathcal{Z}_2^n$. This is known as Fourier analysis over the Boolean cube $\{-1,1\}^n$. The Fourier expansion of a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ or even a real-valued Boolean function $f : \{-1,1\}^n \to [-1,1]$ is its representation as a real polynomial, which is multilinear because of the Boolean nature of its variables. Thus for Boolean functions their Fourier representation is identical to their polynomial representation. In this paper we use the two terms interchangeably. Unlike functions of real variables, the full finite Fourier expansion is exact, instead of an approximation. There is no need to distingush between trigonometric and real polynomials. Most of the properties of standard harmonic analysis are otherwise preserved, including Parseval theorem. The terms in the expansion correspond to the various monomials; the low order ones are parity functions over small subsets of the variables and correspond to low degrees and low frequencies in the case of polynomial and Fourier approximations, respectively, for functions of real variables.

2 Does Physics or Neuroscience imply compositionality?

It has been often argued that not only text and speech are compositional but so are images. There are many phenomena in nature that have descriptions along a range of rather different scales. An extreme case consists of fractals which are infinitely self-similar, iterated mathematical constructs. As a reminder, a self-similar object is similar to a part of itself (i.e. the whole is similar to one or more of the parts). Many objects in the real world are statistically self-similar, showing the same statistical properties at many scales: clouds, river networks, snow flakes, crystals and neurons branching. A relevant point is that the shift-invariant scalability of image statistics follows from the fact that objects contain smaller clusters of similar surfaces in a selfsimilar fractal way. Ruderman [49] analysis shows that image statistics reflects what has been known as the property of compositionality of objects and parts: parts are themselves objects, that is selfsimilar clusters of similar surfaces in the physical world. Notice however that, from the point of view of this paper, it is misleading to say that an image is compositional: in our terminology a *function on an image may be compositional but not its argument*. In fact, functions to be learned may or may not be compositional even if their input is an image since they depend on the input but also on the task (in the supervised case of deep learning networks all weights depend on x and y). Conversely, a network may be given a function which can be written in a compositional form, independently of the nature of the input vector such as the function "multiplication of all scalar inputs' components". Thus a more reasonable statement is that "many natural questions on images correspond to algorithms which are compositional". Why this is the case is an interesting open question. An answer inspired by the condition of "locality" of the constituent functions in our theorems and by the empirical success of deep convolutional networks has attracted some attention. The starting observation is that in the natural sciences– physics, chemistry, biology – many phenomena seem to be described well by processes that *that take place at a sequence of increasing scales and are local at each scale, in the sense that they can be described well by neighbor-to-neighbor interactions*.

Notice that this is a much less stringent requirement than renormalizable physical processes [36] where the *same* Hamiltonian (apart from a scale factor) is required to describe the process at each scale (in our observation above, the renormalized Hamiltonian only needs to remain local at each renormalization step)[2]. As discussed previously [3] hierarchical locality may be related to properties of basic physics that imply local interactions *at each level in a sequence of scales*, possibly different at each level. To complete the argument one would have then to assume that several different questions on sets of natural images may share some of the initial inference steps (first layers in the associated deep network) and thus share some of features computed by intermediate layers of a deep network. In any case, at least two open questions remain that require formal theoretical results in order to explain the connection between hierarchical, local functions and physics:

- can hierarchical locality be derived from the Hamiltonians of physics? In other words, under which conditions does coarse graining lead to local Hamiltonians? According to Tegmark locality of the Hamiltonian ensures locality at every stage of coarse graining.
- is it possible to formalize how and when the local hierarchical structure of *computations on images* is related to the hierarchy of local physical process that describe the physical world represented in the image?

It seems to us that the above set of arguments is unsatisfactory and unlikely to provide the answer. One of the arguments is that *iterated local functions* (from $\mathbb{R}^n$ to $\mathbb{R}^n$ with n increasing without bound) can be *Turing universal* and thus can simulate any physical phenomenon (as shown by the game *Life* which is local and Turing universal). Of course, this does not imply that the simulation will be efficient but it weakens the physics-based argument. An alternative hypothesis in fact is that locality across levels of explanation originates from the structure of the brain – wired, say, similarly to convolutional deep networks – which is then forced to use local algorithms of the type shown in Figure 19. Such local algorithms allow the organism to survive because enough of the key problems encountered during evolution can be solved well enough by them. So instead of claiming that all questions on the physical world are local because of its physics we believe that local algorithms are good enough over the distribution of evolutionary relevant problems. From this point of view locality of algorithms follows from the need to optimize local connections and to reuse computational elements. Despite the high number of synapses on each neuron it would be impossible for a complex cell to pool information across all the simple cells needed to cover an entire image, as needed by a single hidden layer network.

3 Splines: some notes

3.1 Additive and Tensor Product Splines

Additive and tensor product splines are two alternatives to radial kernels for multidimensional function approximation. It is well known that the three techniques follow from classical Tikhonov regularization and correspond to one-hidden layer networks with either the square loss or the SVM loss.

We recall the extension of classical splines approximation techniques to multidimensional functions. The setup is due to Jones et

[2] Tegmark and Lin [36] have also suggested that a sequence of generative processes can be regarded as a Markov sequence that can be inverted to provide an inference problem with a similar compositional structure. The resulting compositionality they describe does not, however, correspond to our notion of *hierarchical locality* and thus our theorems cannot be used to support their claims.

al. (1995) [50].

3.1.1 Tensor product splines

The best-known multivariate extension of one-dimensional splines is based on the use of radial kernels such as the Gaussian or the multiquadric radial basis function. An alternative to choosing a radial function is a *tensor product* type of basis function, that is a function of the form

$$K(x) = \Pi_{j=1}^{d} k(x_j)$$

where x_j is the j-th coordinate of the vector x and $k(x)$ is the inverse Fourier transform associated with a Tikhonov stabilizer (see [50]).

We notice that the choice of the Gaussian basis function for $k(x)$ leads to a Gaussian radial approximation scheme with $K(x) = e^{-\|x\|^2}$.

3.1.2 Additive splines

Additive approximation schemes can also be derived in the framework of regularization theory. With additive approximation we mean an approximation of the form

$$f(x) = \sum_{\mu=1}^{d} f_\mu(x^\mu) \tag{17}$$

where x^μ is the μ-th component of the input vector x and the f_μ are one-dimensional functions that will be defined as the *additive components* of f (from now on Greek letter indices will be used in association with components of the input vectors). Additive models are well known in statistics (at least since Stone, 1985) and can be considered as a generalization of linear models. They are appealing because, being essentially a superposition of one-dimensional functions, they have a low complexity, and they share with linear models the feature that the effects of the different variables can be examined separately. The resulting scheme is very similar to Projection Pursuit Regression. We refer to [50] for references and discussion of how such approximations follow from regularization.

Girosi et al. [50] derive an approximation scheme of the form (with i corresponding to spline knots – which are free parameters found during learning as in free knots splines - and μ corresponding to new variables as linear combinations of the original components of x):

$$f(x) = \sum_{\mu=1}^{d'} \sum_{i=1}^{n} c_i^\mu K(\langle t^\mu, x\rangle - b^\mu) = \sum_{\mu=1} \sum_{i=1} c_i^\mu K(\langle t^\mu, x\rangle - b_i^\mu)\,. \tag{18}$$

Note that the above can be called spline only with a stretch of the imagination: not only the w_μ but also the t^μ depend on the data in a very nonlinear way. In particular, the t^μ may not correspond at all to actual data point. The approximation could be called ridge approximation and is related to projection pursuit. When the basis function K is the absolute value that is $K(x-y) = |x-y|$ the network implements piecewise linear splines.

Is the approximation by deep nets more of the fixed knots or the free knots type? In the fixed knots case we expect the weights of all the layers apart the last one to depend on the x part of the examples but not on the y. The simulations shown in figure 8.

3.2 Hierarchical Splines

Consider an additive approximation scheme (see subsection) in which a function of d variables is approximated by an expression such as

$$f(x) = \sum_{i}^{d} \phi_i(x^i) \tag{19}$$

where x^i is the i-th component of the input vector x and the ϕ_i are one-dimensional spline approximations. For linear piecewise splines $\phi_i(x_i) = \sum_j c_{ij}|x^i - b_i^j|$. Obviously such an approximation is not universal: for instance it cannot approximate the function $f(x,y) = xy$. The classical way to deal with the problem is to use tensor product splines. The new alternative that we propose here is *hierarchical additive splines*, which in the case of a 2-layers hierarchy has the form

$$f(x) = \sum_{j}^{K} \phi_j(\sum_{i}^{d} \phi_i(x^i)). \tag{20}$$

and which can be clearly extended to an arbitrary depth. The intuition is that in this way, it is possible to obtain approximation of a function of several variables from functions of one variable because interaction terms such as xy in a polynomial approximation of a function $f(x,y)$ can be obtained from terms such as $e^{log(x)+log(y)}$.

We start with a lemma about the relation between linear rectifiers, which do not correspond to a kernel, and absolute value, which is a kernel.

Lemma 1 *Any given superposition of linear rectifiers* $\sum_i c_i'(x - b'^i)_+$ *with* c_i', b'^i *given, can be represented over a finite interval in terms of the absolute value kernel with appropriate weights. Thus there exist* c_i, b^i *such that* $\sum_i c_i'(x - b'^i)_+ = \sum_i c_i|x - b^i|$.

The proof follows from the facts that a) the superpositions of ramps is a piecewiselinear function, b) piecewise linear functions can be represented in terms of linear splines and c) the kernel corresponding to linear splines in one dimension is the absolute value $K(x,y) = |x-y|$.

Now consider two layers in a network in which we assume degenerate pooling for simplicity of the argument. Because of Lemma 1, and because weights and biases are arbitrary we assume that the the nonlinearity in each edge is the absolute value. Under this assumption, unit j in the first layer, before the non linearity, computes

$$f^j(x) = \sum_{i=1} c_i^j |\left\langle t^i, x \right\rangle - b^i|, \tag{21}$$

where x and w are vectors and the t^i are real numbers. Then the second layer output can be calculated with the nonlinearity $|\cdots|$ instead of $(\cdot)^2$.

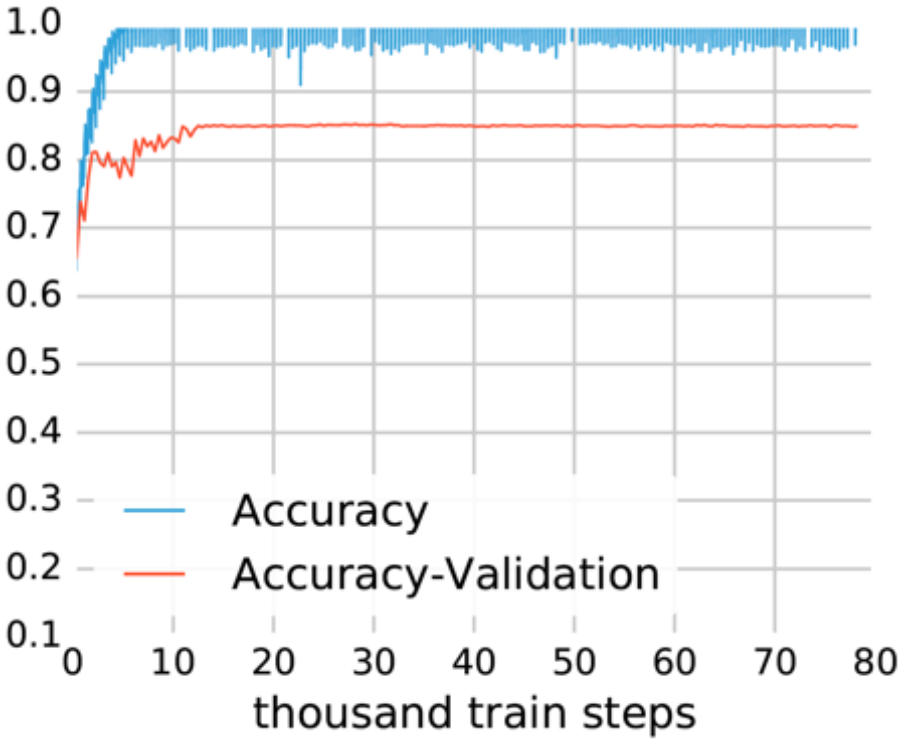

Figure 8 Training original CIFAR. Fix bottom as random, re-train middle and top: training 100%, testing 85%

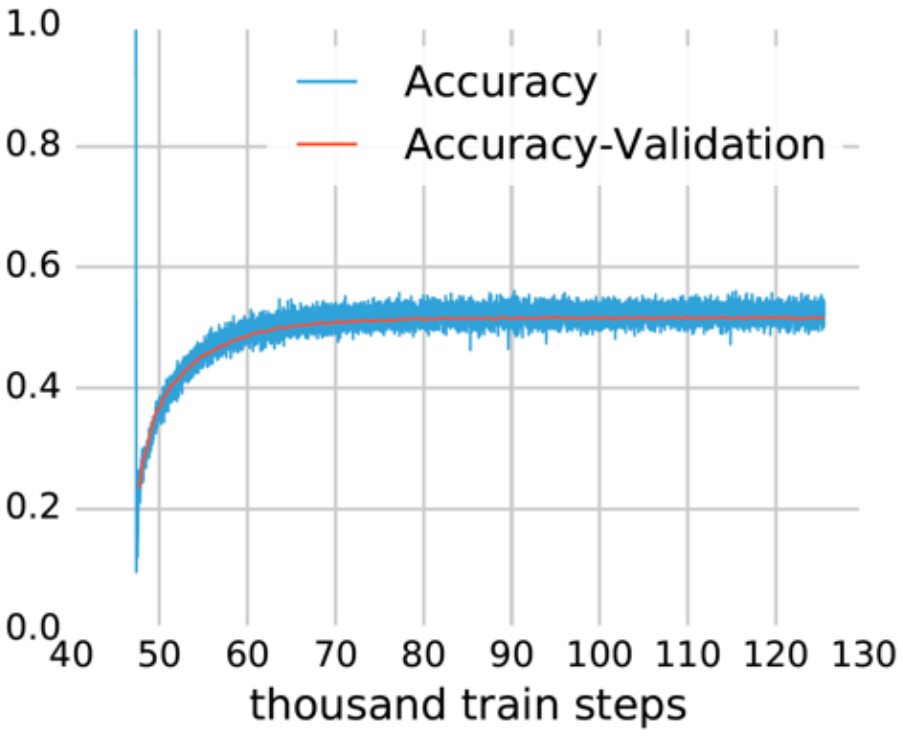

Figure 9 Fix bottom as (the layer from a net trained on random labels), re-train middle and top: training 52.76%, testing 48.15%.

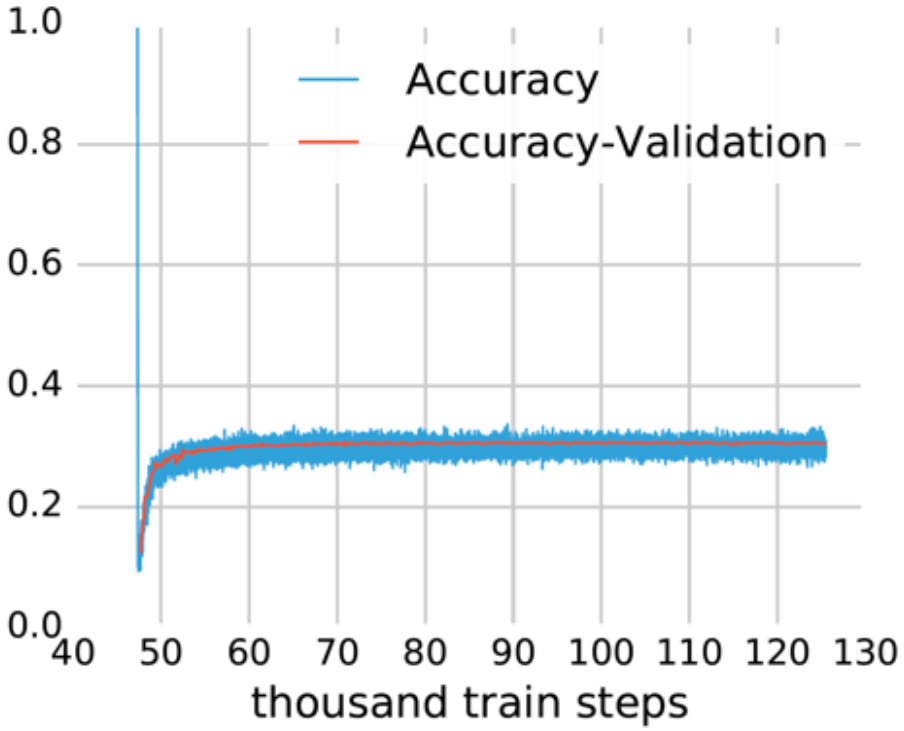

Figure 10 Fix bottom and middle as (the layer from a net trained on random labels), re-train the top: training 30.45%, testing 30.37%.

In the case of a network with two inputs x, y the effective output after pooling at the first layer may be $\phi^{(1)}(x,y) = t_1|x + b_1| + t_2|y + b_2|$, that is the linear combination of two "absolute value" functions. At the second layer terms like $\phi^{(2)}(x,y) = |t_1|x+b_1|+ t_2|y + b_2| + b_3|$ may appear. The output of a second layer still consists of hyperplanes, since the layer is a kernel machine with an output which is always a piecewise linear spline.

Networks implementing tensor product splines are universal in the sense that they approximate any continuous function in an interval, given enough units. Additive splines on linear combinations of the input variables of the form in eq. (21) are also universal (use Theorem 3.1 in [13]). However additive splines on the individual variables are not universal while hierarchical additive splines are:

Theorem *Hierarchical additive splines networks are universal.*

4 On multivariate function approximation

Consider a multivariate function $f : [0,1]^d \to \mathbb{R}$ discretized by tensor basis functions:

$$\phi_{(i_1,...,i_d)}(x_1,...,x_d) := \prod_{\mu=1}^{d} \phi_{i_\mu}(x_\mu), \tag{22}$$

with $\phi_{i_\mu} : [0,1] \to \mathbb{R}, 1 \le i_\mu \le n_\mu, 1 \le \mu \le d$

to provide

$$f(x_1,...,x_d) = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} c(i_1,...,i_d)\phi(i_1,...,i_d)(x_1,...,x_d). \tag{23}$$

The one-dimensional basis functions could be polynomials (as above), indicator functions, polynomials, wavelets, or other sets of basis functions. The total number N of basis functions scales exponentially in d as $N = \prod_{\mu=1}^{d} n_\mu$ for a fixed smoothness class m (it scales as $\frac{d}{m}$).

We can regard neural networks as implementing some form of this general approximation scheme. The problem is that the type of operations available in the networks are limited. In particular, most of the networks do not include the product operation (apart from "sum-product" networks also called "algebraic circuits") which is needed for the straightforward implementation of the tensor product approximation described above. Equivalent implementations can be achieved however. In the next two sections we describe how networks with a univariate ReLU nonlinearity may perform multivariate function approximation with a polynomial basis and with a spline basis respectively. The first result is known and we give it for completeness. The second is simple but new.

4.1 Neural Networks: polynomial viewpoint

One of the choices listed above leads to polynomial basis functions. The standard approach to prove degree of approximations uses polynomials. It can be summarized in three steps:

1. Let us denote with $\mathcal{H}_k$ the linear space of homogeneous polynomials of degree k in $\mathbb{R}^n$ and with $P_k = \bigcup_{s=0}^{k} \mathcal{H}_s$ the linear space of polynomials of degree at most k in n variables. Set $r = \binom{n-1+k}{k} = dim\mathcal{H}_k$ and denote by π_k the space of univariate polynomials of degree at most k. We recall that the number of monomials in a polynomial in d variables with total degree $\leq N$ is $\binom{d+N}{d}$ and can be written as a linear combination of the same number of terms of the form $(\langle w, x\rangle + b)^N$.

   We first prove that

$$P_k(x) = span((\left\langle w^i, x \right\rangle)^s : i = 1, \cdots, r,\ s = 1, \cdots, k \tag{24}$$

   and thus, with, $p_i \in \pi_k$,

$$P_k(x) = \sum_{i=1}^{r} p_i(\langle w_i, x\rangle). \tag{25}$$

   Notice that the effective r, as compared with the theoretical r which is of the order $r \approx k^n$, is closely related to the *separation rank* of a tensor. Also notice that a polynomial of degree k in n variables can be represented exactly by a network with $r = k^n$ units.

2. Second, we prove that each univariate polynomial can be approximated on any finite interval from

$$\mathcal{N}(\sigma) = span\{\sigma(\lambda t - \theta)\}, \lambda, \theta \in \mathbb{R} \tag{26}$$

   in an appropriate norm.

3. The last step is to use classical results about approximation by polynomials of functions in a Sobolev space:

$$E(\mathcal{B}_p^m; P_k; L_p) \leq Ck^{-m} \tag{27}$$

   where $\mathcal{B}_p^m$ is the Sobolev space of functions supported on the unit ball in $\mathbb{R}^n$.

The key step from the point of view of possible implementations by a deep neural network with ReLU units is step number 2. A univariate polynomial can be synthesized – in principle – via the linear combination of ReLUs units as follows. The limit of the linear combination $\frac{\sigma((a+h)x+b)-\sigma(ax+b)}{h}$ contains the monomial x (assuming the derivative of σ is nonzero). In a similar way one shows that the set of shifted and dilated ridge functions has the following property. Consider for $c_i, b_i, \lambda_i \in \mathbb{R}$ the space of univariate functions

$$\mathcal{N}_r(\sigma) = \left\{ \sum_{i=1}^{r} c_i \sigma(\lambda_i x - b_i) \right\}. \tag{28}$$

The following (see Propositions 3.6 and 3.8 in [13]) holds

Proposition 4. *If $\sigma \in \mathcal{C}(\mathbb{R})$ is not a polynomial and $\sigma \in C^\infty$, the closure of $\mathcal{N}$ contains the linear space of algebraic polynomial of degree at most $r-1$.*

Since $r \approx k^n$ and thus $k \approx r^{1/n}$ equation 27 gives

$$E(\mathcal{B}_p^m; P_k; L_p) \leq Cr^{-\frac{m}{n}}. \tag{29}$$

4.2 Neural Networks: splines viewpoint

Another choice of basis functions for discretization consists of splines. In particular, we focus for simplicity on indicator functions on partitions of $[0, 1]$, that is piecewise constant splines. Another attractive choice are Haar basis functions. If we focus on the binary case, section 4.3 tells the full story that does not need to be repeated here. We just add a note on establishing a partition

Suppose that $a = x_1 < x_2 \cdots < x_m = b$ are given points, and set Δx the maximum separation between any two points.

- If $f \in C[a, b]$ then for every $\epsilon > 0$ there is a $\delta > 0$ such that if $\Delta x < \delta$, then $|f(x) - Sf(x)| < \epsilon$ for all $x \in [a, b]$, where Sf is the spline interpolant of f.
- if $f \in \mathcal{C}^2[a, b]$ then for all $x \in [a, b]$

$$|f(x) - Sf(x)| \leq \frac{1}{8}(\Delta x)^2 max_{a \leq z \leq b}|f''(z)|$$

The first part of the Proposition states that piecewise linear interpolation of a continuous function converges to the function when the distance between the data points goes to zero. More specifically, given a tolerance, we can make the error less than the tolerance by choosing Δx sufficiently small. The second part gives an upper bound for the error in case the function is smooth, which in this case means that f and its first two derivatives are continuous.

4.3 Non-smooth ReLUs: how deep nets may work in reality

Our main theorem (2) in this paper is based on polynomial approximation. Because of the n-width result other approaches to approximation cannot yield better rates than polynomial approximation. It is, however, interesting to consider other kinds of approximation that may better capture what deep neural network with the ReLU activation functions implement in practice as a consequence of minimizing the empirical risk.

Our construction shows that a network with non-smooth ReLU activation functions can approximate any continuous function with a rate similar to our other results in this paper. A weakness of this results wrt to the other ones in the paper is that it is valid in the L_2 norm but not in the sup norm. This weakness does not matter in practice since a discretization of real number, say, by using 64 bits floating point representation, will make the class of functions a finite class for which the result is valid also in the L_∞ norm. The logic of the argument is simple:

- Consider the constituent functions of the binary tree, that is functions of two variables such as $g(x_1, x_2)$. Assume that g is Lipschitz with Lipschitz constant L. Then for any ϵ it is possible to set a partition of x_1, x_2 on the unit square that allows piecewise constant approximation of g with accuracy at least ϵ in the sup norm.
- We show then that a multilayer network of ReLU units can compute the required partitions in the L_2 norm and perform piecewise constant approximation of g.

Notice that partitions of two variables x and y can in principle be chosen in advance yielding a finite set of points $0 =: x_0 < x_1 < \cdots < x_k := 1$ and an identical set $0 =: y_0 < y_1 < \cdots < y_k := 1$. In the extreme, there may be as little as one partition – the binary

case. In practice, the partitions can be assumed to be set by the architecture of the network and optimized during learning. The simple way to choose partitions is to choose an interval on a regular grid. The other way is an irregular grid optimized to the local smoothness of the function. As we will mention later this is the difference between fixed-knots splines and free-knots splines.

We describe next a specific construction.

Here is how a linear combination of ReLUs creates a unit that is active if $x_1 \leq x \leq x_2$ and $y_0 \leq y \leq y_1$. Since the ReLU activation t_+ is a basis for piecewise linear splines, an approximation to an indicator function (taking the value 1 or 0, with knots at $x_1, x_1 + \eta$, $x_2\ x_2 + \eta$,) for the interval between x_1 and x_2 can be synthesized using at most 4 units in one layer. A similar set of units creates an approximate indicator function for the second input y. A set of 3 ReLU's can then perform a min operations between the x and the y indicator functions, thus creating an indicator function in two dimensions.

In greater detail, the argument is as follows: For any $\epsilon > 0$, $0 \leq x_0 < x_1 < 1$, it is easy to construct an ReLU network R_{x_0,x_1} with 4 units as described above so that

$$\|\chi_{[x_0,x_1)} - R\|_{L^2[0,1]} \leq \epsilon.$$

We define another ReLU network with two inputs and 3 units by

$$\begin{aligned} \phi(x_1, x_2) &:= (x_1)_+ - (-x_1)_+ - (x_1 - x_2)_+ = \min(x_1, x_2) \\ &= \frac{x_1 + x_2}{2} + \frac{|x_1 - x_2|}{2}. \end{aligned}$$

Then, with $I = [x_0, x_1) \times [y_0, y_1)$, we define a two layered network with 11 units total by

$$\Phi_I(x, y) = \phi(R_{x_0,x_1}(x), R_{y_0,y_1}(y)).$$

Then it is not difficult to deduce that

$$\begin{aligned} \|\chi_I - \Phi_I\|^2_{L^2([0,1]^2)} &= \int_0^1 \int_0^1 \\ &|\min(\chi_{[x_0,x_1)}(x), \chi_{[y_0,y_1)}(y)) - \\ &\min(R_{x_0,x_1}(x), R_{y_0,y_1}(y))|^2 dxdy \leq c\epsilon^2. \end{aligned}$$

Notice that in this case dimensionality is $n = 2$; notice that in general the number of units is proportional to k^n which is of the same order as $\binom{n+k}{k}$ which is the number of parameters in a polynomial in n variables of degree k. The layers we described compute the entries in the 2D table corresponding to the bivariate function g. One node in the graph (there are $n - 1$ nodes in a binary tree with n inputs) contains $O(k^2)$ units; the total number of units in the network is $(n - 1)O(k^2)$. This construction leads to the following results.

Proposition 5. *Compositional functions on the unit cube with an associated binary tree graph structure and constituent functions that are Lipschitz can be approximated by a deep network of ReLU units within accuracy ϵ in the L_2 norm with a number of units in the order of $O((n - 1)L\epsilon^{-2})$, where L is the worse – that is the max – of the Lipschitz constant among the constituent functions.*

Of course, in the case of machine numbers – the integers – we can think of zero as a very small positive number. In this case, the symmetric difference ratio $((x + \epsilon)_+ - (x - \epsilon)_+)/(2\epsilon)$ is the hard threshold sigmoidal function if ϵ is less than this smallest positive number. So, we have the indicator function exactly as long as we stay away from 0. From here, one can construct a deep network as usual.

Notice that the number of partitions in each of two variables that are input to each node in the graph is $k = \frac{L}{\epsilon}$ where L is the Lipschitz constant associated with the function g approximated by the node. Here the role of smoothness is clear: the smaller L is, the smaller is the number of variables in the approximating Boolean function. Notice that if $g \in W_1^2$, that is g has bounded first derivatives, then g is Lipschitz. However, *higher order smoothness* beyond the bound on the first derivative *cannot be exploited by the network* because of the non-smooth activation function[3].

We *conjecture that the construction above that performs piecewise constant approximation is qualitatively similar to what deep networks may represent after training*. Notice that the partitions we used correspond to a uniform grid set a priori depending on global properties of the function such as a Lipschitz bound. In supervised training of deep network the location of each partition is likely to be optimized in a greedy way as a function of the performance on the training set and therefore as a function of inputs and output. Our Theorem 2 and Theorem 5 are obtained under this assumption. In any case, their proofs suggest two different ways of how deep networks could perform function approximations, the first by using derivatives and the second by using piecewise linear splines. In the latter case, optimizing the partition as a function of the input-output examples correspond to *free-knots splines*. The case in which the partitions depend on the inputs but not the target, correspond to classical *fixed-knots splines*. As an aside, one expects that networks with smooth ReLUs will perform better than networks with non-smooth ReLUs in the approximation of very smooth functions.

Still another way to create with ReLUs a discrete table corresponding to the multiplication of two variables, each taking discrete values on a bounded interval is hash the two dimensional table into a one dimensional table. For instance assume that x and y take integer values between $[0, 9]$. Set the variable $z = 10x + y$. This is a one dimensional table equivalent to the 2-dimensional table $x \times y$. This Cantor-like mapping idea works only if restricted to machine numbers, that is to the integers.

5 Vector Quantization and Hierarchical Vector Quantization

Let us start with the observation that a network of radial Gaussian-like units become in the limit of $\sigma \to 0$ a look-up table with entries corresponding to the centers. The network can be described in terms of *soft Vector Quantization* (VQ) (see section 6.3 in Poggio and Girosi, [51]). Notice that hierarchical VQ (dubbed HVQ) can be even more efficient than VQ in terms of storage requirements (see e.g. [52]). This suggests that a hierarchy of HBF layers may be similar (depending on which weights are determined by learning) to HVQ. Note that *compression is achieved when parts can be reused in higher level layers as in convolutional networks*. Notice that the center of one unit at level n of a "convolutional" hierarchy is a combinations of parts provided by each of the lower units feeding in it. This may even happen without convolution and pooling as shown in the following extreme example.

Example Consider the case of kernels that are in the limit delta-like functions (such as Gaussian with very small variance). Suppose that there are four possible quantizations of the input x: x_1, x_2, x_3, x_4. One hidden layer would consist of four units $\delta(x - x_i), i = 1, \cdots, 4$. But suppose that the vectors x_1, x_2, x_3, x_4 can be decomposed in terms of two smaller parts or features x' and x", e.g. $x_1 = x' \oplus x$", $x_2 = x' \oplus x'$, $x_3 = x$" $\oplus\, x$"

[3] In the case of univariate approximation on the interval $[-1, 1]$, piecewise linear functions with inter-knot spacing h gives an accuracy of $(h^2/2)M$, where M is the max absolute value of f''. So, a higher derivative does lead to better approximation : we need $\sqrt{2M/\epsilon}$ units to give an approximation of ϵ. This is a saturation though. Even higher smoothness does not help.

and $x_4 = x" \oplus x'$. Then a two layer network could have two types of units in the first layer $\delta(x - x')$ and $\delta(x - x")$; in the second layer four units will detect the conjunctions of x' and $x"$ corresponding to x_1, x_2, x_3, x_4. The memory requirements will go from $4N$ to $2N/2 + 8$ where N is the length of the quantized vectors; the latter is much smaller for large N. Memory compression for HVQ vs VQ – that is for multilayer networks vs one-layer networks – increases with the number of (reusable) parts. Thus for problems that are *compositional*, such as text and images, hierarchical architectures of HBF modules minimize memory requirements.

Classical theorems (see refrences in [53, 54] show that one hidden layer networks can approximate arbitrarily well rather general classes of functions. A possible advantage of multilayer vs one-layer networks that emerges from the analysis of this paper is memory efficiency which can be critical for large data sets and is related to generalization rates.

6 Approximating compositional functions with shallow and deep networks: numerical experiments

Figures 3, 4 , 14, 5, 11, 12, 13 15, and 16 show some of our numerical experiments.

7 Compositional functions and scalable algorithms

We formalize the requirements on the algorithms of local compositionality is to define *scalable computations* as a subclass of nonlinear discrete operators, mapping vectors from $\mathbb{R}^n$ into $\mathbb{R}^d$ (for simplicity we put in the following $d = 1$). Informally we call an algorithm $K_n : \mathbb{R}^n \mapsto \mathbb{R}$ *scalable* if it maintains the same "form" when the input vectors increase in dimensionality; that is, the same kind of computation takes place when the size of the input vector changes. This motivates the following construction. Consider a "layer" operator $H_{2m} : \mathbb{R}^{2m} \mapsto \mathbb{R}^{2m-2}$ for $m \geq 1$ with a special structure that we call "shift invariance".

Definition 2. *For integer $m \geq 2$, an operator H_{2m} is shift-invariant if $H_{2m} = H'_m \oplus H''_m$ where $\mathbb{R}^{2m} = \mathbb{R}^m \oplus \mathbb{R}^m$, $H' = H''$ and $H' : \mathbb{R}^m \mapsto \mathbb{R}^{m-1}$. An operator $K_{2M} : \mathbb{R}^{2M} \to \mathbb{R}$ is called scalable and shift invariant if $K_{2M} = H_2 \circ \cdots H_{2M}$ where each H_{2k}, $1 \leq k \leq M$, is shift invariant.*

We observe that *scalable shift-invariant operators $K : \mathbb{R}^{2m} \mapsto \mathbb{R}$ have the structure $K = H_2 \circ H_4 \circ H_6 \cdots \circ H_{2m}$, with $H_4 = H'_2 \oplus H'_2$, $H_6 = H''_2 \oplus H''_2 \oplus H''_2$, etc..*

Thus the structure of a *shift-invariant, scalable operator* consists of several layers; each layer consists of identical blocks; each block is an operator $H : \mathbb{R}^2 \mapsto R$: see Figure 19 . We note also that an alternative weaker constraint on H_{2m} in Definition 2, instead of shift invariance, is mirror symmetry, that is $H'' = R \circ H'$, where R is a reflection. Obviously, shift-invariant scalable operator are equivalent to shift-invariant compositional functions. Obviously the definition can be changed in several of its details. For instance for two-dimensional images the blocks could be operators $H : \mathbb{R}^5 \to \mathbb{R}$ mapping a neighborhood around each pixel into a real number.

The final step in the argument uses the results of previous sections to claim that a nonlinear node with two inputs and enough units can approximate arbitrarily well each of the H_2 blocks. This leads to conclude that deep convolutional neural networks are natural approximators of *scalable, shift-invariant operators*.

8 Old-fashioned generalization bounds

Our estimate of the number of units and parameters needed for a deep network to approximate compositional functions with an error ϵ_G allow the use of one of several available bounds for the generalization error of the network to derive sample complexity bounds. It is important to notice however that these bounds *do not* apply to the networks used today, since they assume a number of parameters smaller than the size of the training set. We report them for the interest of the curious reader. Consider theorem 16.2 in [55] which provides the following sample bound for a generalization error ϵ_G with probability at least $1 - \delta$ in a network in which the W parameters (weights and biases) which are supposed to minimize the empirical error (the theorem is stated in the standard ERM setup) are expressed in terms of k bits:

$$M(\epsilon_G, \delta) \leq \frac{2}{\epsilon_G^2}(kW \log 2 + \log(\frac{2}{\delta})) \tag{30}$$

This suggests the following comparison between shallow and deep compositional (here binary tree-like networks). Assume a network size that ensure the same approximation error ϵ .

Then in order to achieve the same generalization error ϵ_G, the sample size $M_{shallow}$ of the shallow network must be much larger than the sample size M_{deep} of the deep network:

$$\frac{M_{deep}}{M_{shallow}} \approx \epsilon^n. \tag{31}$$

This implies that for largish n there is a (large) range of training set sizes between M_{deep} and $M_{shallow}$ for which deep networks will not overfit (corresponding to small ϵ_G) but shallow networks will (for dimensionality $n \approx 10^4$ and $\epsilon \approx 0.1$ Equation 31 yields $m_{shallow} \approx 10^{10^4} m_{deep}$).

A similar comparison is derived if one considers the best possible expected error obtained by a deep and a shallow network. Such an error is obtained finding the architecture with the best trade-off between the approximation and the estimation error. The latter is essentially of the same order as the generalization bound implied by inequality (30), and is essentially the same for deep and shallow networks, that is

$$\frac{rn}{\sqrt{M}}, \tag{32}$$

where we denoted by M the number of samples. For shallow networks, the number of parameters corresponds to r units of n dimensional vectors (plus off-sets), whereas for deep compositional networks the number of parameters corresponds to r units of 2 dimensional vectors (plus off-sets) in each of the $n - 1$ units. Using our previous results on degree of approximation, the number of units giving the best approximation/estimation trade-off is

$$r_{shallow} \approx \left(\frac{\sqrt{M}}{n}\right)^{\frac{n}{m+n}} \qquad \text{and} \qquad r_{deep} \approx \left(\sqrt{M}\right)^{\frac{2}{m+2}} \tag{33}$$

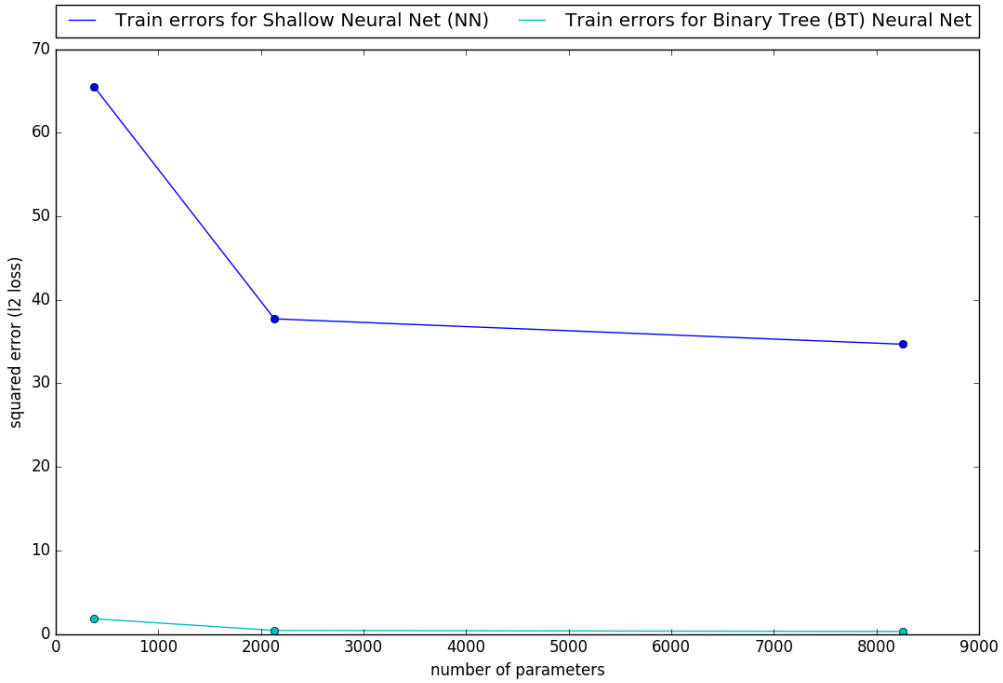

Figure 11 An empirical comparison of shallow vs 3-layers binary tree networks in the learning of compositional functions. The loss function is the the standard mean square error (MSE). There are several units per node of the tree. In our setup here the network with an associated binary tree graph was set up so that each layer had the same number of units. The number of units for the shallow and binary tree neural network were chosen such that both architectures had the approximately same number of parameters. The compositional function is $f(x_1, \cdots, x_8) = h_3(h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8)))$ and is approximated by a network with ReLU activations. A description of the compositional function is as follows: the first layer units $h_{11}, h_{12}, h_{13}, h_{14}$ are equal to $h_1(x, y) = 0.59cos(1.5\pi(x + y))$, the second layer units h_{21}, h_{22} are equal to $h_2(x, y) = 1.1(x + y)^2 - 1$ and the final layer unit h_3 is also $h_3(x, y) = 1.1(x + y)^2 - 1$. The training and test sets both had 60K training examples. The variant of SGD that was used was the Adam [32] optimizer for both experiments. In order to get the best solution possible we ran 200 independent hyper parameter searches using random search [33] and then reported the one with the lowest training error for both tasks. The hyper parameters included the step size, the decay rate, frequency of decay and the mini-batch size. The exponential decay hyper parameters for Adam were kept fixed to the recommended values according to the original paper [32]. The implementations were based on TensorFlow [34].

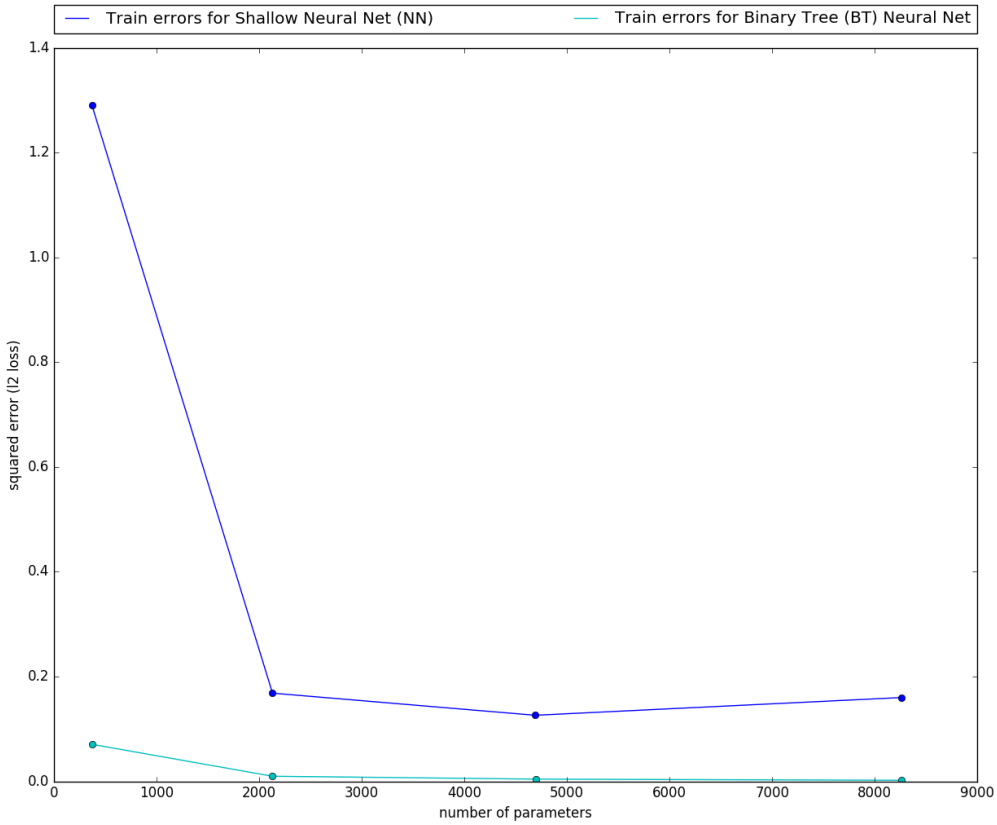

Figure 12 An empirical comparison of shallow vs 3-layers binary tree networks in the learning of compositional functions. The loss function is the the standard mean square error (MSE). There are several units per node of the tree. In our setup here the network with an associated binary tree graph was set up so that each layer had the same number of units. The number of units for the shallow and binary tree neural network were chosen such that both architectures had the approximately the same number of parameters. The compositional function is $f(x_1, \cdots, x_8) = h_3(h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8)))$ and is approximated by a network with ReLU activations. A description of the compositional function is as follows: the first layer units $h_{11}, h_{12}, h_{13}, h_{14}$ are equal to $h_1(x, y) = 0.7(x+2y)^2$, the second layer units h_{21}, h_{22} are equal to $h_2(x, y) = 0.6(1.1x + 1.9y)^3$ and the final layer unit h_3 is also $h_3(x, y) = 1.3\sqrt{1.2x + 1.3}$. The experiment setup (training and evaluation) was the same as in 11.

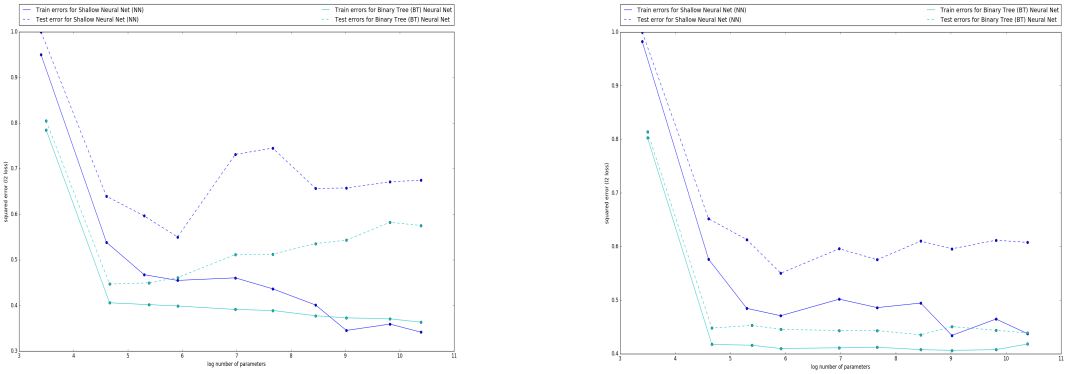

Figure 13 An empirical comparison of shallow vs 3-layers binary tree networks in the learning of compositional functions with Gaussian noise. The loss function is the the standard mean square error (MSE). There are several units per node of the tree. In our setup here the network with an associated binary tree graph was set up so that each layer had the same number of units. The number of units for the shallow and binary tree neural network were chosen such that both architectures had the same number of parameters. The noisy compositional function is $f(x_1, \cdots, x_8) = h_3(h_{21}(h_{11}(x_1, x_2), h_{12}(x_3, x_4)), h_{22}(h_{13}(x_5, x_6), h_{14}(x_7, x_8))) + \epsilon$ and is learned by a network with ReLU activations. The functions h_1, h_2, h_3 are as described in Figure 12. The training and test sets both had 2K training examples. The variant of SGD that was used was the Adam [32] optimizer for both experiments. In order to get the best solution possible we ran 200 independent hyper parameter searches using random search [33]. For the left figure we choose the hyper parameter selection that resulted in the lowest training error. For the right figure choose ch the hyper parameter selection that resulted in the lowest validation error. For both we reported the corresponding training and test error. The hyper parameters included the step size, the decay rate, frequency of decay and the mini-batch size. The exponential decay hyper parameters for Adam were the recommended values according to the original paper [32]. Its interesting to note that when the training error was used as a selection criterion, overfitting seemed to happen for both the shallow and binary tree network. However, when the validation set is used as the selection criterion overfitting is not observed anymore for either model. In addition, the test error for the binary tree neural network noticeably decreases while it does not for the shallow network.

for shallow and deep networks, respectively. The corresponding (excess) expected errors $\mathcal{E}$ are

$$\mathcal{E}_{shallow} \approx \left(\frac{n}{\sqrt{M}}\right)^{\frac{m}{m+n}} \tag{34}$$

for shallow networks and

$$\mathcal{E}_{deep} \approx \left(\frac{1}{\sqrt{M}}\right)^{\frac{m}{m+2}} \tag{35}$$

for deep networks. For the expected error, as for the generalization error, deep networks appear to achieve an exponential gain. The above observations hold under the assumption that the optimization process during training finds the optimum parameters values for both deep and shallow networks. Taking into account optimization, e.g. by stochastic gradient descent, requires considering a further error term, but we expect that the overall conclusions about generalization properties for deep vs. shallow networks should still hold true.

9 Part II and III of the Theory

As we mentioned in the introduction, a theory of Deep Learning should consist of at least three main parts: Approximation, Optimization of the Empirical Risk and Generalization.

This paper is mainly concerned with the first part – Approximation Theory. We summarize briefly the two other parts.

Theory II: Landscape of the Empirical Risk

We assume a deep networks of the binary tree type with weight sharing. We also assume overparametrization, that is more parameters than data points, since this is how successful deep networks have been used.

In general, under these conditions, we expect that zeros of the empirical error yield a set of quasi-polynomial equations (at the zeros) that have an infinite number of solutions (for the network weights) and are sparse because of ReLUs (the arguments of some of the ReLUs are negative and thus the value of the corresponding ReLUs is zero). The equations have the form $c_1 ABCA + c_2 ABCdc_3 A'BC'b + ... = y$ with coefficients representing components of the data vectors (one vector per data point); the unknowns are $A, B, C, B', \cdots$. The system of equations is linear in the monomials with nonlinear constraints e.g. $ABCD = z_1$, $ABCd = z_2$ on the unknown weights $A, B, C \cdots$. It is underdetermined (more unknowns than equations, e.g. data points) because of the assumed overparametrization. We use Bezout theorem to estimate an upper bound in the number of real zeros.

Theory III: Generalization by SGD

Part III deals with the puzzle that solutions found by "repeat SGD" in underdetermined situations (fewer data points e.g. equations than unknown parameters) empirically generalize well on new data. Our key result is to show that repeat SGD is CV_{loo} stable on a empirical risk which has many degenerate global minima. It therefore generalizes. In other words we have the following claim

Claim: Suppose the following properties of $\mathcal{H}, \rho, Z$ are satisfied:

1. *$I_{S_n}[f]$ has M (convex, possibly degenerate) global minima*

Then repeat SGD on S_n generalizes.

Noteworthy features of generalization by "repeat SGD" is that the intrinsic noise associated with SGD wrt GD is the key to its generalization properties. The intuition is that because of the noise term, SGD finds with high probability *structurally stable* zeros (that is robust wrt to perturbations of the weights) and these coincide with CV_{loo} stable solutions.

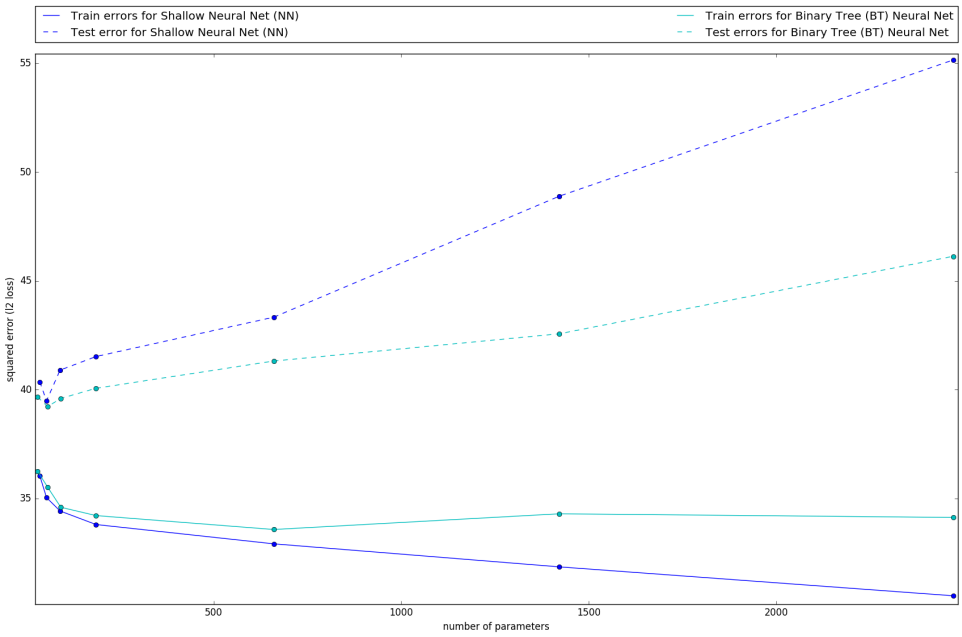

Figure 14 An empirical comparison of shallow vs 2-layers binary tree networks in the learning of compositional functions with Gaussian noise. The loss function is the the standard mean square error (MSE). There are several units per node of the tree. The network with an associated binary tree graph was set up so that each layer had the same number of units. By choosing appropriately the number of units for the shallow and binary tree neural networks, both architectures had the same number of parameters. The noisy compositional function is $f(x_1, x_2, x_3, x_4) = h_2(h_{11}(x_1, x_2), h_{12}(x_3, x_4)) + \epsilon$ and is learned by a network with ReLU activations. The functions h_{11},h_{12}, h_2 are as described in Figure 3. The training and test sets both had 2K training examples. The variant of SGD that was used was the Adam [32] optimizer for both experiments. In order to get the best solution possible we ran 200 independent hyper parameter searches using random search [33] and then reported the one with the lowest training error for both tasks. The hyper parameters included the step size, the decay rate, frequency of decay and the mini-batch size. The exponential decay hyper parameters for Adam were the recommended values according to the original paper [32]. The implementations were based on TensorFlow [34].

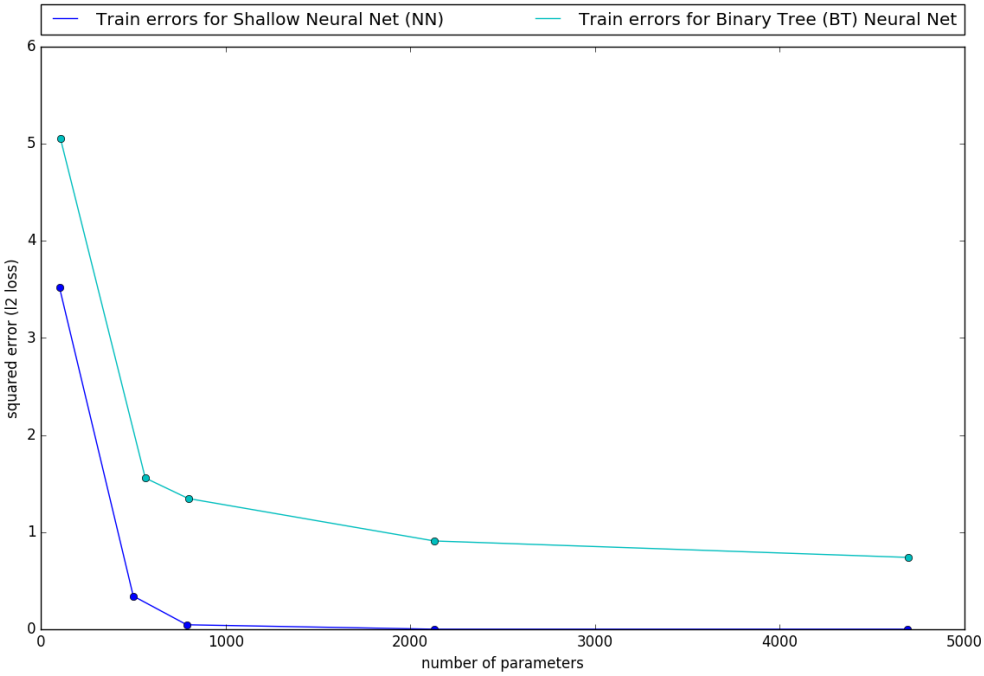

Figure 15 Here we empirically show that depth of the network and compositionality of the function are important. We compare shallow vs 3-layers binary tree networks in the task of approximating a function that is not compositional. In particular the function is a linear combination of 100 ReLU units $f(x) = \sum_{i=1}^{100} c_i(\langle w_i, x\rangle + b_i)_+$. The loss function is the the standard mean square error (MSE). The training set up was the same as in figure 12. This experiment shows that when compositionality is not present in the function to be approximated then having depth in the network can actually worsen performance of the network (confront with in 12).

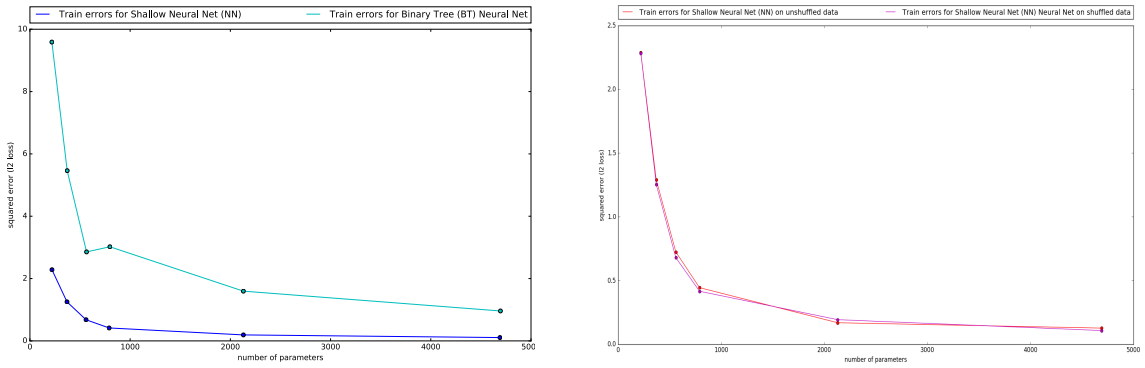

Figure 16 The figure suggests again that that depth of the network and compositionality of the function are important. The figure compares shallow vs a 3-layers binary tree networks in the task of approximating a function that lacks the structure of local hierarchy and compositionality. In particular if the compositionality of the function presented in figure 12 is missing, 3-layered binary tree networks show a decreased approximation accuracy (see left figure). However, the performance of shallow networks remain completely unchanged as shown on the right figure. Compositionality of the function in 12 was broken by shuffling the input coordinates $x \in \mathbb{R}^8$ randomly but uniformly across all the training examples. More precisely a fixed shuffling function $shuffle(x)$ was selected and the deep and shallow networks were trained with the data set $X = \{(shuffle(x_i), f(x_i)\}_{i=1}^{60,000}$. The figure on the left shows that the binary tree network was not able to do better than the shallow network once compositionality was missing from the data. The figure on the right shows that this did not change the performance of the shallow network.

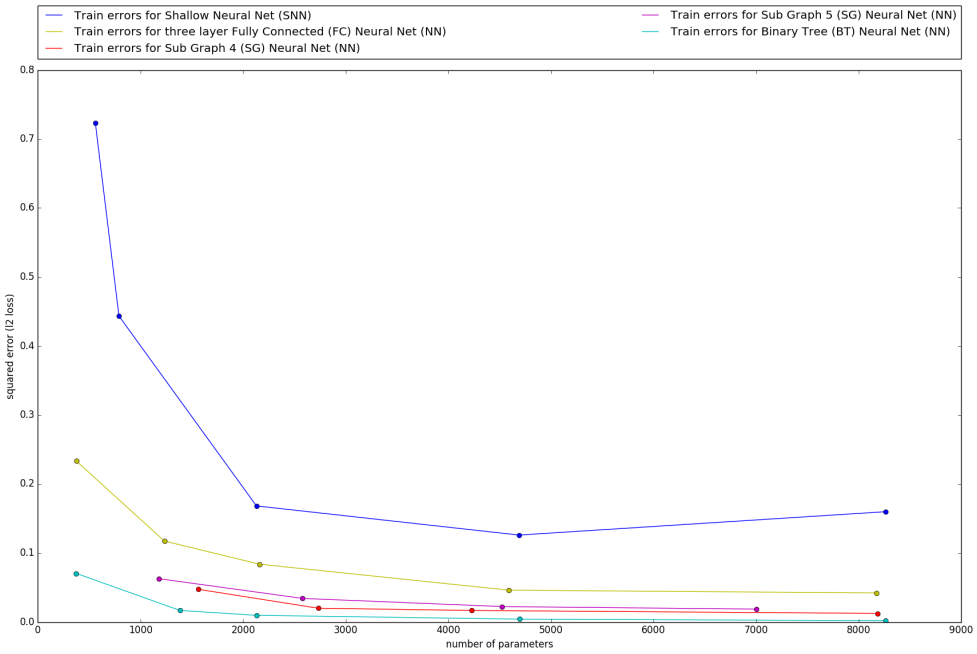

Figure 17 This figure shows a situation in which the structure of the neural network does not match exactly the structure of the function to be learned. In particular, we compare the approximation accuracy of networks that contain the original function as a function of increasing in similarity between the network graph and the function graph. The function being approximated is the same as in 12.d. The network subgraphs 4 and 5 have the same convolution structure on the first layer with filter size 4. For the second layer subgraph 5 has more connections than subgraph 4, which reflects the higher error of subgraph 5. More precisely, subgraph 4 has a filter of size of two units with stride of two units while subgraph 5 has filter size of three units and stride of size one. This figure shows that even when the structure of the networks does not match exactly the function graph, the accuracy is still good.

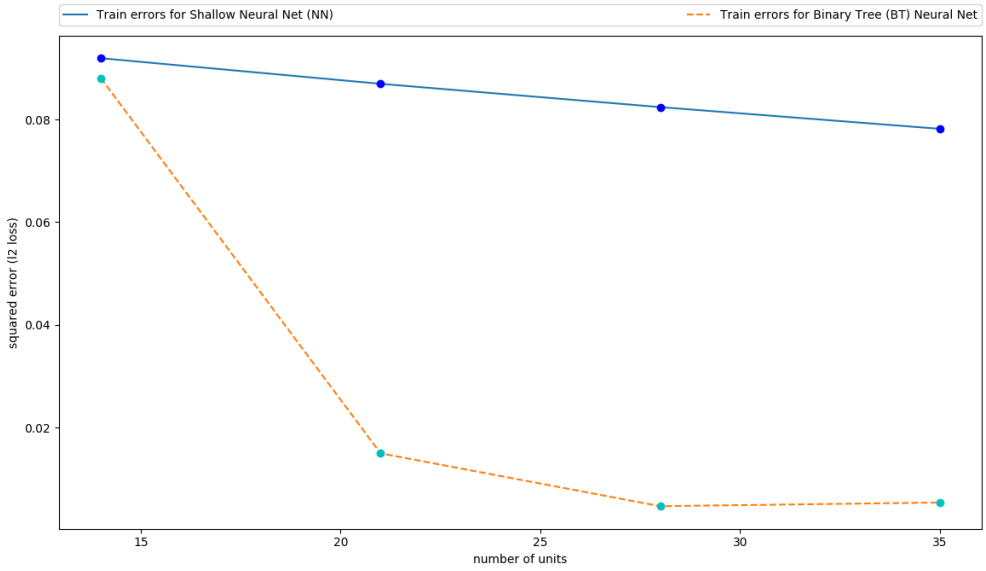

Figure 18 An empirical comparison of shallow vs 3-layers binary tree networks in the learning of compositional functions. The loss function is the the standard mean square error (MSE). There are several units per node of the tree. In our setup here the network with an associated binary tree graph was set up so that each layer had the same number of units. The number of units for the shallow and binary tree neural network are the same. The compositional function is the product function $f(x_1, ..., x_8) = \prod_{i=1}^{8} x_i$ (also known the parity function when the input is restricted to $\{-1, +1\}$) and is approximated by a network with ReLU activations. The variant of SGD that was used was the Adam [32] optimizer for both experiments. In order to get the best solution possible we ran 200 independent hyper parameter searches using random search [33] and then reported the one with the lowest training error for both tasks. The hyper parameters included the step size, the decay rate, frequency of decay and the mini-batch size. The exponential decay hyper parameters for Adam were kept fixed to the recommended values according to the original paper [32]. The implementations were based on TensorFlow [34].

Figure 19 A shift-invariant, scalable operator. Processing is from the bottom (input) to the top (output). Each layer consists of identical blocks, each block has two inputs and one output; each block is an operator $H_2 : \mathbb{R}^2 \mapsto R$. The step of combining two inputs to a block into one output corresponds to coarse-graining of a Ising model.