CENTER FOR
Brains
Minds+
Machines

CBMM Memo No. 138 **March 8, 2023**

How Deep Sparse Networks Avoid the Curse of Dimensionality: Efficiently Computable Functions are Compositionally Sparse

A Perspective on the Foundations of Deep Learning[1]

Tomaso Poggio
Center for Brains, Minds, and Machines, MIT, Cambridge, MA, USA

Abstract

The main claim of this perspective is that compositional sparsity of the target function, which corresponds to the task to be learned, is the key principle underlying machine learning. Mhaskar and Poggio (2016) proved that sparsity of the compositional target functions, naturally leads to sparse deep networks for approximation and thus for optimization. This is the case of most CNNs in current use, in which the known compositional structure, described in terms of the function graph of the target function is reflected in the sparse connectivity of the network. When the compositional sparsity of the target function is unknow, I conjecture that transformers are able to implement a flexible version of compositional sparsity (selecting which input tokens interact in the MLP layer), through the self-attention layers.

Surprisingly, the assumption of compositional sparsity of the target function is not restrictive in practice, since I prove here that for computable functions *compositional sparsity is equivalent to efficient computability, that is Turing computability in polynomial time*.

This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216.

[1]The first version, published on October 11, 2022 is available under request. Several appendices have been deleted in this version which also differs because of several edits in the main text and theorems.

1 Introduction

We still do not understand why deep networks work. Until recently this question could have been be rephrased as a question about why CNNs work so well, since they, unlike dense networks, achieved significantly superior performance in certain tasks, compared to classical techniques such as kernel machines. In the meantime, however, other architectures, especially transformers, also show amazing performance. Is there a common principle at the core of these successful neural network architectures? In the following, I describe a framework built around a specific fundamental principle that, I conjecture, must underlie most of deep learning.

This note is thus about foundations of ML. It shows how the curse of dimensionality in the approximations of functions can be avoided for the broad class of *e ciently computable functions*, thus establishing *a bridge between computability and approximation*. The main result is that for smooth functions on the reals – functions with Lipschitz continuity – compositional sparsity is equivalent to e ciently computability, that is, they can be computed in polynomial time by a Turing Machine (the term e cient is used here to mean non-exponential). The result implies that all "realistic" smooth functions are compositionally sparse. The situation for Boolean functions is even simpler. It also implies that deep sparse RELU networks are natural parametric and constructive approximators for these functions. In a sense, compositionally sparse functions are in practice "all" functions and sparse RELU networks are the associated class of their universal approximators.

2 Machine learning and approximation theory

In the theory of ML, the first conceptual step is to define parametric approximators of the class of functions to be learned. Examples of approximators are generalized additive models, polynomials and deep RELU network. In this first step of the theory, we want approximators to a class of functions, ideally as large as possible. Furthermore, we want the parametric approximators to be e ciently computable, to ensure that optimization on the training data is possible. In particular, this means that the number of parameters in the approximators cannot be exponential in the *e ective* dimensionality $\overline{d} = \frac{d}{s}$, where s is a measure of smoothness such as the number of bounded derivatives: in other words, the approximators must avoid the curse of dimensionality. Recall that approximation of a generic continuous function on the reals using multivariate polynomials of degree k has exponential complexity $O(k^{\frac{d}{s}})$ in the number of parameters. Notice that evaluation at points of f and even continuity of f are not enough to ensure a meaningful approximation, defined as a convergent sequence of approximating f_n that converges to f .

3 Compositionally sparse functions can be approximated by deep networks without curse of dimensionality

Let me define a class of sparse functions, that is *sparse compositional functions* that are the composition of sparse constituent functions. This class is interesting for approximation theory: in fact the assumption of sparse target functions has appeared often in the recent approximation literature (see [1, 2, 3, 4, 5]). It is important to notice that *all functions have a compositional representation which is not unique*, since, in general, a function admits more than one compositional graph representation[1].

Definition 1. *A sparse compositional function on the reals of d variables with smoothness s, is a function that can be represented as the composition of no more than $poly(\frac{d}{s})$ sparse functions, that is functions each depending on a small number $\leq d_0$ of variables, with $d_0 << \frac{d}{s}$.*

A similar definition applies to Boolean functions by replacing the e ective dimensionality $\overline{d}$ with d.

An interesting, specific pair (function class, approximator) is the class of compositional smooth functions and of deep RELU networks. In fact, the following theorem by Mhaskar and Poggio [6]

[1] In the following I often refer to "sparse function" meaning "a sparse representation of the function"

shows that functions with bounded first derivatives, that can be represented by a *compositionally sparse* function graph, can be approximated arbitrarily well by deep, sparse RELU networks with $poly(d)$ parameters. The motivation for the result was the classical curse of dimensionality: an upper bound on the number of parameters needed for approximation of a continuous function supported on a compact domain of $\mathcal{R}^d$ is $W = \mathcal{O}(\epsilon^{-\frac{d}{s}})$, where ϵ is the approximation error and s – the number of bounded derivatives – is a measure of smoothness of the function with $s \geq 1$. The quantity $\frac{d}{s}$ can be called *e ective dimensionality*. The curse can be avoided by shallow or deep networks if s is large and in particular if s grows with d (as in The Barron class of functions). The curse can also be avoided by deep networks, but not by shallow ones, if the representation of the smooth function is *compositionally sparse*, that is if the function graph is such that each constituent function has small e ective dimensionality[2].

Theorem 1. *[6] 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 constituent functions is in the Sobolev space $W_{m_v}^{d_v}$. Consider shallow and deep networks with infinitely smooth activation function. 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*

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

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

$$N_d = \mathcal{O}\left(\sum_{\eta \in V} \epsilon^{-d_v/m_v}\right).$$

Remark A slightly di erent definition can be given in which it is assumed that there exists a parametrized sequence of constructive computable approximations (e.g. by RELU networks) f_n to f, each depending on n parameters, with desired errors $\leq \epsilon_n$ in the sup norm. As emphasized by H. Mhaskar, approximation theorems, such as Theorem 1 are too crude a tool to use - one has to add that the approximation must be constructive, based on values of the target function. Otherwise, as shown in [7], where for the first time both dimension independent as well as constructive bounds for the same class of functions are proven), ReLU networks can achieve dimension independent bounds, obviating the need to have deep networks from the point of view of degree of approximation alone.

4 E ciently computable functions are compositionally sparse

Sparse compositional functions with bounded first derivatives can be approximated by a deep network with the same graph without curse of dimensionality. But how broad is the class of sparse compositional functions? In this section I show that it is quite broad since it is equivalent to the class of e ciently computable functions.

I will first provide a specific version of the definition of a *computable* function. For Boolean functions, computability is equivalent to computability by a Turing machine[3]. For functions on the reals there are various notions of computability. The simplest is Borel-Turing computability. As shown very recently, they all have some technical problems (see [8, 9]), in the sense that there exist functions on the reals (such as the pseudoinverse) that are not are not computable. Here we bypass this issues and consider the standard case, that is functions that are Borel-Turing computable. Our focus in this note is whether such computable functions, are, or are not, computable in polynomial time.

[2] I use the term *compositional sparsity* following [1] instead of another equivalent term we used earlier: *hierarchical compositionality*.

[3] The Church-Turing thesis states that any real-world computation can be translated into an equivalent computation involving a Turing machine, which is equivalent to using general recursive functions.

Definition 2. *A function $f : I \to \mathbb{R}_c^k, I \subset \mathbb{R}_c^d$, where $\mathbb{R}_c$ is the set of computable real numbers, is called Borel-Turing computable, if there exists an algorithm (or Turing machine) that transforms each given computable representation of a computable vector $x \in I$ into a representation for $f(x)$. The special case of e cient computability requires computability in polynomial time/space in d.*

The following observation, cast as a theorem here is natural.

Theorem 2. *E ciently computable functions on the reals with bounded first derivatives are compositionally sparse. E ciently computable Boolean functions are compositionally sparse.*

Proof sketch Assume that a function is e□ciently computable by a Turing machine. This means that the function can be represented by the composition of at most a polynomial number of functions, each corresponding to the basic read-write step in a Turing machine, which is itself a sparse function. In fact, a Turing machine can be written as a compositional function $y = f^{(t}(x, s)$ where $f : Z^n \times S^m \mapsto Z^h \times s^k$, S being parameters characterizing the state of the machine (this observation is due to S. Vempala). If t is bounded we have a finite state machine, otherwise a Turing machine. Also notice that a Turing machine computes recursive functions which are compositions of sparse functions.

5 E□cient computability, compositional sparsity and deep, sparse RELU networks

Here are two obvious but interesting corollaries directly implied by the theorems above.

5.1 E□cient computability is equivalent to compositional sparsity

Consider smooth real-valued functions in d variables, that is functions with Lipschitz continuity, that are compositionally sparse. Theorem 1 shows that such functions are computed by deep RELU networks (assuming computability of the latter) with a number of parameters which is $poly(d)$. The same is true for Boolean functions. RELU networks can be simulated e□ciently by a Turing machine, since each layer in a deep network corresponds to a finite number of steps of a Turing machine. For the other direction, Theorem 3 shows that e□ciently computable functions are compositionally sparse.

Corollary 1. *For computable functions (if the function is on the reals, bounded first derivatives are required), compositional sparsity is equivalent to $poly(d)$ computability.*

Theorem 3. *Functions on $I \subset \mathbb{R}^d$ with Lipschitz continuity which are e ciently computable are compositionally sparse. E ciently computable Boolean functions are compositionally sparse.*

Proof a) Assume that a function is computable by a Turing machine. This implies that the function on the reals can be approximated by a Boolean function (obtained by discretizing the function which is Lipshitz continuous, see [10]). b) Since the function is e□ciently computable, it must have a sparse Fourier representation which is the sum of $poly(d)$ sparse terms. c) Theorem 4.1 in [11] shows that f is an s-sparse polynomial in d variables, with individual degrees of its variables bounded by k, then the sparsity of each factor of f is bounded by $s^{\mathcal{O}(k^2 \log d)}$.This also corresponds to a conjunction of disjunctions – in other words to a product of sums, which is itself equivalent to the composition of sparse Boolean functions. In the other direction, if a function is sparse compositional then a sparse deep network can approximate it because of theorem 1. Then the network is e□ciently Turing computable.

Remarks

- There are other ways to prove the result. A Turing computable function can be represented by the composition of at most a polynomial number of functions, each corresponding to the basic read-write step in a Turing machine, which is itself a sparse function. In fact, a Turing machine can be written as a compositional function $y = f^t(x, s)$ where $f : Z^n \times S^m \mapsto Z^h \times S^k$, S being parameters characterizing the state of the machine and Z Boolean variables (this observation is due to S. Vempala). If t is bounded we have a finite state machine, otherwise a Turing machine. Also notice that a Turing machine computes recursive functions which are compositions of sparse functions.
- The restriction to computable functions in the theorem is to avoid the problem that several functions on the reals are not computable according to standard definitions [9] .
- An alternative approach that avoids the issue of computability of functions on the reals is to work with Boolean functions throughout, replacing the MP theorem with its Boolean version, as sketched in section 9.4.1.
- Without the assumption of smooth target functions, which is equivalent to smooth constituent functions for a compositional function, the equivalence between compositional sparsity and computability does not hold. The Appendices discuss the situation.

5.2 E□ciently computable functions can be approximated by a deep network with appropriate sparse architecture without curse of dimensionality

Corollary 2. *All e ciently computable functions (if the function is on the reals, Lipschitz continuity is required), can be approximated by a deep network with the appropriate sparse architecture matching the graph of a sparse representation of the function.*

Remark One of the main implications of theorem 2 is that in practice all functions may be approximated by an *appropriately sparse* neural network without curse of dimensionality. This, in turn, provides theoretical foundations for

1. using deep sparse networks in general learning tasks where the parametric approximation is optimized by training on a training set since our results show there for any realistic function there exist a sparse representation that can be approximated well by a deep sparse network;
2. using sparse tensor representations such as the Hierarchical Tucker format [12, 13] in representing rather generic functions.

Notice that this implies that one of the main challenges in learning is hypothesizing or finding a sparse compositional graph representing the class of functions to be learned (see later section on CNNs and transformers).

6 Learning theory: compositional sparsity leads to orders-of-magnitude better generalization

The theorems above imply that deep, sparse RELU networks can be used for training, that is for optimization of the function class wrt given data and a chosen loss function. The optimized network may or may not generalize well. The next question provides some light on this issue, independently of whether the optimization is in the underparametrized or overparametrized case. The latter is more relevant for current usage.

It is possible to prove that sparsity of a network approximating a (sparse) target function reduces its complexity by orders of magnitude. In particular, the following result holds [14]

Theorem 4. *(informal) The Rademacher complexity of a deep overparametrized network is much smaller for convolutional layers with a local kernel than for a dense layers: if the kernel has dimensionality k and the dimensionality of the layer is n, then the contribution of the layer to the Rademacher complexity of the network is $\sqrt{\frac{k}{n}}\|W\|$ instead of $\|W\|$, where $\|W\|$ is the Frobenius norm of the layer weight matrix W.*

In complete analogy with the approximation result the key property here is locality of the convolution kernel ($k << n$) and *not weight sharing*. Notice that an equivalent result for underparametrized networks follows directly from considerations of VC dimension (see Appendix, section 8 in [15]). The novelty here is to show that sparsity can lead to generalization in the overparametrized case, when weight decay, that is regularization, is present[4].

7 Optimization and open questions

7.1 The sparse graph is known: CNNs

In the underparametrized case, recent work (see for instance [2]) has shown that an optimal tradeo between approximation and generalization error can be achieved, assuming that optimization finds a good minimum. In the much more interesting overparametrized square loss case, generalization depends on solving a sort of *regularized ERM*, that consists of finding minimizers of the empirical risk with zero loss, and then select the one with lowest complexity [17]. Recent work [14] has provided theoretical and empirical evidence that this can be accomplished by SGD provided that the following conditions are satisfied:

1. the sparse function graph of the underlying regression function is assumed to be known and to be reflected in the architecture of the approximating network;
2. the network is overparametrized allowing zero empirical loss;
3. the loss function is the regularized (e.g. with weight decay) square loss (or an exponential loss) function.

Thus the conjecture is that this optimization problem can be solved by SGD if the graph of the underlying regression function *is known and takes the form of a compositionally sparse graph*, such as, for instance, a convolutional network.

Empirical evidence suggests that for dense networks that do not reflect the sparse graph the same problem cannot be solved using ℓ_2 minimization. Sparsity must be explicit in the architecture of the network for ℓ_2 minimization to work. Theoretical and empirical evidence points in the same direction: generalization bounds are several orders of magnitudes better for CNNs than for dense networks and close to be non-vacuous for CNNs (and presumably for other sparse networks).

The performance of trained neural networks is robust to harsh levels of pruning[5]. This empirical fact supports the hypothesis that the network should reflect the sparsity of the underlying target function. However, ℓ_2 optimization cannot attain sparsity by itself, since it preserves very small weights that should in fact be zero. Appendix 9.6 is about pruning and related issues. Empirically it seems that the graph of the target function needs to be known approximately. I conjecture that it is su cient that the sparse network contains as a subgraph the target function graph.

The conclusion is that if the sparse graph is known and approximately implemented in the architecture of the network minimization in either ℓ_2 or ℓ_1 should work. A conjecture may be

Conjecture 1. *If the sparse graph of the target function is reflected in the network and zero loss is attained then both ℓ_1 and ℓ_2 minimization lead to solutions with good expected error.*

In addition, it is likely that ℓ_1 minimization – when successful – can lead to pruned networks wrt ℓ_2 optimized networks.

[4] And even without weight decay under appropriate conditions that induce small ρ – which is the product of the Frobenius norm of the weight matrices [16].

[5] Coupled with the ever-growing size of deep learning models, this observation has motivated extensive research on learning sparse models.

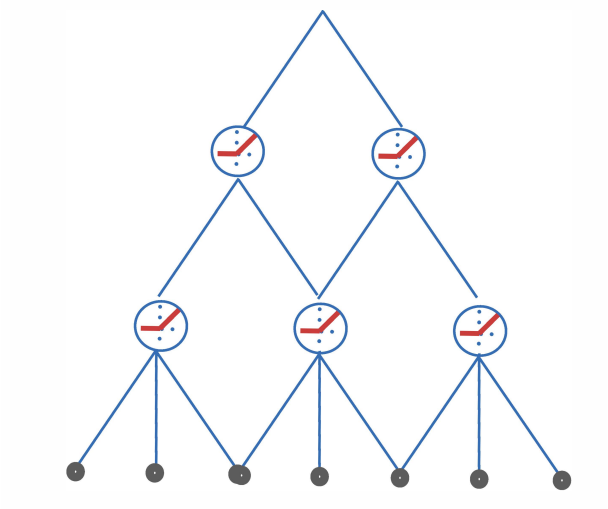

Figure 1: *The network here – similar to a CNN – reflects in a "hardwired" way the sparse compositional function graph of the target function. The function graph is supposed to be known.*

7.2 The sparse graph is unknown

The second part of the argument is about the case of unknown function graph and sparsity constraints in optimization. I propose the conjecture that when the sparse graph structure of the underlying regression function is not known, optimization with sparsity constraints is needed. In particular, two situations should be considered. The first main one is focused on dense networks under sparsity constraints, the second on transformers.

7.2.1 Dense networks optimized under sparsity constraints

For dense networks it is known that a CNN-like inductive bias can be learned from data and through training by using a modified ℓ_1 regularization. Consistent with this empirical finding, pruning of a dense network by using iterative magnitude pruning (IMP) also seems to work.

7.2.2 Self-attention as flexible sparsity

For transformers a key question is: how does self-attention find the sparse set of tokens that are input to a processing node (that is are the variables of a constituent function)? I propose the conjecture that self-attention selects, for each token, the relevant other tokens in the sequence, that is a flexible node of a hardwired CNN network. An equivalent formulation is

Conjecture 2. *Self-attention in a transformer selects a sparse subset of variables (e.g. tokens) for each RELU unit, trying to mimic the compositional sparsity of the underlying target function.*

This conjecture leaves open the interesting question of whether self-attention can deal with all compositionally sparse functions. A more likely possibility is that not all sparse functions are easy to learn by transformers.

The matrices W_Q and W_K that are set during the training time in such a way that $A = QK^T$ – with $Q = xW_Q$, $K = xW_K$ – may be together somewhat similar to a learned Malanhobis distance. In Appendix 9.7 the normalized softmax $H_D(x) = xH(x)$ (with H being a threshold on x) induces sparsity in the selection of "active" connections, preferring only a small number of very similar token – where the similarity is tuned via the learned W_H, W_Q matrices. After the attention step, there is a one-layer dense network on the linear combination of a few tokens – this is very similar to the node of a convolutional network, but with soft-wired connections instead of hard-wired.

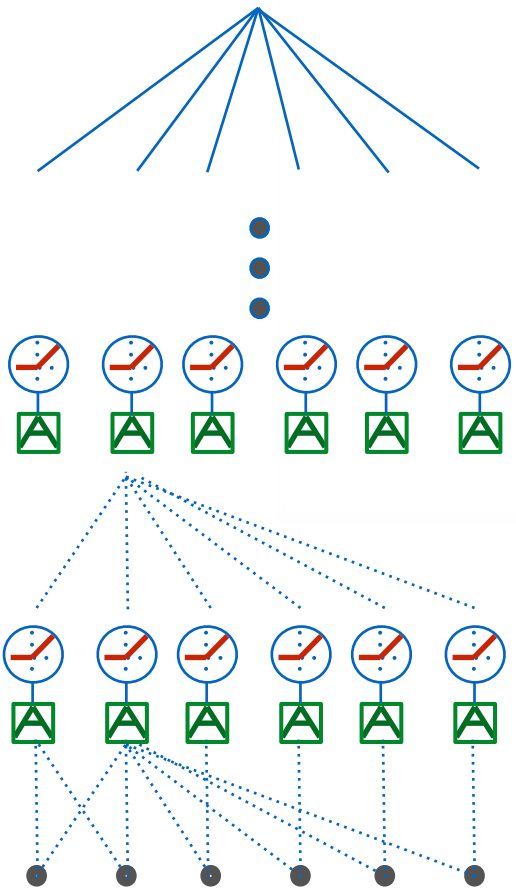

Figure 2: *Here attention (followed by a one-layer RELU network) selects for each input token its connections to other tokens, e ciently instantiating a network that reflects a compositionally sparse function graph. Each input here is a token, that is a vector, such as a patch of an image. The "A" box is the self-attention algorithm; the RELU circle represents a one-layer NN.*

8 Summary and open problems

8.1 Summary

This paper introduces a theoretical framework to explain why deep networks work and what are the properties of di erent architectures.

The *key claim* is about the world, that is about the tasks that networks could try to learn. The claim is that all functions that in practice can be approximated/computed must have a representation with the property of compositional sparsity, that is they can be represented as compositional functions with a function graph comprising sparse constituent functions, each of which has a bounded, "small" dimensionality $\frac{d}{s}$. The connection with deep networks depends on theorem 2, which claims that for functions with bounded first order derivatives *computable approximation is equivalent to compositional sparsity.*

Consider now sparse networks: if each unit in a certain layer of a deep network receives inputs from only a small subset of the units below, the corresponding weight matrix is sparse, with several zero components in each row. Somewhat surprisingly, sparsity of the network is a key property for good *generalization*. An interesting case of this sparsity is represented by standard convolutional layers with a local kernel (where the sparsity comes from locality of the kernel; convolution does not help here). The following property then holds: *the Rademacher complexity of a deep network is much smaller for convolutional deep networks with local kernels, relative to dense networks.* In complete analogy with the

approximation result the key property here is locality of the convolution kernel and not weight sharing.

From the point of view of *optimization*, two main cases should be considered: 1) the sparse graph of the underlying target functions is known, 2) the sparse graph is unknown.

1. In the overparametrized square loss case, generalization depends on solving a sort of *regularized ERM*, that consists of finding minimizers of the empirical risk with zero loss, while selecting the one with lowest complexity. Recent work has provided theoretical and empirical evidence that this can be accomplished by SGD (with norm regularization under the square loss or without regularization under an exponential loss) with weight decay in the overparametrized case when the network architecture reflects the sparse graph of the target function. This implies that this optimization problem can be solved if the graph of the underlying regression function *is known and takes the form of a compositionally sparse graph, such as, for instance, a convolutional network.*

   Empirical (and perhaps theoretical) evidence shows that for dense networks the same problem cannot be solved using ℓ_2 minimization. Sparsity must be explicit in the architecture of the network for ℓ_2 minimization to work.

2. The second case is about optimization of an unknown function graph with sparsity constraints. In particular, two situations should be considered. The main one is focused on transformers, the second on dense networks under sparsity constraints.

   - For transformers a reasonable conjecture is that the self-attention layer finds the sparse graph structure of the underlying regression function. From this point of view, the stages of self-attention can be seen as a sparsification step followed by a one-layer MLP with normalization and residual connections.
   - For dense networks it is known that a CNN-like inductive bias can be learned from data and through training by using a modified ℓ_1 regularization. Consistent with this empirical finding, pruning of a dense network by using iterative magnitude pruning (IMP) after ℓ_2 optimization also seems to work.

8.2 Future directions

- Computable functions as defined here are "in practice all" functions in the same sense that the Church-Turing thesis is accepted.
- The results outlined here imply that any e□ciently computable function is e□ciently computable by a compositionally sparse deep network and that any such function is compositionally sparse.
- They suggest the conjecture that when the sparse graph of the compositional target function is known, optimization and generalization are possible in many cases (but certainly not always)!
- They also suggest the conjecture that, under conditions to be characterized, self-attention may be capable of finding the sparse graph of the target function (for some functions classes) without a priori knowledge of it.
- Suppose the first two points above are correct. Then the key problem in learning is to generate a good hypothesis of the sparse compositional graph underlying the specific task or class of tasks (such as deep convolutional graphs for vision tasks).
- Consider the third point. It is very likely that the self-attention mechanism does not work for all computable functions but only for a subset of them. Which one? Furthermore it is very likely that there exist better, more general algorithms able to identify an unknown graph than self-attention (in the way it is currently implemented). What are these algorithms?
- This paper suggests that every computable function has at least one sparse compositional representation (as composition of sparse functions). The key task for learning would then be finding this sparse compositional representation.

Summary Why do deep networks work as well as they do? The answer I propose here is that certain deep architectures – such as CNNs and transformers – exploit a general property of all e□ciently computable (smooth) functions: their compositional sparsity.

Acknowledgments I thanks Fabio Anselmi, Sophie Langer, Tomer Galanti, Akshay Rangamani, Shimon Ullman, Yaim Cooper, Gitta Kutyniok, Lorenzo Rosasco and the Compositional Sparsity (CoSp) Collaboration (Santosh Vempala, Hrushikesh Mhaskar, Eran Malach, Seth Lloyd) for illuminating discussions. This material is based upon work supported by the Center for Minds, Brains and Machines (CBMM), funded by NSF STC award CCF-1231216. This research was also sponsored by grants from the National Science Foundation (NSF-0640097, NSF-0827427), AFSOR-THRL (FA8650-05-C-7262) and Lockeed-Martin.

References

[1] Wolfgang Dahmen. Compositional sparsity, approximation classes, and parametric transport equations, 2022.

[2] Michael Kohler and Sophie Langer. Discussion of: "Nonparametric regression using deep neural networks with ReLU activation function". *The Annals of Statistics*, 48(4):1906 – 1910, 2020.

[3] Johannes Schmidt-Hieber. Nonparametric regression using deep neural networks with ReLU activation function. *The Annals of Statistics*, 48(4):1875 – 1897, 2020.

[4] Markus Bachmayr, Anthony Nouy, and Reinhold Schneider. Approximation by tree tensor networks in high dimensions: Sobolev and compositional functions, 2021.

[5] Gitta Kutyniok. Discussion of: "Nonparametric regression using deep neural networks with ReLU activation function". *The Annals of Statistics*, 48(4):1902 – 1905, 2020.

[6] H.N. Mhaskar and T. Poggio. Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, pages 829– 848, 2016.

[7] H. Mhaskar. Dimension independent bounds for general shallow networks. *Neural Networks*, 2020.

[8] Alexander Bastounis, Anders C Hansen, and Verner Vlačić. The extended smale's 9th problem – on computational barriers and paradoxes in estimation, regularisation, computer-assisted proofs and learning, 2021.

[9] Holger Boche, Adalbert Fono, and Gitta Kutyniok. Limitations of deep learning for inverse problems on digital hardware, 2022.

[10] T. Galanti and T. Poggio. Sgd noise and implicit low-rank bias in deep neural networks. *Center for Brains, Minds and Machines (CBMM) Memo No. 134*, 2022.

[11] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree, 2018.

[12] Wolfgang Hackbusch and Stephan Kühn. A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15:706–722, 2009.

[13] Lars Grasedyck. Hierarchical Singular Value Decomposition of Tensors. *SIAM J. Matrix Anal. Appl.*, (31,4):2029–2054, 2010.

[14] M. Xu, A. Rangamani, A.and Banburski, Q.and Galanti Liao, T., and T. Poggio. Dynamics and neural collapse in deep classifiers trained with the square loss. CBMM Memo 117, CBMM, MIT, 2022.

[15] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Theory I: Why and when can deep - but not shallow - networks avoid the curse of dimensionality. Technical report, CBMM Memo No. 058, MIT Center for Brains, Minds and Machines, 2016.

[16] Mengjia Xu, Akshay Rangamani, Qianli Liao, Tomer Galanti, and Tomaso Poggio. Dynamics in deep classifiers trained with the square loss: normalization, low rank, neural collapse and generalization bounds. *Research*, 2023.

[17] Eran Malach and Tomaso Poggio. Compositional locality and optimization. *CBMM Memo*, 2023.

[18] A. N. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk SSSR*, 114:953–956, 1957.

[19] V.I. Arnol'd. On functions of three variables. *Dokl. Akad. Nauk SSSR*, 114:679–681, 1957.

[20] J.P. Kahane. Sur le theoreme de superposition de Kolmogorov. *Journal of Approximation Theory*, 13:229–234, 1975.

[21] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[22] GG Lorentz. Approximation of functions, athena series. *Selected Topics in Mathematics*, 1966.

[23] A. G. Vitushkin and G.M. Henkin. Linear superposition of functions. *Russian Math. Surveys*, 22:77–125, 1967.

[24] A. G. Vitushkin. On Hilbert's thirteenth problem. *Dokl. Akad. Nauk SSSR*, 95:701–704, 1954.

[25] Johannes Schmidt-Hieber. The kolmogorov-arnold representation theorem revisited. *CoRR*, abs/2007.15884, 2020.

[26] Hrushikesh Narhar Mhaskar. Approximation properties of a multilayered feedforward artificial neural network. *Advances in Computational Mathematics*, 1(1):61–80, 1993.

[27] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: a tensor analysis. *CoRR*, abs/1509.0500, 2015.

[28] Behnam Neyshabur. Towards learning convolutions from scratch. *CoRR*, abs/2007.13657, 2020.

[29] Franco Pellegrini and Giulio Biroli. Sifting out the features by pruning: Are convolutional networks the winning lottery ticket of fully connected ones? *CoRR*, abs/2104.13343, 2021.

[30] Stéphane d'Ascoli, Levent Sagun, Joan Bruna, and Giulio Biroli. Finding the needle in the haystack with convolutions: on the benefits of architectural bias, 2019.

[31] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.

[32] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, September 1990.

[33] R. Brunelli and T. Poggio. HyperBF Networks for Gender Classification. In *Proceedings of the Image Understanding Workshop*, pages 311–314, San Mateo, CA, 1992. Morgan Kaufmann.

[34] T. Poggio. A theory of how the brain might work. In *Cold Spring Harbor Symposia on Quantitative Biology*, pages 899–910. Cold Spring Harbor Laboratory Press, 1990.

[35] Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. *CoRR*, abs/1712.06541, 2017.

[36] Patrick Rebeschini. Algorithmic foundations of learning lecture 3: Rademacher complexity, 2020.

9 Appendices

9.1 Compositionality and sparsity

9.1.1 All continuous function are compositionally (non-smooth) sparse

An obvious question is how "big" is the class of compositionally sparse functions wrt the class of all continuous functions (all functions are trivially compositional, since every function can be composed with the identity function). An answer for continuous functions was given by the solution of Hilbert's thirteen problem due to Kolmogorov and Arnold [18, 19, 20]: every continuous functions can be represented *exactly* as a compositions of $poly(d)$ functions of one variable, that is as the composition of sparse functions.

Theorem 5. *All continuous functions are compositionally sparse, that is they have an exact representation in terms of sparse non-smooth constituent functions.*

In this representation, the constituent functions are very non-smooth, that is $s = 0$. Appendix 9.3 has more information. This fact implies that the Kolmogorov-Arnold representation is not e□ciently computable in the sense that continuous functions cannot be approximated with non-exponential rates.

9.1.2 Compositional S-sparsity implies computable approximation

Let us first define smooth sparse functions.

Definition 3. *A S-sparse (e.g. smoothly sparse) compositional function of d variables is a sparse compositional function with constituent functions that have bounded first derivatives.*

With this definition, we can then reformulate the MP theorem (see Appendix) as

Theorem 6. *Compositionally S-sparse functions are e ciently computable.*

9.1.3 Computable approximation is not equivalent to S-sparsity

Networks with non-smooth RELU can approximate arbitrarily well S-sparse functions. They are compositionally sparse but not smooth. Thus one can imagine, in a theorem such as d'Arzela'-Ascoli, that there is a sequence of sparse compositional f_n – provided by RELU networks – converging to a smooth compositionally sparse f without the f_n being smooth themselves! CAN THIS BE TRUE?

9.2 Are computable functions compositionally sparse?

Let us state an obvious conjecture.

Conjecture 3. *Functions with an e ciently computable approximation are compositionally sparse.*

If the above were true, the story would take the following form

Conjecture 4. *Compositional S-sparse functions have an e ciently computable approximation; functions with an e ciently computable approximation are compositionally sparse.*

This would mean that the class of computable functions is larger than the class of functions for which we can guarantee e□cient approximation.

9.2.1 Remarks

- The curse of dimensionality bound is an upper bound but a similar lower bound holds for a set of functions that has large measure, see Pinkus [21] comments about Maiorov's results.
- A definition of sparse compositional functions must include an implicit or explicit constraint on the number of nodes in the associated DAG, that is on the number of constituent functions. In the specific example of a binary tree graph the depth of the graph increases only logarithmically as the dimensionality d increases.
- As emphasized by H. Mhaskar, just the degree of approximation theorem is too crude a tool to use - one has to add that the approximation must be constructive, based on values of the target function. Otherwise, as shown in [7] where for the first time both dimension independent as well as constructive bounds for the same class of functions are proven), ReLU networks can achieve dimension independent bounds, obviating the need to have deep networks from the point of view of degree of approximation alone.
- The curse of dimensionality holds not only for real-valued continuous functions but also for Boolean functions. A specific constructions of relevant Boolean functions is discussed in the Appendix (see 9.5.1 and [15]).
- Because of theorem 9 (see also [15]) all compositionally S-sparse, continuous, real-valued functions can be approximated by a Boolean compositionally sparse function.
- Computable by a Turing machine usually doesn't assume polynomial time/space complexity, but the term *e ciently computable* used in this paper implies polynomial time/space requirements.
- All compositionally S-sparse functions are e□ciently computable by a Turing machine and admit an e□cient approximator in terms of a deep RELU network with an architecture reflecting the sparse function graph.
- E□cient approximation can be thought of as the computation by a Turing machine of a Boolean function. Such a Boolean function is the composition of the Boolean functions that approximate the constituent functions. The complexity of the resulting function is at most $O(poly(d))$.
- Observe that a non-sparse continuous function $f : \mathcal{R}^{1000} \to \mathcal{N}$ requires a memory of up to $> 10^{1000}$ bits, larger than the number of protons in the Universe, which is in the order of 10^{80}. A classical example is a dense polynomial in d dimensions of arbitrarily high degree.
- There are (compositional) functions with constituent functions that are not smooth and are e□ciently computable. An example of such functions are the Boolean functions that are ultimately used in a Turing machine to represent (and approximate) a continuous smooth function. Another example of (compositional) functions with constituent functions that are not smooth are some of the functions in the Takagi class (see Appendix). Furthermore, deep RELU networks are compositionally sparse but non smooth functions.
- Stability of the approximation in probability wrt perturbations of the inputs seems an important requirement for any function and its approximation. This seems to imply smoothness of the constituent functions when they are continuous.

9.3 Kolmogorov's theorem[18]

Theorem 7. *(Kolmogorov, 1957; see also [22]). There exist fixed (universal) increasing continuous functions $h_{pq}(x)$, on $I = [0, 1]$ so that each continuous function f on I^d can be written in the form*

$$f(x_1, \ldots, x_d) = \sum_{q=1}^{2d+1} g_q \left(\sum_{p=1}^{d} h_{pq}(x_p) \right),$$

where g_q are properly chosen continuous functions of one variable.

This result asserts that every multivariate continuous function can be represented by the superposition of a small number of univariate continuous functions. In terms of networks this means that every continuous function of many variables can be computed by a network with two hidden layers, whose hidden units compute continuous functions (the functions g_q and h_{pq}).

The interpretation of Kolmogorov's theorem in term of networks is very appealing: the representation of a function requires a fixed number of nodes, polynomially increasing with the dimension of the input space. Unfortunately, these results are somewhat pathological and their practical implications are very limited. The problem lies in the inner functions of Kolmogorov's formula: although they are continuous, theorems of Vitushkin and Henkin [23] prove that they must be highly non-smooth. One could ask if it is possible to find a superposition scheme in which the functions involved are smooth. The answer is negative, even for two variable functions, and was given by [24] with the following theorem:

Theorem 8. *(Vitushkin 1954). There are $r(r = 1, 2, \ldots)$ times continuously di erentiable functions of $n \geq 2$ variables, not representable by superposition of r times continuously di erentiable functions of less than n variables; there are r times continuously di erentiable functions of two variables that are not representable by sums and continuously di erentiable functions of one variable.*

Recent interesting extensions of Kolomogorov's theorem are due to [25].

9.4 Boolean functions

One of the most important tools for theoretical computer scientists for the study of computable functions is the study of Boolean functions, that is functions of n Boolean variables. A key tool here 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. 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.

9.4.1 *Boolean Functions and Sparsity*

The curse of dimensionality holds not only for real-valued continuous functions but also for Boolean functions (see discussion in [15]). The following theorem states that compositionally sparse Boolean functions avoid the curse.

Theorem 9. *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 $f : \{1, -1\}^n \mapsto \mathbb{R}$ be a compositional $\mathcal{G}$-function, where each of the constituent functions is a function g in d_v Boolean variables, $g : \{1, -1\}^d \mapsto \{1, -1\}$. Consider shallow and deep networks with a RELU activation functions or a hard threshold. Then deep networks - with an associated graph that corresponds to the graph of f - can avoid the curse of dimensionality in approximating f for increasing d, since the number of required parameters is $\propto \mathcal{O}(poly(d))(max_v d_v)$.*

The converse results from the observation that the Fourier representation of a Boolean function in d variables can have up to N non-zero monomials where $N = \binom{2d}{d} = \frac{2d!}{d!d!}$. Thus $N > 2^d$. Clearly a Boolean functions with all non-zero monomials is not e□ciently computable. In fact the following holds

Theorem 10. *All e ciently computable Boolean functions are compositionally sparse, that is they can be represented as the composition of a $\leq poly(d)$ number of constituent functions with a bounded "small" dimensionality.*

Combining the previous two theorems we obtain the following

Theorem 11. *For Boolean functions, e ciently computable is equivalent to compositionally sparse.*

9.5 Spline approximations, Boolean functions and tensors

Consider the case of a multivariate smooth function $f : [0,1]^d \to \mathbf{R}$. Suppose to discretize it by a set of piecewise constant splines and their tensor products[6]. Each coordinate is e□ciently 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 $f : \{0,1\}^N \to \mathbf{R}$.

- Every smooth function f can be approximated by an epsilon-close binary function f_B. Binarization of $f : \mathbf{R}^n \to \mathbf{R}$ is done by using k partitions for each variable x_i and indicator functions. Thus $f \mapsto f_B : \{0,1\}^{kn} \to \mathbf{R}$ 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.
- 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 [27]). This work, as well more classical papers [13], does not characterize directly the class of functions for which these decompositions are e□ective approximations. Notice that tensor decompositions *assume* that the sum of polynomial functions of order d is sparse (see eq. at top of page 2030 of [13]). Our results provide a rigorous grounding in terms of approximation theory for papers on tensors related to deep learning. There is obviously a wealth of interesting connections with approximation theory that should be explored.

9.5.1 *On multivariate function approximation*

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

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

with $\phi_{i_\mu} : [0,1] \to \mathbf{R}, 1 \leq i_\mu \leq n_\mu, 1 \leq \mu \leq d$
to provide

$$f(x_1,...,x_d) = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} c(i_1,...,i_d)\phi(i_1,...,i_d)(x_1,...,x_d). \tag{2}$$

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. We describe nest 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.

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:

[6] An argument similar to the one below for polynomials was used by Mhaskar[26] to show that a multivariate tensor product spline can be synthesized exactly using a deep network with activation function $(x_+)^2$; with Yarotsky's theorem, this can be translated into deep networks with ReLU functions without incurring in saturation phenomena.

1. Let us denote with $\mathcal{H}_k$ the linear space of homogeneous polynomials of degree k in $\mathbf{R}^n$ 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 $((w, x) + b)^N$.

   We first prove that

$$P_k(x) = span(((w^i, x))^s : i = 1, \cdots, r,\ s = 1, \cdots, k \tag{3}$$

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

$$P_k(x) = \sum_{i=1}^{r} p_i((w_i, x)). \tag{4}$$

   Notice that the e□ective 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

$$\mathcal{N}(\sigma) = span\{\sigma(\lambda t - \theta)\}, \lambda, \theta \in \mathbf{R} \tag{5}$$

   in an appropriate norm.

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

$$E(\mathcal{B}_p^m; P_k; L_p) \leq Ck^{-m} \tag{6}$$

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

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 $c_i, b_i, \lambda_i \in \mathbf{R}$ the space of univariate functions

$$\mathcal{N}_r(\sigma) = \left\{ \sum_{i=1}^{r} c_i \sigma(\lambda_i x - b_i) \right\}. \tag{7}$$

The following (see Propositions 3.6 and 3.8 in [21]) holds

Lemma 1. *If $\sigma \in \mathcal{C}()$ 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 6 gives

$$E(\mathcal{B}_p^m; P_k; L_p) \leq Cr^{-\frac{m}{n}}. \tag{8}$$

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 9.5.1 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 $f \in^2 [a, b]$ 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 su□ciently 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.

Boolean functions and curse of dimensionality The classical curse of dimensionality result 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.

A network with non-smooth ReLU activation functions can approximate any continuous function. A weakness of this results wrt to other ones 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 a compositional function with a function graph given by a 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. This is the di□erence between fixed-knots splines and free-knots splines.

I 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 result (for the special case of a compositionally sparse function with a binary tree function graph):

Lemma 2. *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 max of the Lipschitz constants 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[7].

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.

9.6 Pruning

Empirically it seems that dense networks cannot learn convolution under L_2 minimization but can under L_1 minimization. In particular, the possibility of learning CNN-like inductive bias from data and through training was investigated in [28]. It was shown that training using a modified L_1 regularization is a way to induce local masks for visual tasks. Consistent with this finding, pruning of a dense network by using iterative magnitude pruning (IMP) on FCNs trained on a low resolution version of ImageNet uncovers (see [29]) sub-networks characterized by local connectivity, especially in the first hidden layer, and masks leading to local features with patterns very reminiscent of the ones of trained CNNs[8].

[7] 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.

[8] Deeper layers are made up of these local features with larger receptive fields hinting at the hierarchical structure found in CNNs. Pruning induces locality also beyond the first hidden layer. Their remark "These results highlight the role of the task in shaping the properties of the network obtained by pruning: only for the task that the network can efficiently learn, and not just memorize, local features emerge..." is consistent with our hypothesis of compositional sparsity of the underlying task, in this case a visual task.

This is similar to the following empirical result: enforcing sparsity during training leads to structures characterized by locality. [30] studies the role of CNN-like inductive biases by embedding convolutional architectures within the general FCN class. It shows that enforcing CNN-like features in an FCN can improve performance even beyond that of its CNN counterpart. Finally, [31] show that by considering a particular multilayer perceptron architecture, called MLP-mixer, some of the CNN features can be learned from scratch using a large training dataset.

9.7 Transformers

$X \in \mathcal{R}^{T,d_{in}}$; $Q = XW_Q$ with $W_Q \in \mathcal{R}^{d_{in},d_k}$; $K = XW_K$ with $W_K \in \mathcal{R}^{d_{in},d_k}$; $V = XW_V$ with $W_V \in \mathcal{R}^{d_{in},d_{out}}$

Notice that the standard formulation of the transformer layers can be written as

$$y = x + MLP(LayerNorm(x + Attention(LayerNorm(x)))$$

.

9.7.1 Transformers as associative memories

Consider $AX = Y$. The best A is given by

$$A = YX^T(XX^T)^{-1}. \tag{9}$$

If $(XX^T)^{-1} \approx I$ – which happens for noiselike X – then $A = YX^T$ implying $Ax = YX^Tx$. Typically the dimensionality of the columns of X is large to allow for the noiselike property (and sparsity).

Transformers transform input matrices into output matrices of the same dimensionality for instance a German sentence into a French one. In other words, functions implemented by self-attention map from $R^{T,d}$ to itself, so that instances from this function class can be composed. This is important for compositionality in *compositional sparsity*. It is also important in the use of transformers a sequence of associations from an input x' to an output $x"$ which is then used for another association. x' could be a sentence with a missing word and $x"$ its completion.

The idea of associative memory is consistent with the interpretation of the self-attention layer as a learned, di□erentiable lookup table. The Q, K, and V are described as "queries," "keys," and "values" respectively, which seem to invoke such an interpretation. Consider only one attentional head. Each object or token x_i has a query $Q(x_i)$ that it will use to test "compatibility" with the key $K(x_j)$ of each object x_j. Compatibility of x_i with x_j is defined by the inner product $Q(x_i), K(x_j)$; if this inner product is high, then x_i 's query matches x_j key and so we look up x_js value $V(x_j)$. We construct then a soft lookup of values compatible with x_i's key: we sum up the value of each object x_j proportional to the compatibility of x_i with x_j.

9.7.2 Self-attention: similarity with HyperBF

The attention mechanism has been widely used in many sequence modeling tasks. Its dot-product variant is the key building block for the state-of-the-art transformer architectures [**?**]. Let $\mathbf{q}_t$ denote a query vector, that attends to sequences of L pairs $\mathbf{k}_i, \mathbf{v}_i$ of key and value vectors (see Figure 2). At each timestep, the attention linearly combines the values weighted by the outputs of a softmax:

$$\mathrm{attn}\left(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}\right) = \sum_i \frac{\exp\left(\mathbf{q}_t \cdot \mathbf{k}_i/\sigma^2\right)}{\sum_j \exp\left(\mathbf{q}_t \cdot \mathbf{k}_j/\sigma^2\right)} \mathbf{v}_i^\top. \tag{10}$$

σ^2 is the hyperparameter determining how "flat" the softmax is. Calculating attention for a single query takes $\mathcal{O}(L)$ time and space; complexity for a whole attention layer has complexity $\mathcal{O}(L^2)$.

Interestingly, such an attention layer is equivalent to a set of HyperBF networks[32, 33]. Assume that $\mathbf{q}_t$ and $\mathbf{k}_i$ are normalized that is $\|\mathbf{q}_t\| = 1$ and $\|\mathbf{k}_i\| = 1$. Then $1 - \frac{\|\mathbf{q}_t - \mathbf{k}_i\|^2}{2} = \mathbf{q}_t \cdot \mathbf{k}_i$ and

$$\mathrm{attn}\left(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}\right) = \sum_i \frac{\exp(-\frac{\|\mathbf{q}_t - \mathbf{k}_i\|^2}{2\sigma^2})}{\sum_j \exp(-\frac{\|\mathbf{q}_t - \mathbf{k}_j\|^2}{2\sigma^2})} \mathbf{v}_i^\top. \tag{11}$$

Furthermore, define $\mathbf{q}_t = W_Q x$ and $\mathbf{k}_i = W_K y$, where the matrices W_Q and W_K that are learned during training have the same dimensions (in the case of self-attention). Then $\mathbf{q}_t \cdot \mathbf{k}_i = x^T W_Q^T W_K y$ and $\|\mathbf{q}_t - \mathbf{k}_i\|^2 = 2 - x^T W_Q^T W_K y - y^T W_K^T W_Q x$ assuming $\|W_Q\| = 1$, $\|W_K\| = 1$. The matrix $M = W_Q^T W_K$ is symmetric and $\|\mathbf{q}_t - \mathbf{k}_i\|^2$ is a Mahalanobis distance. As an example, a diagonal M, with some 0's on the diagonal, e□ectively "switches o□" specific components of the vectors from the similarity estimation. This extension of Radial Basis Functions with a trainable M was called HyperBF in [32] and described as one of the ways the brain may learn and compute in [34]. The fact that a HyperBF network can be used to implement self-attention is consistent with architectures such as the MLP mixer[**?**]. Sparseness in a RELU network such as depicted in Figure **??** is encoded in sparse weight matrices. The equivalent HyperBF formulation has a sparse diagonal M matrix.

The attention layer satisfies an additional sparsity requirement since a transformer can be thought as a set of RELU networks each operating on a token provided by a selfattention gate. Thus a transformer operates on a large sequence of token vectors. The softmax can induce sparsity in the selection of "active" connections by an attention head, selecting only a small number of very similar tokens – where the similarity is tuned via the learned W_K, W_Q matrices or equivalently by the M matrix. This sparsity corresponds to e□ectively switching-o□ the tokens that are not close enough in Mahalanobis distance to the center of the HyperBF corresponding to each unit of the attention layer. After the attention step, there is a two-layers dense RELU network (see Figure 2) acting on the output of the attention unit which is a vector with the same dimensionality as the selected input tokens of which it is a linear combination. This is similar to the node of a convolutional network, but with soft-wired connections instead of hard-wired.

9.8 Generalization bounds for Sparse Networks

The classical bounds are for generic deep networks. In such a general case, ρ in those bounds is the product of the Frobenius norms of all the weight matrices. For convolutional networks the weight matrices are Toeplitz matrices. This gives large bounds.

Here we show that the bound on the Rademacher complexity can be reduced by exploiting two typical properties of CNNs: a) the locality of the convolutional kernels and b) shared weights. They allow us to use only the norm of the kernels in the calculation of ρ_k instead of the norm of the corresponding Toeplitz matrix. In this section we give an outline of the results with more precise statements and proofs to be published later.

We start by considering the simple situation of non-overlapping convolutional patches. In other words, the stride of the convolution is equal to the size of the kernel in each layer. This means that in the associated Toeplitz matrix the non-zero components in each row do not overlap with the non-zero components of the row above or the one below. In other words, if K is the number of patches, ℓ is the size of each patch and $x \in \mathbb{R}^d$, then $d = K\ell$. Notice that the standard bounds give a Rademacher complexity proportional to the product of the Frobenius norms of each weight matrix $\|W\|$ time the norm of $\|x\|$, where $\|W\| \propto \sqrt{k}M$, where M is the norm of the kernel.

In [14] we describe generic bounds on the Rademacher complexity of deep neural networks. In these cases, ρ measures the product of the Frobenius norms of the network's weight matrices in each layer. For convolutional networks, however, the operation in each layer is computed with a kernel, described by the vector w, that acts on each patch of the input separately. Therefore, a convolutional layer is represented by a Toeplitz matrix W, whose blocks are each given by w. In this section (from [14]) we provide an informal analysis of the Rademacher complexity, showing that it can be reduced by exploiting mainly the first one of the two properties of convolutional layers: (a) the locality of the convolutional kernels that is the sparsity of the associated Toeplitz matrix and (b) weight sharing. These properties allow us to bound the Rademacher complexity by taking the products of the norms of the kernel w instead of the norm of the associated Toeplitz matrix W. Here we outline the results with more precise statements and proofs to be published separately.

We consider the case of 1-dimensional convolutional networks with non-overlapping patches and one channel per layer. For simplicity, we assume that the input of the network lies in $\mathbb{R}^d$, with $d = 2^L$ and the stride and the kernel of each layer are 2. The analysis can be easily extended to kernels of di□erent sizes. This means that the network $h(x)$ can be represented as a binary tree, where the output neuron is computed as $W^L \cdot \sigma(v_1^L(x), v_2^L(x))$, $v_1^L(x) = W^{L-1} \cdot \sigma(v_1^{L-1}(x), v_2^{L-1}(x))$ and $v_2^L(x) = W^{L-1} \cdot \sigma(v_3^{L-1}(x), v_4^{L-1}(x))$ and so on. This means that we can write the i'th row of

the Toeplitz matrix of the l'th layer $(0,\ldots,0,-W^l-,0\ldots,0)$, where W^l appears on the 2^i-1 and 2^i coordinates. We define a set $\mathcal{H}$ of neural networks of this form, where each layer is followed by a ReLU activation function and $\prod_{l=1}^{L} W^l \leq \rho$.

Theorem 12. *Let $\mathcal{H}$ be the set of binary-tree structured neural networks over $\mathbb{R}^d$, with $d=2^L$ for some natural number L. Let $X=\{x_1,\ldots,x_m\}\subset\mathbb{R}^d$ be a set of samples. Then,*

$$\mathcal{R}_X(\mathcal{H}) \leq \frac{2^L \rho \sqrt{\sum_{i=1}^{m}\|x_i\|^2}}{m} \tag{12}$$

Proof sketch. First we rewrite the Rademacher complexity in the following manner:

$$\begin{aligned}\mathcal{R}_X(\mathcal{H}) &= \mathbb{E}_\epsilon \sup_{h\in\mathcal{H}} |\frac{1}{m}\sum_{i=1}^{m}\epsilon_i\cdot h(x_i)| \\ &= \mathbb{E}_\epsilon \sup_{h\in\mathcal{H}} \frac{1}{m}|\sum_{i=1}^{m}\epsilon_i\cdot W^L\cdot\sigma(v_1(x),v_2(x))| \\ &= \mathbb{E}_\epsilon \sup_{h\in\mathcal{H}} \frac{1}{m}\sqrt{|\sum_{i=1}^{m}\epsilon_i\cdot W^L\cdot\sigma(v_1(x),v_2(x))|^2}\end{aligned} \tag{13}$$

Next, by the proof of Lem. 1 in [35], we obtain that

$$\begin{aligned}\mathcal{R}_X(\mathcal{H}) &\leq 2\mathbb{E}_\epsilon \sup_{h\in\mathcal{H}} \frac{1}{m}\sqrt{\|W^L\|^2\cdot\|\sum_{i=1}^{m}\epsilon_i(v_1(x),v_2(x))\|^2} \\ &= \mathbb{E}_\epsilon \sup_{h\in\mathcal{H}} \frac{1}{m}\sqrt{\|W^L\|^2\cdot\sum_{j=1}^{2}\|\sum_{i=1}^{m}\epsilon_i v_j(x_i)\|^2}\end{aligned} \tag{14}$$

By applying this peeling process L times, we obtain the following inequality:

$$\begin{aligned}\mathcal{R}_X(\mathcal{H}) &\leq 2^{L-1}\mathbb{E}_\epsilon \sup_{h\in\mathcal{H}} \frac{1}{m}\sqrt{\prod_{l=1}^{L}\|W^l\|^2\cdot\sum_{j=1}^{d}\|\sum_{i=1}^{m}\epsilon_i x_{ij}\|^2} \\ &= 2^{L-1}\mathbb{E}_\epsilon \sup_{h\in\mathcal{H}} \frac{1}{m}\sqrt{\prod_{l=1}^{L}\|W^l\|^2\cdot\|\sum_{i=1}^{m}\epsilon_i x_i\|^2} \\ &\leq \frac{2^{L-1}\rho\mathbb{E}_\epsilon\|\sum_{i=1}^{m}\epsilon_i x_i\|}{m} \\ &\leq \frac{2^{L-1}\rho\sqrt{\sum_{i=1}^{m}\|x_i\|^2}}{m}\end{aligned} \tag{15}$$

where the factor 2^{L-1} is obtained because the last layer is linear (see [36]). We note that a better bound can achieved when using the reduction introduced in [35] which would give a factor of $\sqrt{2\log(2)L}+1$ instead of 2^{L-1}. □

One-layer convolutional classifier Consider a ReLU convolutional classifier with k patches. $\hat{\mathcal{R}}_m$, in the standard bounds would be

$$\hat{\mathcal{R}}_m \leq BX$$

where B is the Frobenius norm of the Toeplitz matrix with k rows, each row consisting of the kernel w. Thus $B=\sqrt{K}\|w\|$ and $X=\|x\|$.

Our calculation gives with x^1 representing the first patch of x and x^K the last one:

$$\hat{\mathcal{R}}_m \leq \sqrt{\|w\|^2\|x^1+\cdots+x^K\|^2} = \sqrt{\|w\|^2\|x\|^2} = \|w\|\|x\|.$$

instead of the general bound usually referred which is

$$\hat{\mathcal{R}}_m \leq \|W\|\|x\| = \sqrt{k}\|w\|\|x\|$$

Multi-layer convolutional classifier The Rademacher complexity of a feed-forward neural network can be bounded recursively by considering each layer at a time. A bound that can be used for the recursion is given by the following proposition (see [36, 35]), that expresses the Rademacher complexities at the outputs of one layer in terms of the outputs at the previous layers.

Lemma 3. *Let $\mathcal{H}$ be a class of functions from $\mathbb{R}^d$ to $\mathbb{R}$. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be the ReLU function which is 1-Lipschitz and define $\mathcal{H}' := \left\{x \in \mathbb{R}^d \to \sigma\left(\sum_{j=1}^k w_j h_j(x)\right) \in \mathbb{R} : \|w\|_2 \leq M, h_1, \ldots, h_k \in \mathcal{H}\right\}$. Then, for any $x_1, \ldots, x_m \in \mathbb{R}^d$*

$$\mathcal{R}\left(\mathcal{H}' \circ \{x_1, \ldots, x_m\}\right) \leq 2M\mathcal{R}\left(\mathcal{H} \circ \{x_1, \ldots, x_m\}\right).$$

We apply now the Lemma to the class of L-depth ReLU real-valued CNN, with each layer's kernel w_d with norm at most M_d.

Theorem 13. *(informal) The Rademacher complexity of a convolutional deep net with RELUs in all d layers but the last linear one and with non-overlapping convolutional patches can be bounded as*

$$\mathcal{R}_m(\mathcal{H}_d) \leq (\hat{\sqrt{2\log(2)L}} + 1) \prod_{j=1}^{L} M_j \|x\| \tag{16}$$

Proof sketch. Each $h_k^d \in \mathcal{H}_\lceil$ ($k = 1, \cdots, Q$ is a ReLU classifier inputs from patch j of the layer below. Patch k in layer $d-1$ can be written as a vector v_k consisting of ℓ classifiers $v_k = h_{k\cdot 1}^{d-1}, h_{k\cdot 2}^{d-1}, \cdots, h_{k\cdot \ell}^{d-1}$. Then $h_k^d = \sigma(w \cdot v_k)$. Notice that because of our assumption of non-overlapping patches the number of units in layer $d-1$ is ℓ times the number of units in layer d. Then

$$\hat{\mathcal{R}}_m\left(\mathcal{H}_d\right) = \mathbb{E}_\epsilon \sup_{h_i \in \mathcal{H}_d} \frac{1}{m} \sum_{i=1}^m \epsilon_i h_i = \mathbb{E}_\epsilon \sup_{h_i \in \mathcal{H}_{d-1} w : \|w\| \leq M} \frac{1}{m} \sum_{i=1}^m \epsilon_i w \cdot (\sum_k v_k), \tag{17}$$

can be upper bounded as follows

$$\hat{\mathcal{R}}_m\left(\mathcal{H}_d\right) \leq 2M_d \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}_d} \|\frac{1}{m} \sum_{i=1}^m \epsilon_i (\sum_k (v_k)_i)\| \leq \frac{1}{m} \sqrt{(w \cdot (\sum_k v_k)^2} = \frac{1}{m} \sqrt{(w \cdot (\sum_k v_k)^2} = 2M_d \hat{\mathcal{R}}_m\left(\mathcal{H}_{d-1}\right), \tag{18}$$

because $(\sum_k v_k)^2 = \sum_k v_k^2$ since the various patches are zero-mean and uncorrelated.
Continuing the peeling we obtain

$$\mathcal{R}_m(\mathcal{H}_L) \leq 2^{L-1} \hat{M_L} \cdot M_{L-1} \cdots M_1 \|x\|, \tag{19}$$

As before, we can further apply the reduction used by [35] to finally obtain the result. □

Thus one ends up with a bound scaling as the product of the norms of the kernel at each layer. The constants may change depending on the architecture, the number of patches, the size of the patches and their overlap.

This special non-overlapping case can be extended to the general convolutional case:

Conjecture 5. *If a convolutional layer has overlaps among its patches then the bound*

$$\mathcal{R}_m(\mathcal{H}_L) \leq 2^{L-1} \hat{M_L} \cdot M_{L-1} \cdots M_1 \|x\|$$

holds with $\|x\|$ replaced by

$$\|x\| \sqrt{\frac{K}{K-O}},$$

where K is the size of the kernel (number of components) and O is the size of the overlap.

Sketch proof Call P the number of patches and O the overlap. With no overlap then $PK = D$ where D is the dimensionality of the input to the layer. In general $P = \frac{D-O}{K-O}$. It follows that a layer with the most overlap can add at most $< \|x\|\sqrt{K}$ to the bound. Notice that we assume that each component of x_i averaged across i will have norm $\sqrt{\frac{1}{d}}$.