CENTER FOR
Brains
Minds+
Machines

CBMM Memo No. 138 **December 6, 2022**

Compositional Sparsity: a framework for ML

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

Abstract

I propose a theoretical framework that aims to explain why deep networks work and what are the properties of different architectures. The framework describes a few results and conjectures. The core claim is that compositional sparsity of the underlying target function (i.e. regression function), which corresponds to the task to be learned, is the key principle of machine learning. Furthermore I propose the thesis (with S. Lloyd) that *all learnable functions must be compositionally sparse*. Sparsity of the target functions then naturally leads to sparse networks and sparsity-biased optimization techniques. I argue that this is the case of transformers, through the self-attention layers, implementing a flexible version of sparsity (that is, selecting which input tokens interact in the MLP layer). For more classical feedforward multilayer networks, ℓ_2 optimization can also be used when the sparsity of the target function is known and is reflected in the architecture of the network, as it is the case for CNNs.

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

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. In the meantime, other architectures, especially transformers, also show amazing performance. Is there a common principle at the core of these successful neural networks? In the following, I describe a framework built around around a specific principle that, I conjecture, must underlie most of deep learning. In this preliminary note, the main text outlines the main argument, while the appendices provide random details.

2 Approximation theory: the curse of dimensionality can be avoided for target functions that are *compositionally sparse*

The relatively old work of Mhaskar and Poggio [1] starts from 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 a measure of smoothness of the function with $s \geq 1$ (since otherwise Kolmogorov theorem will apply, see Appendix). The curse can be avoided by shallow or deep networks if s is large and in particular if s grows with d. The curse can also be avoided by deep networks, but not by shallow ones, if the function is *compositionally sparse*, that is if the function graph is such that each constituent function has low dimensionality[1]. The theorem is

Theorem 1. *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 $W_{m_v}^{d_v}$. 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*

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

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

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

In the meantime the assumption of sparse target functions has appeared often in the most recent approximation literature (see [2, 3, 4, 5, 6]).

It is important to observe that the curse does not apply only to real-valued continuous functions but also to *binary approximations of continuous functions*; it also applies to all Lipschitz functions that approximate a continuous function. A specific constructions of relevant Boolean functions is discussed in the Appendix (see 6.2.3 and [7]), leading to the following

Theorem 2. *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 $f : \mathbb{B}, -\mathbb{B}^n \mapsto \mathbb{R}$ be a compositional $\mathcal{G}$-function, where each of the constituent functions is a Boolean function in d_v Boolean variables. Consider shallow and deep networks with possibly non-smooth activation functions. 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 n, since $max_v d_v$ effectively sets the upper bound on the number of required parameters.*

Remarks

- All functions have a compositional representation but *in general a function admits more than one compositional graph representation*.
- Because of theorem 2 (see also [7]) all compositionally sparse real-valued functions can be approximated by a Boolean compositionally sparse function which is computable.

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

2.1 All physically computable functions must be sparse

An obvious question is how "big" is the class of such compositionally sparse functions wrt the class of all continuous functions. Perhaps unsurprisingly, the answer seems to be that *in practice* all functions – with at least a continuous derivative – must be (compositionally) sparse, that is sparse or sparse in their constituent functions (all functions are trivially compositional[2]). Let us recall the classical definition of a computable function:

Definition 1. *$f : N^d \to N$ is computable if there is an effective procedure or algorithm that correctly calculates f.*

Here, computable usually doesn't assume polynomial time/space complexity, but *efficiently computable* means in polynomial time/space.

The definition above is enough for our purposes. For more clarity we may also consider an equivalent definition of *efficient computability* (work with S. Lloyd, [8]):

Definition 2. *A function $f : N^d \to N$ is physically computable (or efficiently computable) if an approximation of it – within an arbitrary error ϵ – can be computed/stored in time/memory that is at most polynomial in d.*

Then the "PL thesis" is:

Thesis 1. *All physically computable functions are compositionally sparse, that is they must have a compositional representation, in general not unique, in which all constituent functions have "small" $\frac{d}{s}$.*

Informal proofs in support of this thesis are as follows:

- (for the real valued case) An arbitrary function $f : \mathcal{R}^{1000} \to \mathcal{N}$ may require 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}.
- (for the case of computable Boolean functions) Assume that the function is computable by a Turing machine. This means that the function can be represented by the composition 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. If t is bounded we have a finite state machine, otherwise a Turing machine. Each layer in a deep network corresponds to one time step in a Turing machine. In a sense, this is sequential compositionality.

Kolmogorov and Arnold [9, 10, 11] solution of Hilbert's thirteenth problem shows that every continuous functions can be represented exactly as a compositions of functions of one variable. However, in this case the constituent functions are very non-smooth, that is $s < 1$. Appendix 6.1 has more information.

Remark Yes, computable usually doesn't assume polynomial time/space, but efficiently computable means polynomial time. So, if a function is efficiently computable with Turing machines, then it is compositionally sparse.

3 Learning theory: compositional sparsity underlies better generalization

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 [12]

Theorem 3. (*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.*

[2]Since every function can be composed with the identity function

In complete analogy with the approximation result the key property here is locality of the convolution kernel ($k << n$) and *not weight sharing*. In a somewhat similar way, lower rank of the weight matrices (dense or convolutional) improves generalization – assuming that the empirical error does not change. Notice that an equivalent result for underparametrized networks follows directly from considerations of VC dimension (see Appendix, section 8 in [7], 10.1214/19-AOS1912). The novelty here is to show that sparsity can lead to generalization in the overparametrized case, when weight decay, that is regularization, is present[3].

4 Open questions in optimization

4.1 The sparse graph is known: CNNs

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, and then select the one with lowest complexity [13]. Recent work [12] 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[4]. 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 6.3 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 sufficient 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. ℓ_1 minimization however leads to pruned networks wrt ℓ_2 optimized networks.*

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

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

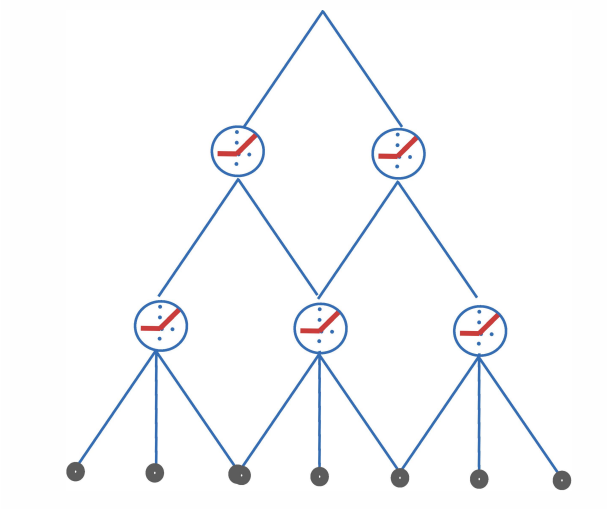

Figure 1: *The network here – similar to a CNN – reflects in a "hardwired" way the sparse compositional function graph of the target function. The function graph is supposed to be known.*

4.2 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.

4.2.1 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.

4.2.2 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.

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 6.4 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.

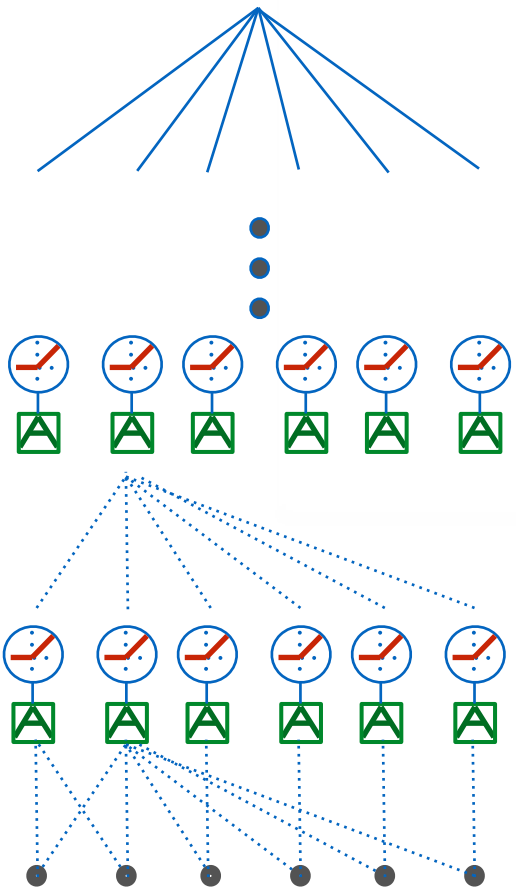

Figure 2: *Here attention (followed by a one-layer RELU network) selects for each input token its connections to other tokens, effectively instantiating a network that reflects a compositionally sparse function graph. Each input here is a token, that is a vector, such as a patch of an image. The "A"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

5 Summary

I propose a theoretical framework that aims to explain why deep networks work and what are the properties of different architectures. The framework describes a few conjectures and partial results that require further empirical and theoretical analysis.

The *key assumption* is about the world, that is about the tasks that networks could try to learn. The assumption is that all learnable functions must have a representation with the property of *compositional sparsity*, that is they can be represented as compositional functions with a function graph comprising constituent functions with a bounded – and "small" – dimensionality. The assumption is justified by a conjecture/theorem stating that *all physically computable functions*, which are of course compositional, are also compositionally sparse.

The next main result is the following statement in *approximation* theory: *For functions that admit a compositionally sparse directed acyclic graph (DAG), approximation by appropriate multilayer networks is possible without incurring in the curse of dimensionality.*

Sparsity is also a key property for *generalization*: 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. An interesting case of this sparsity is represented by convolutional layers with a local kernel. 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. In a similar way, lower rank of the weight matrices improves generalization assuming that the empirical error does not change.

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 part of our framework is about the case of unknown function graph and sparsity constraints in optimization. In particular, two situations should be considered. The main one is focused on transformers, the second on dense networks under sparsity constraints.

   For transformers the conjecture is that the self-attention layer finds the sparse graph structure of the underlying regression function. I will show that the stages of self-attention and MLP with normalization and residual connections can be seen as a sparsification step followed by a one-layer MLP.

   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 works.

In summary, the claim is that compositional sparsity of the underlying regression function is the key assumption/principle in machine learning. Sparsity of the target function then naturally leads to sparsity-biased optimization techniques. This is the case of transformers. ℓ_2 optimization, however, can be used when the sparse graph of the target function is known and is reflected in the architecture of the approximating/learning network (eg CNNs).

Acknowledgments I thanks Tomer Gallant, Seth Lloyd, Akshay Rangamani, Shimon Ullman, Yaim Cooper, Eran Malach and Santosh Vempala 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), and AFSOR-THRL (FA8650-05-C-7262).

References

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

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

[3] 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.

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

[5] Markus Bachmayr, Anthony Nouy, and Reinhold Schneider.

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

[7] 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.

[8] Seth Lloyd and Tomaso Poggio. Are all physically computable functions compositionally sparse? *CBMM Memo*, 2022.

[9] 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.

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

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

[12] 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.

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

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

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

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

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

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

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

[20] 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.

[21] 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.

[22] 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.

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

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

[25] M. Xu, T. Poggio, and T. Galanti. Complexity bounds for sparse networks. *CBMM memo 1XX*, 2022.

6 Appendices

6.1 Kolmogorov's theorem[9]

Theorem 4. (*Kolmogorov, 1957*). *There exist fixed increasing continuous functions $h_{pq}(x)$, on $I = [0, 1]$ so that each continuous function f on I^n can be written in the form*

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

where g_q are properly chosen continuous functions of one variable.

This result asserts that every multivariate continuous function can b 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 [14] 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 [15] with the following theorem:

Theorem 5. (*Vitushkin 1954*). *There are $r(r = 1, 2, \ldots)$ times continuously differentiable functions of $n \geq 2$ variables, not representable by superposition of r times continuously differentiable functions of less than n variables; there are r times continuously differentiable functions of two variables that are not representable by sums and continuously differentiable functions of one variable.*

6.2 Computable functions and Boolean representations

6.2.1 *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.

6.2.2 *Spline approximations, Boolean functions and tensors*

- Consider the case of a multivariate function $f : [0, 1]^d \to$. 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 $f : \{0, 1\}^N \to$.
- As shown later, every function f can be approximated by an epsilon-close binary function f_B. Binarization of $f :^n \to$ is done by using k partitions for each variable x_i and indicator functions. Thus $f \mapsto f_B : \{0, 1\}^{kn} \to$ 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 [16]). This work, as well more classical papers [17], 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 [17]). 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.

6.2.3 *On multivariate function approximation*

Consider a multivariate function $f : [0,1]^d \rightarrow$ 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 $\phi_{i_\mu} : [0,1] \rightarrow, 1 \leq i_\mu \leq n_\mu, 1 \leq \mu \leq d$
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. 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.

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 ^{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 $(wx+b)^N$.

   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 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

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

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 ^{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 $c_i, b_i, \lambda_i \in$ 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 [18]) 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 6.2.3 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 $f \in^2 [a,b]$ 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.

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 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

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

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[5].

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.

6.3 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 [19]. 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 [20]) 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[6].

This is similar to results: enforcing sparsity during training leads to structures characterized by locality. [21] 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, [22] 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.

6.4 Transformers

6.4.1 K, Q, V

$X \in \mathcal{R}^{T,d_{in}}$; $Q = XW_Q$ 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 $W_V \in \mathcal{R}^{d_{in},d_{out}}$

Lemma 3. *The matrix $XW_QW_K^TX^T \in \mathcal{R}^{T,T}$ can be a RIP matrix with appropriate choices of W_K, W_Q.*

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

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

[5]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.

[6]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 remarks "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..." are consistent with our hypothesis of compositional sparsity of the underlying task, in this case a visual task.

.

This implies that the sparsity function and the nonlinear association are intertwined – together one stage in a multistage architecture. This may be ideal to represent compositional sparsity. There is however a formulation with similar empirical performance (see PALM paper) which can be written as

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

.

7 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, differentiable 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.

8 Background: Sparse solutions of $Ax = y$

8.0.1 Restricted Isometry

In linear algebra, the restricted isometry property (RIP) characterizes matrices which are nearly orthonormal, at least when operating on sparse vectors. The concept, which is a formalization of the old concept of noiselike" signals used in the context of holographic memories, was introduced by Emmanuel Candès and Terence Tao. It has been shown that with exponentially high probability, random Gaussian, Bernoulli, and partial Fourier matrices satisfy the RIP property with a number of measurements nearly linear in the sparsity level.

Let A be an $m \times p$ matrix and let $1 \leq s \leq p$ be an integer. Suppose that there exists a constant δ_s such that, for every $m \times s$ submatrix A_s of A and for every s-dimensional vector y,

$$(1 - \delta_s)||y||_2^2 \leq ||A_s y||_2^2 \leq (1 + \delta_s)||y||_2^2. \tag{10}$$

Then, the matrix A is said to satisfy the s-restricted isometry property with restricted isometry constant δ_s. This condition is equivalent to the statement that for every $m \times s$ submatrix A_s of A we have

$$||(A_s)^T A_s - I_{s\times s}||_2 \leq \delta_s. \tag{11}$$

where $I_{s\times s}$ is the identity matrix and $||\cdot||_2$ is the operator norm. Finally this is equivalent to stating that all eigenvalues of are in the interval $[1 - \delta_s, 1 + \delta_s]$.

8.1 Iterative thresholding

The iterative hard thresholding algorithm is an iterative algorithm to solve the rectangular system $Az = y$, knowing that the solution is s-sparse. We shall solve the square system $A^T Az = A^T y$ instead, which can be interpreted as the fixed-point equation

$$z = (\mathbf{I}_d - A^T)z + A^T y. \tag{12}$$

Classical iterative methods suggest the fixed-point iteration $x_{n+1} = (I_d - A^T A)x_n + A^T y$. Since we target s-sparse vectors, we only keep the $(I_d - A^T A)x_n + A^T y = x_n + A^T(y - Ax_n)$ at each iteration. The resulting algorithm reads as follows:

$$x^{n+1} = H_D(x^n + A^T(y - Ax^n)) \tag{13}$$

9 Generalization bounds for Sparse Networks

9.1 Convolutional networks with local kernel

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 section **??** 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. A naive application of (**??**) to convolutional networks give a large bound, where the Frobenius norm of the Toeplitz matrix is equivalent to norm of the kernel multiplied by the number of patches.

In this section we provide an informal analysis of the Rademacher complexity, showing that it can be reduced by exploiting maonly the first one of the two properties of convolutional layers: (a) the locality of the convolutional kernels 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 different 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 $(0, \dots, 0, -W^l-, 0 \dots, 0)$, 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 6. *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,*

$$\mathcal{R}_X(\mathcal{H}) \leq \frac{2^L \rho \sqrt{\sum_{i=1}^m \|x_i\|^2}}{m} \tag{14}$$

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

$$\begin{aligned}
\mathcal{R}_X(\mathcal{H}) &= \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}} |\frac{1}{m} \sum_{i=1}^m \epsilon_i \cdot h(x_i)| \\
&= \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}} \frac{1}{m} |\sum_{i=1}^m \epsilon_i \cdot W^L \cdot \sigma(v_1(x), v_2(x))| \\
&= \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}} \frac{1}{m} \sqrt{|\sum_{i=1}^m \epsilon_i \cdot W^L \cdot \sigma(v_1(x), v_2(x))|^2}
\end{aligned} \tag{15}$$

Next, by the proof of Lem. 1 in [23], we obtain that

$$\begin{aligned}
\mathcal{R}_X(\mathcal{H}) &\leq 2\mathbb{E}_\epsilon \sup_{h \in \mathcal{H}} \frac{1}{m} \sqrt{\|W^L\|^2 \cdot \|\sum_{i=1}^m \epsilon_i (v_1(x), v_2(x))\|^2} \\
&= \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}} \frac{1}{m} \sqrt{\|W^L\|^2 \cdot \sum_{j=1}^2 \|\sum_{i=1}^m \epsilon_i v_j(x_i)\|^2}
\end{aligned} \tag{16}$$

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

$$\begin{aligned}
\mathcal{R}_X(\mathcal{H}) &\leq 2^{L-1} \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}} \frac{1}{m} \sqrt{\prod_{l=1}^L \|W^l\|^2 \cdot \sum_{j=1}^d \|\sum_{i=1}^m \epsilon_i x_{ij}\|^2} \\
&= 2^{L-1} \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}} \frac{1}{m} \sqrt{\prod_{l=1}^L \|W^l\|^2 \cdot \|\sum_{i=1}^m \epsilon_i x_i\|^2} \\
&\leq \frac{2^{L-1} \rho \mathbb{E}_\epsilon \|\sum_{i=1}^m \epsilon_i x_i\|}{m} \\
&\leq \frac{2^{L-1} \rho \sqrt{\sum_{i=1}^m \|x_i\|^2}}{m}
\end{aligned} \tag{17}$$

where the factor 2^{L-1} is obtained because the last layer is linear (see [24]). We note that a better bound can achieved when using the reduction introduced in [23] 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 [24, 23]), that expresses the Rademacher complexities at the outputs of one layer in terms of the outputs at the previous layers.

Lemma 4. *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 7. (*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*

$$\hat{\mathcal{R}_m(\mathcal{H}_d)} \leq (\sqrt{2\log(2)L} + 1) \prod_{j=1}^{L} M_j \|x\| \tag{18}$$

Proof sketch. Each $h_k^d \in \mathcal{H}_\Gamma$ ($k = 1, \cdots, Q$ 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

$$\hat{\mathcal{R}}_m\left(\mathcal{H}_d\right) = \mathbb{E}_\epsilon \sup_{h_i \in \mathcal{H}_d} \frac{1}{m} \sum_{i=1}^{m} \epsilon_i h_i = \mathbb{E}_\epsilon \sup_{h_i \in \mathcal{H}_{d-1} w: \|w\| \leq M} \frac{1}{m} \sum_{i=1}^{m} \epsilon_i w \cdot (\sum_k v_k), \tag{19}$$

can be upper bounded as follows

$$\hat{\mathcal{R}}_m\left(\mathcal{H}_d\right) \leq 2M_d \mathbb{E}_\epsilon \sup_{h \in \mathcal{H}_d} \|\frac{1}{m} \sum_{i=1}^{m} \epsilon_i (\sum_k (v_k)_i)\| \leq \frac{1}{m} \sqrt{(w \cdot (\sum_k v_k)^2} = \frac{1}{m} \sqrt{(w \cdot (\sum_k v_k)^2} = 2M_d \hat{\mathcal{R}}_m\left(\mathcal{H}_{d-1}\right), \tag{20}$$

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

$$\mathcal{R}_m(\mathcal{H}_L) \leq 2^{L-1} \hat{M_L} \cdot M_{L-1} \cdots M_1 \|x\|, \tag{21}$$

where the factor 2^{L-1} is obtained because the last layer is linear (see [24]). To this result we can further apply the reduction used by [23] to finally obtain the result. □

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.

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. In fact a proof of the following conjecture will be provided in [25]

Conjecture 3. *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 $\sqrt{\frac{1}{d}}$.

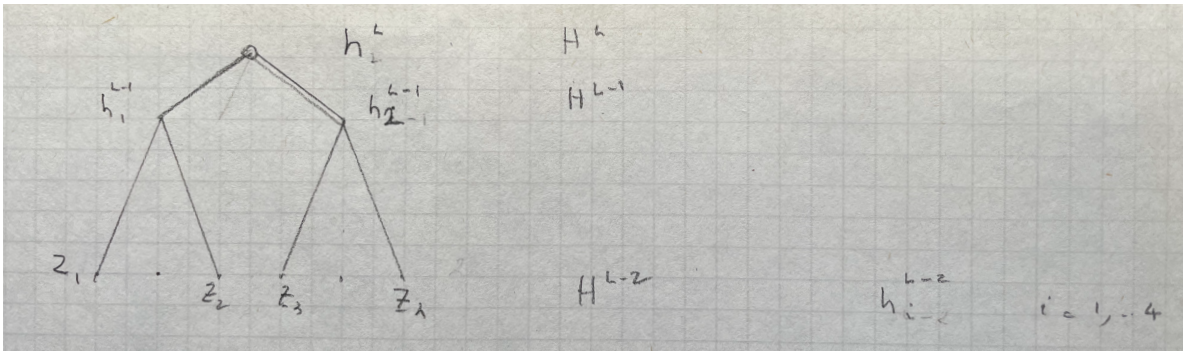

Figure 3: A binary CNN.

9.2 Low-rank dense weight matrices

An obvious question is whether a deep RELU network that fits the data generalizes better than another one if the rank of its weight matrices is lower. A forthcoming paper [25] proves that

Theorem 8. (*informal*) *Let f_W be a neural network, trained with SGD under square loss in the presence of WN. Assume that the weight matrix W_k of dimensionality n, n has rank $r < n$. Then its contribution to the Rademacher complexity of the network will be $\|W\|\sqrt{\frac{r}{n}}$ instead of just $\|W\|$ in typical bounds.*

Sketch proof We start assuming W^k of norm 1 that is $\|W^k\| = \|V^k\| = 1$ and

$$\mathcal{R}_k = \mathbb{E}_\epsilon \sup_{W^k} \frac{1}{m} \sqrt{\| \sum_{i=1}^{m} \epsilon_i W^k \sigma(x)\|^2}. \tag{22}$$

then assume that W^k of dimensionality $n \times n$ becomes at convergence of rank r whereas the input activities $h_{k-1} = z$ has norm $\|z\| = 1$ and dimensionality n. We also assume that in expectation z have equal components in terms of the standard basis in $\mathcal{R}^n$. We rewrite

$$\mathcal{R}_k = \mathbb{E}_\epsilon \sup_{\Sigma} \frac{1}{m} \sqrt{\| \sum_{i=1}^{m} \epsilon_i U \Sigma V^T z\|^2}. \tag{23}$$

Since V^T is defined to be orthonormal $z' = V^T z$ has the same norm of z. Thus

$$\begin{aligned} \mathcal{R}_k = \mathbb{E}_\epsilon \sup_{\Sigma} \frac{1}{m} \sqrt{\| \sum_{i=1}^{m} \epsilon_i U \Sigma z'\|^2} &= \mathbb{E}_\epsilon \sup_{\Sigma} \frac{1}{m} \sqrt{\| \sum_{i=1}^{m} \epsilon_i U z^{(r)}\|^2} \\ &= \mathbb{E}_\epsilon \frac{1}{m} \sqrt{\|U\|^2 \|z^{(r)}\|^2 \| \sum_{i=1}^{m} \epsilon_i\|^2} \\ &= \sqrt{\frac{2}{m}} \|z^{(r)}\| = \sqrt{\frac{2}{m}} \sqrt{\frac{r}{n}}. \end{aligned} \tag{24}$$

where $z^{(r)}$ consists of the vector with r components of z'.