CENTER FOR
Brains
Minds+
Machines

CBMM Memo No. 138 **October 11, 2022**

It is 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 thesis is that compositional sparsity of the underlying regression function – thus an hypothesis about the task to be learned – is the key principle of machine learning. I discuss 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 be rephrased as the question of why CNNs work so well. In the meantime other architectures, especially transformers, also show amazing performance. Is there a common explanatory principle? In the following, I describe a framework built around around a specific principle that I conjecture to be at the core of all of deep learning. In this preliminary brief note, the main text outlines the main argument, while the appendices provide more details – either background or proofs.

2 The curse of dimensionality can be avoided for target functions that are either very smooth or compositionally sparse

The relatively old work of Maskhar 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 is a a measure of smoothness of the function. 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

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

3 Physically computable functions must be sparse

An obvious question at this point 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 must be sparse compositional. Let us recall the classical definition of a computable function:

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

The definition above may be enough for our purposes – if "effective" is appropriately defined. For more clarity we may also consider an explicitly more restricted definition of computability (work with S. Lloyd):

Definition 2 *A function $f : N^d \to N$ is physically computable if it can be computed/stored in time/memory that is at most polynomial in d.*

Then the "thesis" is:

Thesis 1 *All efficiently computable functions are compositionally sparse, that is their constituent functions have "small" $\frac{d}{s}$.*

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

The argument for this "thesis" is simple: a function $f : \mathcal{N}^{1000} \to \mathcal{N}$ requires a memory of $> 10^{1000}$ bits, larger than the number of protons in the Universe, 10^{80}.

Kolmogorov and Arnold [3, 4, 5] solution of Hilbert's thirteenth problem shows that every continuous functions can be represented exactly as a compositions of functions of one variable. However, the constituent functions are very non-smooth. Appendix 7.1 has more information.

4 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, while selecting the one with lowest complexity (Malach and Poggio, in preparation). Recent work [6] has provided theoretical and empirical evidence that this can be accomplished by SGD provided

1. the sparse function graph of the underlying regression function is assumed to be known and takes the form, for instance, of a convolutional network;
2. the network is overparametrized allowing zero empirical loss;
3. the loss function is the regularized (e.g. weight decay) square loss or an exponential loss.

Thus 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 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[2]. 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 7.2 is about pruning and related issues. Empirically it seems that the graph of the target function needs to be known approximately: it is sufficient that the sparse network contains as a subgraph the target function graph. A proof would be interesting.

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.*

5 The sparse graph is unknown: transformers

The second part of our framework 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.

5.1 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 works.

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

5.2 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)? The tentative answer proposed here is that self-attention for each token selects the relevant other tokens in the sequence. 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 7.3 the normalized softmax $H_D(x) = xH(x)$ (with H being a threshold on x) induce sparsity in the selection of connections, preferring only a small number of very similar token (where the similarity is adjusted 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 first node of a convolutional network.

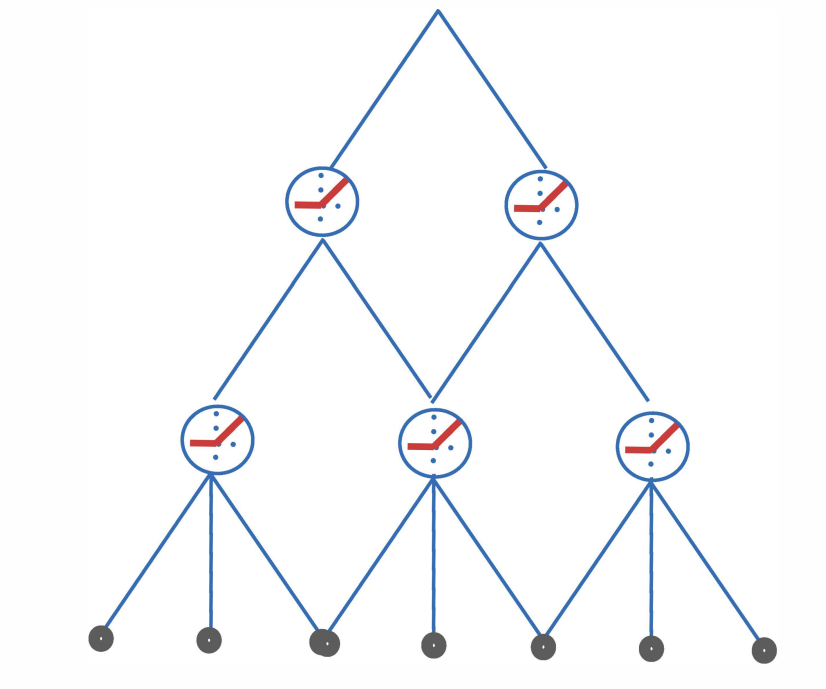

Figure 1: The network here – similar to a CNN – reflects the sparse compositional function graph of the target function.

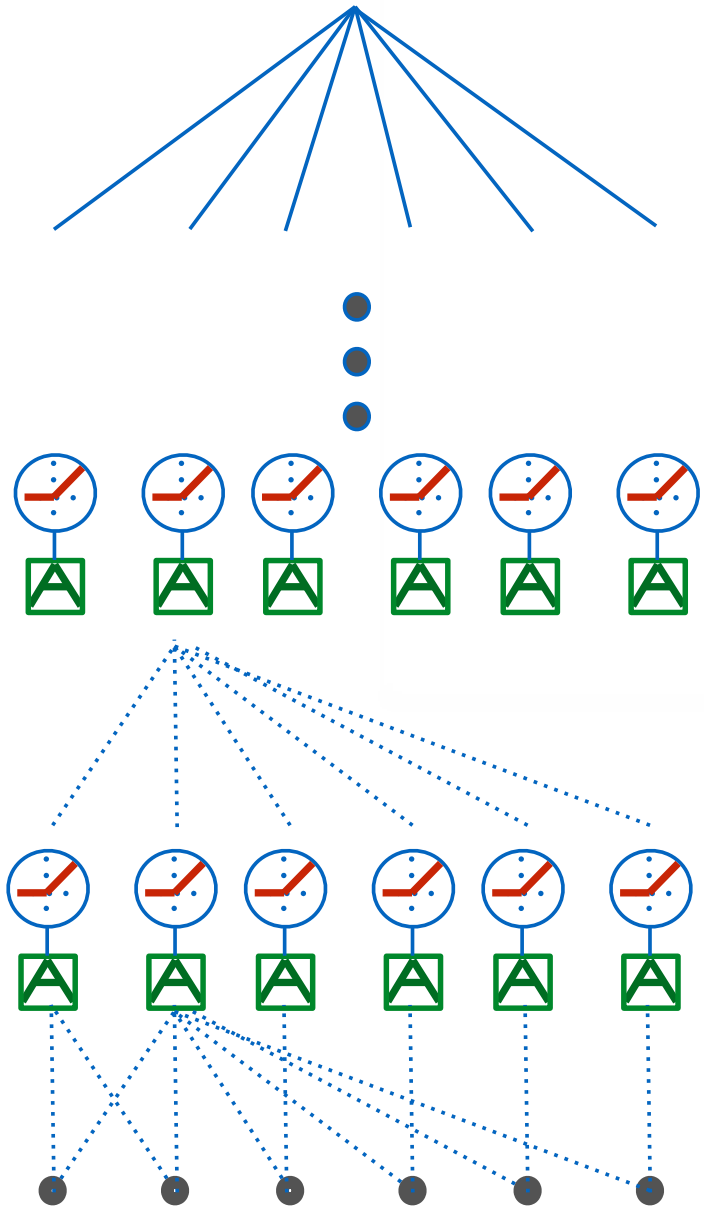

Figure 2: Here attention (followed by a one-layer RELU network) selects for each input token its connections, effectively instantiating a network that tries to reflect a compositional function graph – or an approximation to it. Each input here is a token, that is a vector, such as a patch of an image. The "A" box is the self-attention algorithm; the RELU circle represent a one-layer NN.

6 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 much 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 efficiently computable functions* are compositionally sparse (this statement is also independently supported by Kolmogorov solution of Hilbert's 13th problem.). The next main result is the following theorem: *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.* 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 sparsity of the underlying regression function is the key assumption/principle in machine learning. Sparsity then naturally leads to sparsity-biased optimization techniques. This is the case of transformers. ℓ_2 optimization however can be used when the sparsity of the target function is known and is embedded in the architecture of the network (eg CNNs).

Acknowledgments 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] 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.

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

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

[6] 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.

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

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

7 Appendices

7.1 Kolmogorov's theorem[3]

Theorem 2 (*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 very limited. The problem lies in the inner functions of Kolmogorov's formula: although they are continuous, theorems of Vitushkin and Henkin [7] prove that they must be highly nonsmooth. 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 [8] with the following theorem:

Theorem 3 (*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.*

7.2 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 Neyshabur (2020). 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 https://doi.org/10.48550/arxiv.2104.13343) 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[3].

This is similar to results: enforcing sparsity during training leads to structures characterized by locality. d'Ascoli et al. (2019) 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, Tolstikhin et al. (2021) shows 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.

7.3 Transformers

7.3.1 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 $W_V \in \mathcal{R}^{d_{in},d_{out}}$

[3]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.

Lemma 1 *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

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

.

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

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

.

8 Training transformers

In this paper, I describe results and conjectures on trained transformers. I leave open all the theoretical questions about characterizing the training of transformers and the convergence to matrices Q, K, V that guarantee sparsity.

8.1 Transformers as associative memories

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.