CBMM Memo No. 145 **February 8, 2024**

Compositional Sparsity of Learnable Functions

Tomaso Poggio and Maia Fraser

Abstract

Neural networks have demonstrated impressive success in various domains, raising the question of what fundamental principles underlie the effectiveness of the best AI systems and quite possibly of human intelligence. This perspective argues that compositional sparsity, or the property that a compositional function have "few" constituent functions, each depending on only a small subset of inputs, is a key principle underlying successful learning architectures. Surprisingly, all functions that are efficiently Turing computable have a compositional sparse representation. Furthermore, deep networks that are also sparse can exploit this general property to avoid the “curse of dimensionality". This framework suggests interesting implications about the role that machine learning may play in mathematics.

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

BULLETIN (New Series) OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 00, Number 0, Pages 000–000
S 0273-0979(XX)0000-0

COMPOSITIONAL SPARSITY OF LEARNABLE FUNCTIONS

TOMASO POGGIO AND MAIA FRASER

Abstract. Neural networks have demonstrated impressive success in various domains, raising the question of what fundamental principles underlie the effectiveness of the best AI systems and quite possibly of human intelligence. This perspective argues that compositional sparsity, or the property that a compositional function have "few" constituent functions, each depending on only a small subset of inputs, is a key principle underlying successful learning architectures. Surprisingly, all functions that are efficiently Turing computable have a compositional sparse representation. Furthermore, deep networks that are also sparse can exploit this general property to avoid the “curse of dimensionality". This framework suggests interesting implications about the role that machine learning may play in mathematics.

The impressive success of machine learning across various domains — including "hard" problems such as protein folding, playing Go, driving cars, generating near-human text — has led to a growing interest in understanding fundamental principles underlying the effectiveness of these methods, in particular, neural networks which are crucial in all these systems. At the same time, as we consider the question "will machines change mathematics?", it is natural to seek intuition regarding the prospect of machines succeeding at core challenges of research mathematics, namely, theorem proving, as well as proposing conjectures. The present contribution offers a mathematical perspective on the success of neural networks, i.e. foundations of so-called deep learning, in terms of the notion of *compositional sparsity*. Roughly speaking, a compositional function is sparse if it is the composition of "few" constituent functions, each depending on only a small subset of inputs (we make this more precise in Definition 2.2). This intuition, as we will explain, is particularly relevant to the question of theorem proving, and links closely to Gowers’ analysis of "how it can be that humans find proofs" [10].

Before turning to technical definitions, we first briefly discuss some history, and the meaning of the word "learning" as it appears in machine learning, deep learning, and learning theory etc. The advent of modern logic and mathematics formalization more than a century ago was closely tied from the start to interest in automated theorem proving. Early research in this area, e.g. by Russell, Herbrand, Gödel, Church, Turing, also led to the development of theoretical computer science as a discipline, and central figures in the new field of artificial intelligence — Wiener, Turing, Pitts — were mathematicians, working in logic. Wiener, a polymath in his time, distinguished between two types of machine: those that interact with the user or environment according to fixed rules, and those that do so according to evolving rules which depend on previous experience [34, 35]. He referred to the

2020 *Mathematics Subject Classification.* Primary .

latter as *learning* systems[1]. This characterization was an attempt to formalize the phenomenon of learning observed in biological systems (including human beings). Artificial intelligence, a looser term[2] includes machines both with and without learning capabilities, and during the next fifty years, until the 1990s, the most visible advances in AI were non-learning (rule-based) systems. These have come to be known as GOFAI for "good old-fashioned AI", or "expert systems"; they focused on replicating rules and reasoning that human experts consciously use. Parallel developments in learning theory and learning systems continued throughout this time, particularly for systems modeled after biological neural networks. By the 1980s the field of machine learning had become increasingly active and, in the last 10 years especially, it has had a wide influence on computer science more broadly, changing the basic paradigm from "programming" to "training". Simultaneously, breakthroughs in computing power and accompanying algorithmic innovations have led to widespread practical success of neural networks. Previously, only shallow networks with two layers had been manageable, thus greatly limiting their effectiveness; the new paradigm, networks with three or more layers, became known as deep learning. With this success came broad media attention for AI in general, widespread use by the public (e.g. in social media, search engines, and voice recognition), uptake by business and scientific professions - and also massive investment and profits in the private sector developing these tools. In the last decade alone such developments have far eclipsed the earlier achievements of GOFAI so that "AI" in mainstream media nowadays usually denotes machine learning, or even more specifically deep learning.

In fact, combining learning modules, and for example melding rule-based and learning systems, has always been a strategy of interest in AI and will surely be further developed in future (cf. Gowers' announcement [9] where he mentions "plenty of potential for combining machine learning with GOFAI ideas"). In recent years, modern proof assistants such as Lean are increasingly used by mathematicians, and new areas of math like homotopy type theory and univalent foundations themselves heavily incorporate (and rely on) computer formalization. Lean and other proof assistants are so far not learning systems, but rather hard-coded software. As we focus in this paper on the abilities of deep learning, and the potential use of learning systems for proving or proposing mathematical statements of "interest" to mathematicians [10], the reader is invited to keep a high-level viewpoint and to imagine a situation where logical steps and propositions could be represented in a system such as Lean, and deep learning, possibly in tandem with other refinements,

[1]The latter can use fixed rules as well as malleable ones, the former only used fixed rules.

[2]Wiener's work on intelligence had focused strongly on the role of learning and (analog) feedback for intelligent behaviour and he coined the name Cybernetics for that field [34]. The term Artificial Intelligence was coined several years later at the Dartmouth Conference [19] by others, including mathematicians McCarthy and Shannon, who intended to use fixed-rule as well as learning approaches: "every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves." They expected to achieve this within the summer of 1956; in fact, we are only now getting close to some of these goals.

used to produce tentative proofs or conjectures[3]. To give a sense of the abilities of deep learning, we will make use of a *learning theory* viewpoint.

The paper is structured as follows. In Section 1, we give a brief introduction to learning theory, the mathematical study of learning systems. Here we identify three key considerations — approximation, optimization, and generalization — and establish terminology such as *target class* and *approximator class*. We then define *neural networks*, and highlight a central challenge in designing learning systems: close approximation and efficient optimization are inherently at odds. In Section 2, we explain how deep neural networks can resolve this tension for certain classes of target function, namely those that are *compositionally sparse*. We define and discuss compositional sparsity, explaining its relationship to efficient computability. In Section 3, we point out a further advantage of deep neural networks: if one restricts, as in the previous section, to deep networks with architecture matching the compositional structure of a given target class of compositionally sparse functions, this leads also to order of magnitude improvement in generalization, the key criterion by which learning systems are judged. Next, in Section 4, we discuss two types of neural network that exploit this phenomenon: CNNs and Transformers (the former commonly used for computer vision tasks, the latter at the core of chat bots such as ChatGPT). Finally, in Section 5, we return to discuss the prospect that learning systems might increasingly be able to prove theorems or make conjectures (see [39]). In particular, we note that the learning theoretic insights sketched above give a partial answer to questions posed by Gowers [10] concerning the class of statements mathematicians are interested in proving/refuting. This in turn sheds light on the potential usefulness of learning machines.

1. A Perspective on the Foundations of Deep Learning

Essentially, at the most basic level, a neural network is a parametric representation that is used to approximate a function. Usually this function is implicitly given by a large "training" set of input-output data. In the following, we refer to "approximators" — the networks — and to "target functions" — the maps to be learned.

1.1. **The framework.**

We sketch very briefly some basic ideas of learning theory. A supervised learning problem is defined by an unknown probability measure μ on the product space $X \times Y$; the training data $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^m$ are i.i.d. samples from μ. For simplicity, we will assume $Y = [0, 1]$.

The measure μ defines a function

[3]We are not advocating a specific design for such machines, but instead broadly considering some relevant design considerations. A useful image to have in mind could be a machine like a chat bot that proposes next steps in a proof. It would do so having already been given either the final goal statement or some other description of the context, depending on whether we seek a proof/refutation or a conjecture. This setup is akin to specifying the kind of text or response we'd like a chat bot to produce; however, unlike current chat bots we could assume that this "math bot" starts with a knowledge base of known true statements. We might want it to operate at lowest granularity, proposing a logical sequence of subsequent statements (where typically each one can be verified, for example by Lean, to follow logically from previous ones), or very high granularity, proposing essentially just a single (difficult) conjecture, or anywhere in between, in which case it would be proposing a sequence of intermediate statements that are predicted to plausibly lead to the desired final statement, as potential roadmap to assist theorem proving by a different machine, or by human mathematicians.

$$f_\mu : X \to Y \tag{1.1}$$

satisfying $f_\mu(x) = \int y d\mu_x$, where μ_x is the conditional measure on $\{x\} \times Y$.

From this construction f_μ can be viewed as the true input-output function reflecting the environment which produces the data[4]. The goal of supervised learning is to "closely" approximate this function. To do so, a parametric approximator f is used, and closeness is measured by the (*generalization*) *error* of f (also called the *true risk* of f) which may be defined by

$$\int_X (f - f_\mu)^2 d\mu_X \tag{1.2}$$

where μ_X is the marginal measure[5] on X. This is the expectation of the squared loss and it conveniently corresponds to the L^2 distance between f and f_μ, but $(f-f_\mu)^2$ is sometimes substituted by $\ell(f_\mu(x), f(x))$ for other pointwise loss functions $\ell : Y \times Y \to [0, \infty)$.

To achieve the goal of supervised learning, i.e. "find" f minimizing generalization error for an unknown f_μ, it is necessary to have a space $\mathbb{F}$ of parametric functions where search is possible, yet which is rich enough to approximate a broad range of possible target functions. We will refer to these as *approximator class* and *target function class* respectively. The *design* of a supervised learning algorithm thus typically begins with the two steps: (1) choose $\mathbb{F}$, (2) specify how to search within $\mathbb{F}$. Step (1) corresponds to approximation, step (2) to optimization. Once we've chosen these, we've designed a learning algorithm. Learning theory then mathematically analyzes the algorithm in terms of how well the choices made in (1) and (2) conspire to produce, for each function f_μ in the target class, an approximator f with low generalization error. Note: the search within $\mathbb{F}$ - and thus the approximator f - typically depend on $\mathcal{S}$, so the generalization error is a random variable and its analysis is subtle[6]; we refer to the analysis of generalization as step (3). Altogether, learning theory thus draws on notions and results from several areas of mathematics, including probability theory, functional analysis, combinatorics, approximation theory and optimization.

In this paper, we focus on the space of deep neural networks with L layers as a choice for $\mathbb{F}$ in step (1). Below we will give the definition of this "model", and

[4]For example, the input-output function of interest might take as input an xray image of some kind and output a binary value y which is 1 if and only if a tumor appears in the image. Here X would be the set of all possible images of the specified kind, and μ would be the distribution of pairs (x, y) as they naturally occur. The training set $\mathcal{S}$ is sampled from μ. In practice, if X consists of all xrays produced in participating hospitals, this might mean sampling images x_i from that database uniformly (which amounts to the marginal μ_X being uniform), then hiring humans to annotate each x_i with the appropriate value 1 or 0 for y_i (this uses humans to reveal the conditional μ_{x_i}; in this example it is concentrated entirely at $y = 1$ or $y = 0$ and humans know which)

[5]This is the pushforward of the measure μ via the projection π_X to X. More technically, one starts with $X \times Y$ a product measurable space determined by the measurable spaces X and Y; then for any measurable set $A \subset X$, $\mu_X(A) := \mu(A \times Y)$.

[6]One common way this is done - the *PAC learning* approach [29] - is to ask if generalization error is below ϵ with high probability, in a way that depends "polynomially" on the size m of $\mathcal{S}$; here, the target function class is called the *concept class*, and the approximator class $\mathbb{F}$ is called the *hypothesis class*. A related approach is found in VC theory, an earlier broad framework for studying learning [30, 31, 32].

later we consider using instead a subclass $\widehat{\mathbb{F}}$. The rest of the paper then justifies this choice, through the 3-step lens of learning theory.

Once $\mathbb{F}$ is chosen in step (1), then as step (2), starting from the data $\mathcal{S}$, one may minimize the *empirical error* (also called *empirical risk*) $\frac{1}{m}\sum_{i=1}^{m}(f(x_i)-y_i)^2$ over $f \in \mathbb{F}$ (or its regularized version - see below) to obtain a, hopefully, unique function $f_{\mathcal{S}} : X \to Y$. This approach is called (*regularized*) *empirical risk minimization* (*ERM*) and it is used in many learning strategies, including deep networks. Note: empirical error of a function f is the average loss of f over the training data $\mathcal{S}$; generalization error of f is the expected loss of f over the space X. The second error is the mean of the first under random sampling of $\mathcal{S}$, so concentration of measure results can be used to relate these two errors. In a sense, ERM picks $f_{\mathcal{S}}$ using the empirical error as a proxy for the generalization error (which cannot be directly calculated). Learning theory, in step (3), then studies the empirical minimizer $f_{\mathcal{S}}$, its associated empirical error and its relation to the expectation

$$\int_X (f_{\mathcal{S}} - f_\mu)^2 d\mu_X \tag{1.3}$$

which is the generalization error of $f_{\mathcal{S}}$. Variations on the ERM strategy, such as regularization (see below), can play a role in further ensuring $f_{\mathcal{S}}$ has small generalization error.

We return now to the focus of this article: the choice of deep networks for step (1). Suppose that $\mathbb{F}$ consists of neural networks[7] with L layers. More precisely, the elements of $\mathbb{F}$ are functions $f_W : \mathbb{R}^d \to \mathbb{R}^q$ that are compositions of the following form, namely alternating between left-multiplication by a matrix W_k and application of a non-linear function σ,

(1.4)

$$\mathbb{R}^d \xrightarrow{W_1} \mathbb{R}^{d_1} \xrightarrow{\sigma} \mathbb{R}^{d_1} \xrightarrow{W_2} \mathbb{R}^{d_2} \xrightarrow{\sigma} \mathbb{R}^{d_2} \cdots \mathbb{R}^{d_{L-2}} \xrightarrow{W_{L-1}} \mathbb{R}^{d_{L-1}} \xrightarrow{\sigma} \mathbb{R}^{d_{L-1}} \xrightarrow{W_L} \mathbb{R}^q.$$

finishing with left-multiplication by a matrix W_L. Here the input to the network is assumed to be $x \in \mathbb{R}^d$; $W_k, k = 1, \cdots, L$ are matrices; and the (abusive) notation $\sigma : \mathbb{R}^j \to \mathbb{R}^j$ denotes coordinate-wise application of the activation function, which we assume is a GELU, that is a smooth version [14] of the rectified linear unit (ReLU) $\sigma : \mathbb{R} \to \mathbb{R}$, $\sigma(x) = \max(0, x)$. Each function $f_W \in \mathbb{F}$ is thus specified by L matrices of parameters $W_k, k = 1, \cdots, L$ of suitable dimensions. It is a composition of maps $\sigma \circ W_k$, followed by a final linear transformation W_L. Note that each map $\sigma \circ W_k : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ has ith coordinate $\sigma(w_i \cdot v)$ for $v \in \mathbb{R}^{d_{k-1}}$, where w_i is the ith row of W_k. Real-valued functions of this form, namely $v \mapsto \sigma(w \cdot v)$ for v, w in Euclidean space, are called *ridge functions* (due to the shape of their graph); they will play a role in the next theorem (Theorem 1.1). Finally, we call networks with $L > 2$ layers *deep networks*, and networks with $L = 2$ (only one σ layer) *shallow*.

For our purposes of giving learning-theoretic insight into this choice of approximator class in step (1), we will avoid going into many technical details of step (2) that arise for deep networks. We mention only in passing two of these. First, deep networks are often overparametrized (more parameters than data) so simple ERM

[7] Many variants of neural network exist, for example convolutional neural networks (CNNs) which are a subclass of the large class $\mathbb{F}$ defined here. The multilayer perceptron (MLP) [26] - the first artificial neural network to be widely used and studied - is on the other hand $\mathbb{F}$ with $L = 2$; like all $\mathbb{F}$, it is *fully connected*: each output of a layer depends on all inputs from the previous layer.

would not return[8] a unique $f_{\mathcal{S}}$. In this case, one typically uses "weight decay" to control the complexity of the chosen $f_{\mathcal{S}}$. Specifically, this means minimizing the *regularized* empirical risk,

$$L_S^{\lambda}(f_W) := \frac{1}{m}\sum_{i=1}^{m} \ell(f_W(x_i), y_i) + \lambda\rho^2,$$

where $\lambda > 0$ is a predefined hyperparameter, $\|\cdot\|$ is the Frobenius norm (i.e. the square root of the sum of the absolute squares of the matrix elements) and $\rho = \|W_1\|\|W_2\|\cdots\|W_L\|$. This controls the weights in the matrices W_k, keeping them small. Secondly, to accomplish the optimization, one typically uses a form of stochastic gradient descent (SGD) called mini-batch SGD[9]. For the reader unfamiliar with these techniques, the details do not matter; both are step (2) techniques that help to obtain $f_{\mathcal{S}} \in \mathbb{F}$ with low generalization error. Our focus will be on justifying step (1). For this, two questions arise: how close are functions of $\mathbb{F}$ to target functions of interest, and how does the choice of $\mathbb{F}$ affect our ability to search for optimal elements in $\mathbb{F}$. As we describe next, these two are a priori in tension, yet deep networks can resolve this conflict.

1.2. **Approximation and the curse of dimensionality.** Consider the first key step in the theory of machine learning: choosing a class $\mathbb{F}$ of parametric approximators for the class of target functions to be learned. Possible classes of approximators include generalized additive models, polynomials, radial basis functions, kernel machines as well as deep GELU or RELU networks that we descibed above. As mentioned, there are two main considerations in choosing $\mathbb{F}$: it should closely approximate a large class of target functions, and it should remain computationally efficient for optimization on the training data. Efficiency implies that the number of parameters in the approximators must not be exponential in d, the number of (real- or Boolean-valued) input variables. On the other hand, for some function classes, a number of parameters exponential in d/s may be required to achieve desired accuracy, where s is a measure of smoothness such as the number of bounded derivatives. This is an example of the so-called *curse of dimensionality* [4], a loose term that refers to situations where crucial resources needed by an algorithm (e.g. time, space, or data) depend exponentially on the dimension of the input. Note: "exponential dependence" and more generally $\mathcal{O}$ and Ω notation describe the limiting (asymptotic) behaviour of a real-valued function, e.g. runtime of an algorithm as a function of input size; see [36] for definitions.

1.2.1. *Curse of dimensionality.* Let $s \geq 1$ be an integer, and W_s^d be the Sobolev space of all functions of d variables with continuous partial derivatives of orders up to $s < \infty$

[8] It might also *overfit*, i.e. "fit to noise". This refers to a situation, especially likely for large $\mathbb{F}$, where $f \in \mathbb{F}$ could be chosen to match whatever random y_i was seen for each x_i in the pairs $(x_i, y_i) \in \mathcal{S}$, rather than $f_\mu(x_i)$. Such f will in general not be approximating f_μ but rather the "noise" in $\mathcal{S}$, and may thus have poor generalization error.

[9] This algorithm samples a "mini-batch" of training data from $\mathcal{S}$, computes the empirical loss on this mini-batch, and moves within parameter space so as to reduce this loss. At the next iteration a new mini-batch is sampled and so on. This has the effect of performing a slightly jittery (i.e. randomized) gradient descent for the full empirical loss function on $\mathcal{S}$, so as to minimize that loss without getting stuck in local minima.

supported on a compact subset of the unit ball $B^d = \left\{\mathbf{x} : \|\mathbf{x}\|_2 = \left(x_1^2 + \cdots + x_d^2\right)^{1/2} < 1\right\}$. In this section, we consider target functions in W_s^d.

Theorem 1.1. (*informal; see* [22] *for more detail*) *Let $\bar{\sigma} : \mathbb{R} \to \mathbb{R}$ be infinitely differentiable, and suppose there is a point where all derivatives of $\bar{\sigma}$ are nonvanishing*[10]*. For $f \in \mathrm{W}_s^d$, shallow networks with $\bar{\sigma}$ can approximate f within ϵ in the sup norm using a number of parameters*

$$N = \mathcal{O}(\epsilon^{-d/s}) \text{ and this is the best possible.} \tag{1.5}$$

Remarks

(1) The proof in [21] (see also [24]) relies on the fact that the algebraic polynomials in d variables of (total or coordinatewise) degree $< k$ are in the uniform closure of the span of $\mathcal{O}(k^d)$ functions of the form $x \mapsto \bar{\sigma}(w \cdot x)$. The estimate (1.5) then derives from an upper bound of $\epsilon = \mathcal{O}(k^{-s})$ on the approximation error by such polynomials. A survey of these and related upper and lower bounds can be found in [24]. Regarding lower bounds, [18] shows that, even allowing a much wider class of continuous $\bar{\sigma}$ which includes the GELU function σ mentioned in Section 1.1, achieving ϵ-approximation of functions $f \in \mathrm{W}_s^d$ with L^2-norm requires an exponential number $N = \Omega(\epsilon^{-d/s})$ of parameters.

(2) Since these results are all based on the approximation of the polynomial space by linear combinations of ridge functions, as implemented by shallow networks, one may ask whether it could be improved by using a different class of approximators. The answer relies on the concept of nonlinear d–width of the compact set W_s^d (cf. [7, 20]). The d-width results imply that the estimate in Theorem (1.1) is *the best possible* among *all* reasonable[11] [7] methods of approximating arbitrary functions in W_s^d.

(3) This exponential dependence on the dimension d to obtain approximation error $\mathcal{O}(\epsilon)$ is an example of the *curse of dimensionality* [4]. Note that the constants involved in $\mathcal{O}$ or Ω in the theorems will depend upon the norms of the derivatives of f as well as $\bar{\sigma}$.

Remark (1) tells us we will need $L > 2$ if we wish to use neural networks as proposed for $\mathbb{F}$. On the other hand, Remark (2) means, for all practical purposes, that any $\mathbb{F}$ that can approximate arbitrary functions from W_s^d to a specific level of accuracy will be cursed by computationally infeasible optimization (searching).

A motivating theme in our paper is that this impasse can be resolved if we do not seek to approximate all functions in W_s^d and we also (de facto) restrict searching in $\mathbb{F}$ to a specific manageable subclass $\widehat{\mathbb{F}}$ that contains approximators for the target functions of interest. This, we claim below, is the case for certain deep neural networks. In particular, we will argue that the key property these neural networks exploit to avoid the curse of dimensionality is *compositional sparsity*[12] which we define below.

[10] In particular, $\bar{\sigma}$ cannot be a polynomial.

[11] This includes our choice of $\mathbb{F}$.

[12] We use the term *compositional sparsity* following [6] instead of another equivalent term we used[22] earlier: *hierarchical local compositionality*.

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

To explain more precisely the argument, we define now the class of *sparse compositional 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 [6, 15, 27, 1, 16]).

A *compositional representation* of a function f is a presentation of f as a composition of constituent functions that are typically required to be of a specific kind, for example ridge functions of some dimension. Constituent functions may in general be multivariate, in which case they take as input the outputs of several other constituent functions. This compositional structure can be summarized by a directed acyclic graph (DAG) where internal nodes are the constituent functions, source nodes are input variables, and sink nodes are output variables (see Figure 1). If the relevant DAG is $\mathcal{G}$, we say the computed function is a *compositional $\mathcal{G}$-function*, and $\mathcal{G}$ is one of its *compositional graph representations* (which are in general non-unique).

Definition 2.1. (*informal*) *A function $f : \mathbb{R}^{d'} \to \mathbb{R}$ is* sparse *if it depends on at most a "small" number d_0 of variables (to be made clear from the context).*

Definition 2.2. *A function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be* compositionally sparse *if it can be represented as the composition of no more than $poly(d)$ constituent functions[13] each of which is sparse, in the sense that it depends on at most a constant number d_0 of variables.*

Similar definitions apply to Boolean functions (cf. *circuit complexity* [28]) but here we consider only real-valued functions on the reals. For simplicity of notation, we assume that $s = 1$; when $s > 1$ the dimensionality d in the theorems should be replaced by $\frac{d}{s}$.

The reader who is familiar with d_0-ary trees[14] may note that the number of internal nodes in a full d_0-ary tree is of the order of the number of leaves, so if $\mathcal{G}$ is a d_0-ary tree, compositional $\mathcal{G}$-functions are sparse with a number of constituent functions linear in d, whereas compositionally sparse functions that use a higher-degree polynomial number of constituent functions must have a graph representation with either reduced degree at many internal nodes (the constituent functions) or non-tree like structure (re-using output of constituents) or both.

Remark

[13] The expression $poly(d)$ means $O(p(d))$ for some polynomial p, and presupposes a sequence of f with arbitrarily large d; for example, detection of a car is a binary-valued function one can define on arbitrarily large images. The same upper bound of $p(d)$ should apply for all sufficiently large input sizes d. Likewise the upper bound d_0 mentioned next is constant in the sense that it is does not change as d increases, i.e. $d_0 \in O(1)$. Weaker but still useful definitions of compositional sparsity are possible, for example by requiring $d_0 \in O(\log d)$, but we will stay with the simple assumption of constant d_0.

[14] A k-ary tree is a generalization of binary ($k = 2$) or ternary ($k = 3$) trees to higher degree. More precisely, a k-ary tree is a tree, i.e. connected acyclic graph, where one node, designated the root, has degree at most k, while all other nodes have either degree 1 (and are called leaves) or else degree at most $k + 1$. Non-leaves (including the root) are called internal nodes. The tree is "full" if "at most" is replaced by "exactly". A k-ary tree is here viewed as a DAG by directing each edge towards the root.

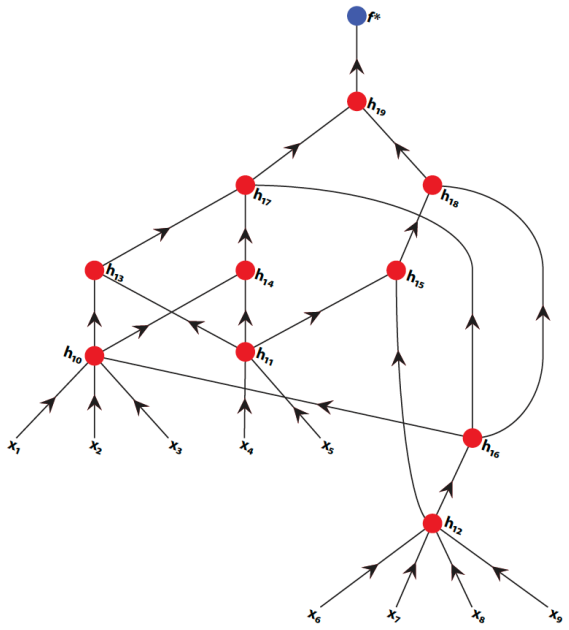

FIGURE 1. *An example of a compositional function graph (DAG). The internal nodes of the DAG $\mathcal{G}$, denoted by red dots, represent the constituent functions. The black arrows represent the input to the various nodes as indicated by the in edges of the red nodes, and the blue dot indicates the output value of the $\mathcal{G}$-function, f in this example.*

- Deep networks $f_W \in \mathbb{F}$ are functions with specific constituent functions (W_k and σ), and specific graph representation that is a priori *fully connected* in the sense that each node at layer k may depend on all nodes at layer $k-1$. This is not sparse. In particular, some constituent functions have d input variables. If one restricts, however, to matrices W_k with specific zero entries, this amounts to using constituent functions with fewer inputs and the resulting graph representation may be sparse. We refer to this de facto graph as the *architecture* of the network since it specifies the flow of information from input to output, and the location of trainable parameters. This is the case with CNNs: approximators are taken from a subclass $\widehat{\mathbb{F}}$ of $\mathbb{F}$ where W matrices have many zero entries, in specific patterns[15]. These

[15]In CNNs, some layers, called *convolutional layers*, have W_k with many zero entries in each row, while the set of nonzero entries is the same in every row (just differently located). This means each coordinate (output) of $W_k v$ will only depend on a few coordinates of v, and will be a dot product of that sub-vector of v with a small vector c, that is the same for all output coordinates (this is called *weight sharing*). Such matrices applied to image data can have the effect of taking a dot product of a "template" with a small patch of an image, for example to detect a specific local shape like an edge, and doing so in identical manner at all small patches in the large image (due to weight sharing). Just as corners are defined by an arrangement of edges, many functions we might try to learn for images, e.g. whether the image contains a car, can be expressed as compositions of successive local shape detectors, and are thus well-approximated by CNNs.

zero entries are never changed, i.e. no optimization of these parameters is carried out: we optimize only the other entries of the matrices W.

The class of sparse compositional functions and the class $\mathbb{F}$ of deep networks — or a subclass $\widehat{\mathbb{F}}$ thereof — form an interesting, specific pair (target function class, approximator class). In fact, the following theorem[16] by Mhaskar and Poggio [22] shows that functions with bounded first derivative — that can be represented by a *compositionally sparse* function graph (see right-hand side of Figure 1) — can be approximated arbitrarily well by deep, sparse RELU networks with $poly(d)$ trainable parameters[17].

Theorem 2.3. [22] *Let $\mathcal{G}$ be a Directed Acyclic Graph (DAG) with internal node set V, d the number of source nodes, and for each $v \in V$, let d_v be the number of in-edges of v. Consider compositional $\mathcal{G}$-functions $f : \mathbb{R}^d \mapsto \mathbb{R}$, where each of the constituent functions of Figure 1 is in the Sobolev space $\mathrm{W}_{s_v}^{d_v}$. Consider shallow and deep networks with infinitely smooth activation function. Then, to ϵ-approximate target functions f, the number of trainable parameters needed in a shallow network is exponential in d*

$$N_{shallow} = \mathcal{O}\left(\epsilon^{-\frac{d}{s}}\right),$$

where $s = \min_{v \in V} s_v$, while the number of trainable parameters needed in a deep network with architecture $\mathcal{G}$ is

$$N_{deep} = \mathcal{O}\left(\sum_{v \in V} \epsilon^{-d_v/s_v}\right).$$

Therefore, to approximate compositionally sparse $\mathcal{G}$-functions, deep networks with an associated graph that corresponds to $\mathcal{G}$ avoid the curse of dimensionality in approximating f for increasing d, whereas shallow networks, in general, cannot avoid the curse.

To see the last statement, recall from Definition 2.2 that compositional sparsity of f means V has polynomially many elements and each d_v is at most constant.

2.1. **Efficiently computable functions are compositionally sparse.** As we've seen, 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 (with constituent functions as stated)? In this section, we show that it is quite broad since it is equivalent to the class of efficiently computable functions.

We first provide, in Definition 2.4, a specific version of the definition of a *computable* function. For Boolean functions, computability is equivalent to computability by a Turing machine[18]. 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 [2, 5]), in the sense that there exist functions on the reals (such as the pseudoinverse) that are not computable. Here we bypass

[16] In it, we assume a smooth version of the RELU activation function (recently the RELU has been replaced in applications by smooth versions such as the GELU activation function[14]).

[17] To deduce this from the Theorem, note N_{deep} will thus be a sum of $|V|$ constants by Definition 2.2, where $|V|$ is polynomial in d.

[18] The Church-Turing thesis states that any real-world computation can be translated into an equivalent computation on a Turing machine (this is equivalent to using general recursive functions). Note, however, the Turing machine may need arbitrarily large memory.

these 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.

Definition 2.4. *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 vector $x \in I$ — for instance using rational numbers — into a representation for $f(x)$. The special case of* efficient computability *requires computability in time/space that are $poly(d)$.*

The following observation, cast here as a theorem, is simple but interesting (for a proof, see [25]).

Theorem 2.5. *Functions on $I \subset \mathbb{R}^d$ with Lipschitz continuity which are efficiently computable are compositionally sparse. Efficiently computable Boolean functions are compositionally sparse.*

2.2. **Efficient computability, compositional sparsity and deep, sparse RELU networks.** Here are two obvious but interesting consequences directly implied by the observations above.

2.3. **Efficient computability is equivalent to compositional sparsity.** Consider smooth real-valued functions in d variables, that is Lipschitz continuous functions, that are compositionally sparse. Theorem 2.3 shows that such functions are computed by deep GELU networks with a number of parameters which is $poly(d)$ (the same is true for Boolean functions). RELU networks can be simulated efficiently by a Turing machine, since each layer in a deep network corresponds to a finite number of steps of a Turing machine. On the other hand, Theorem 2.5 shows that efficiently computable functions are compositionally sparse. We therefore have:

Corollary 2.6. *For computable functions* (*if the function is on $I \subset \mathbb{R}^d$, Lipschitz continuity is required*)*, compositional sparsity is equivalent to $poly(d)$ computability.*

2.4. **Efficiently computable functions can be approximated by a deep, sparse network.** A direct implication of Theorems 2.5 and 2.3 is the following statement which essentially says that for any "reasonable" target functions, i.e., ones that are efficiently computable, the tension in learning systems between close approximation and efficient optimization will be avoided by neural networks that have suitable architecture.

Corollary 2.7. (*informal*) *All efficiently computable functions* (*if the function is on the reals, Lipschitz continuity is required*) *can be approximated without curse of dimensionality by deep networks with architecture matching the graph of a sparse representation of the function.*

Remark

- Recall that a function has usually several compositional representations and thus several compositionally sparse representations. The approximating network should match one of them. Whether this can be done by training on data, that is by the optimization step, is an open question - cf. Section 4. One of the main implications of Theorem 2.6 is that in principle all functions may be approximated by an *appropriately sparse* neural network without curse of dimensionality. Thus the assumption in several recent statistics papers (for instance [3]) that the regression function is some form of a *"generalized hierarchical interaction model"* can be avoided. This observation, in turn, may provide theoretical foundations for several approaches, including tensor representations of data such "as the Hierarchical Tucker format [12, 11], in representing broad classes of functions.
- This implies that perhaps the main challenge in machine learning is the discovery of a sparse compositional graph representing the class of functions to be learned (see Section 4 on CNNs and transformers; the sparse graph structure is known in CNNs and, we conjecture, is learned in transformers). As an aside, compositionality may play an interesting role in simplifying the task of optimization.

3. Learning theory: compositional sparsity can lead to orders-of-magnitude better bounds on expected error

Corollary 2.7 implies that deep GELU networks with polynomially many parameters (and suitable architecture) suffice to approximate compositionally sparse functions, thus enabling efficient (i.e. polynomial-time) training, that is optimization, w.r.t. given data and a chosen loss function. The optimized network will give the lowest possible (regularized) *empirical error* in the class. This is a surrogate for — but may not actually equal — the lowest possible in-class generalization error, known as *approximation error*, which is ultimately our goal. The discrepancy between these errors is known as the *estimation error* and there is a tradeoff. While we saw that choosing a larger class $\mathbb{F}$ reduces approximation error, it also increases the possibility of *overfitting*, (see Section 1.1 and its footnote on overfitting) where some network achieves small or zero empirical error on the training set but the network performs poorly on unseen data. This is especially likely in the overparametrized case. As we describe in the next Theorem, however, it turns out that compositional sparsity has a mitigating effect here too, independently of whether or not the network is overparametrized.

Measures of complexity in learning theory, such as Rademacher complexity or VC dimension, are quantities associated to learning problems, such that bounds on generalization error exist in terms of these quantities (Rademacher complexity, for example, bounds the difference between expected and empirical error). Higher complexity generally gives higher (upper or lower bounds on) generalization error.

The next Theorem [37, 8] therefore implies that sparsity of a network dramatically reduces generalization error.

Theorem 3.1. (*informal*) *The contribution to the Rademacher complexity of a deep network is smaller for sparse layers of weights than for dense layers: if there are k non-zero (i.e. trainable) entries in each row of a weight matrix W and the weight matrix is $n \times n$, then the contribution of the layer to the Rademacher complexity of the network is $\sqrt{\frac{k}{n}}\|W\|$ instead of $\|W\|$, as it would be for a dense layer. Here $\|\cdot\|$ is the Frobenius norm.*

In analogy with the approximation result of Theorem 1.1, the key property here is sparsity (k much smaller than n in the theorem). This property is for example ensured in convolutional layers of typical CNNs - see footnote 15. Notice that an equivalent result for underparametrized networks follows directly from considerations of VC dimension. Thus, focusing on a class $\widehat{\mathbb{F}} \subset \mathbb{F}$ of deep networks with architecture matching the compositionally sparse representation of target functions not only enables efficient search in a space where close approximators lie (as described in Section 2), but interestingly, it also ensures generalization error is also reduced. The next section looks more closely at the interplay between optimization and choice of sparse subclass $\widehat{\mathbb{F}}$.

4. Optimization and open questions

We now turn to two important architectures where compositional sparsity is exploited.

4.1. **The sparse graph is known: CNNs.** In the underparametrized case, recent work[15] has shown that an optimal tradeoff between approximation and generalization error can be achieved, assuming that optimization finds a good minimum. In the 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 selecting the one with lowest complexity. Recent work [38] 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 an exponential loss or the regularized square loss.

Thus a reasonable conjecture is that this optimization problem can be solved by SGD if the graph of the underlying target function *is known and takes the form of a compositionally sparse graph*. This is the case for CNNs and for the network shown in Figure 2.

4.2. **The sparse graph is unknown: transformers.** The second part of the argument is about the case of unknown function graphs. We focus here on the case of transformers.

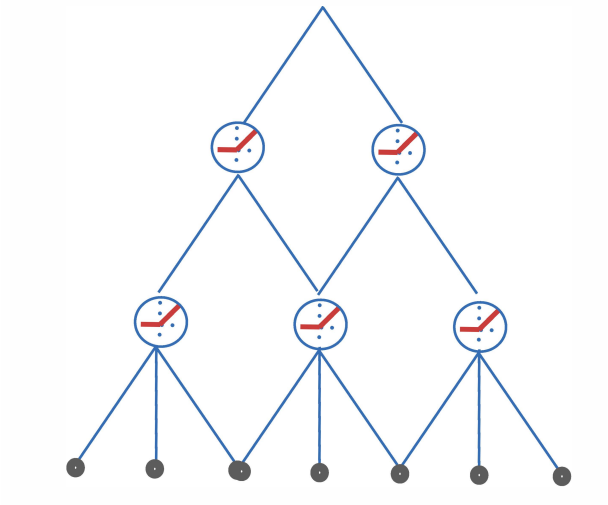

FIGURE 2. *The network here — similar to a CNN — reflects in a "hard-wired" way the sparse compositional function graph of the target function. Thus the function graph is supposed to be known here.*

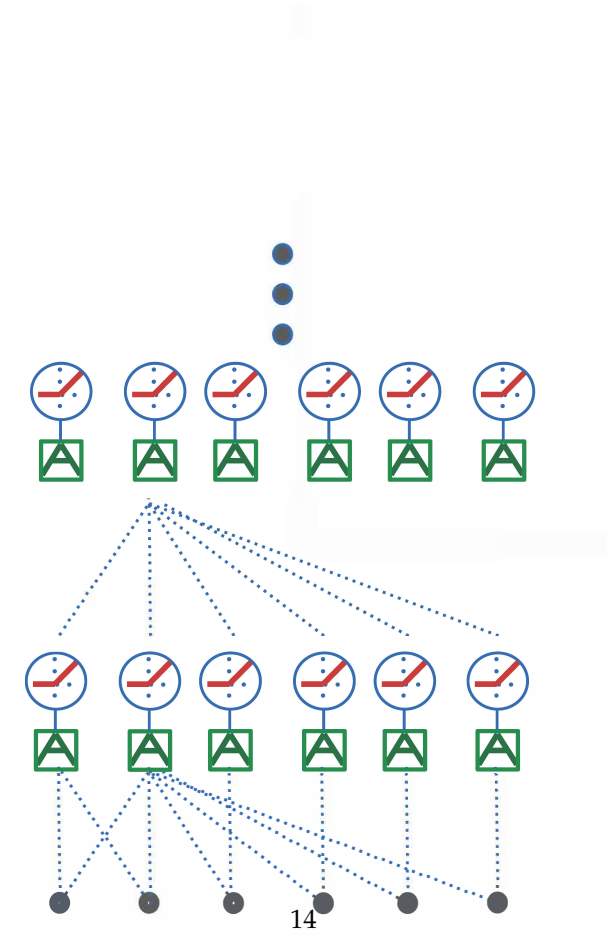

FIGURE 3. *Here a self-attention head (followed by a one-layer RELU network, though two layers are more common) selects and weights, for each input token, the relevant other tokens. The "A" box is the self-attention algorithm; the RELU circles represent the units of a one-layer RELUs. Notice that unlike the network of Figure 2 the selected input*

Transformers are deep learning networks that power the Large Language Models (LLMs) underlying ChatGPT and similar bots[33]. They are designed to deal with sequences of text — originally to solve the problem of translation from one language to another — but can be adapted to other modalities. The basic module of a transformer consists of a self-attention layer, followed by one or two GELU layers. This basic module is repeated to create much deeper networks. As shown in Figure 3 the input to each layer is a sequence of vectors, called tokens, each encoding a word or part of it. The output[19] is also such a token. In the attention layer each node, which receives an input token, selects a weighted subset of similar tokens in the sequence, effectively using a normalized Gaussian kernel with a Mahalanobis metric learned during training ([25], see also [13]). In summary for each node in the layer, self-attention maps a long sequence of context tokens into a linear combination of a few of them. The resulting vector is then the input to the GELU network. This has the effect of reducing dependence on inputs to a smaller number of variables[20], i.e. mimicking sparsity.

We conjecture that this process of learning the autoregressive prediction of the "next token" exploits the compositional structure of language where each token can be predicted from a sparse set of preceeding tokens. Recent papers are consistent with this conjecture (see for instance [17]), [23]. Our arguments show that simple transformers — just able to approximate a short sequence of sparse constituent functions — should approximate any (compositional) function efficiently computed by a Turing machine as long as the training data reflect the step-by-step output of the Turing machine.

The open question is how well can transformers recover the unknown sparse function graph of a function, when the training data do not contain the output of all intermediate constituent functions.

[19] To connect this setup with the basic learning theory we sketched earlier, note that a target function $f_\mu(x)$ taking values in $[0,1]$ as we defined in Section 1.1 can be interpreted as specifying the probability that $y = 1$ given x, in a setting where pairs $(x,0)$ and $(x,1)$ are both possible, i.e., binary y-values are non-deterministically associated to x under μ. This can be extended beyond binary y, to a case where y takes discrete values $a_1, \ldots, a_k$ non-deterministically for each x, by considering instead k target functions f_μ^i with the ith one specifying the probability that $y = a_i$ given x. This is close to the situation of transformers, except that rather than outputting the probabilities of $y = a_i$ for all i, the machine generates an output from the set $\{a_1, \ldots, a_k\}$ of tokens based on those learned probabilities. In particular, a transformer thus takes a sequence of input tokens, and outputs a next token that would be typical under μ. It is able to do this based on the approximation to f_μ^i it learned from its training data. For language modelling these are typically massive quantities of text retrieved from the internet. The transformer sequentially processes this text, thus observing many examples of which next token follows a sequence of tokens, thus learning to perform autoregressive prediction.

[20] Although transformers essentially search in the full $\mathbb{F}$, as the *self-attention module* learns to recognize, for a specific set of data, which inputs matter at each stage of composition, it essentially drives search towards an implicit subclass $\widehat{\mathbb{F}}$. For example, given a large corpus of English language text, self-attention would eventually come to recognize the *functional* roles of subject, verb, prepositions and direct object, in the sense that to predict the final word in the incomplete sentence "The red apple fell from the _____" the (trained) self-attention mechanism would guide the network to form the prediction mainly based on "apple" and "fell", as well as "from".

5. Machine learning and mathematics

We now switch to the broader question of replicating human intelligence in machines and to the implications for mathematics of the previous description of core principles of learning.

The first remark is that this is a time of unique opportunity to study principles of intelligence since we begin to have several instances of "intelligent" systems, such as AlphaZero which can outperform humans on the difficult tasks of playing go, shogi and chess, or ChatGPT which is widely considered to have passed the Turing test. Developments like these spurred Akshay Venkatesh' essay about a prospective Aleph(0) and its potential effects on research math. Our present contribution takes up the parallel question of how likely it is that such machines could soon appear. Based on the perspective we've outlined, this question hinges on the compositional sparsity of relevant functions that one might attempt to learn by machine. There are various possibilities for such functions to be used in research-level math, for example a machine that predicts reference papers of likely interest given a rough draft of a current project, or a machine that makes conjectures in some context, or one that produces tentative proofs or merely assists a (human) mathematician by suggesting next arguments in a proof (see also footnote 3).

Consider, for now, some version of producing next steps in a proof, given preceding interactions (goal statement, hypotheses, steps so far) as well as a knowledge base of true statements. The compositional sparsity principle discussed in the preceding pages, and especially its version for natural language — where the next output typically must be consistent with context and only a handful of the words that went before — seems to have a rough analogue in the structure of human-created mathematical proofs: each statement that appears in such a proof typically follows logically from only a small number of already established facts, and is relevant to the specific context. This does not mean that the choice of next output (in Math or in natural language) was evident from these same precursor statements, just that the new word or statement is consistent in some way with those few precursors. If a suitable network design is developed to exploit this type of sparsity, it seems quite plausible that at least simple versions of proof generation could be soon do-able by machine. This does not in any way mean there are not significant engineering and design challenges that would need to be overcome along the way to achieve even minimal functionality, only that based on the learning principles we have sketched, it does not seem like an impossible task.

Regarding context — mentioned above for natural language or logical arguments, and certainly highly relevant for conjectures — this is closely related to *association*, namely a probabilistic rather than deterministic link between items that are not directly related, in the direct way for example that a verb is conjugated to match the subject, or one mathematical statement is a direct consequence of others. It is rather a higher-level connection that emerges in the presence of many lower, more direct connections. We humans are familiar with the experience of being "reminded of" something. In this case, a new spontaneous image might arise in the presence of a complex web of many other circumstances. That web is the context to which is somehow associated the new image, and it does not require that this exact combination of circumstances was ever encountered before. An approximative description of how shallow and deep networks behave is in terms of memories

that are able to generalize and are not simply look-up tables. In this metaphor, the transformer output is the composition of associations at different levels. In fact, a hierarchy of *associations* was suggested by the first author as a core ability of the human brain [33]. In a similar vein, our arguments about sparse compositionality and its equivalence with efficiently computable functions imply that networks, such as a transformers, that can approximate simple, sparse functions can be trained, with appropriate data, to learn any compositional function of interest, that is any computer program. It follows that, if there exists some computer program capable of proving a set of theorems, then it is in principle possible to learn it from appropriate training sequences of tokens.

Gowers' manifesto [10] considers "the computational problem where the input is a mathematical statement and the output is a proof of that statement if it exists and otherwise a declaration that there is no proof." While this class is undecidable, Gowers points out that humans are primarily concerned with proofs of some limited length, but deciding if such a proof exists and producing it is still NP-complete. To better understand how humans nevertheless do prove theorems of interest, Gowers considers the set $\mathbb{B}$ (for "boring") of all pairs (S, P) such that S is a well-formed mathematical statement and P is a correct proof of S. He then defines the subset $\mathbb{M} \subset \mathbb{B}$ of problems that mathematicians are interested in, remarking: *".. I believe that there is something about* $\mathbb{M}$ *that makes the restriction to* $\mathbb{M}$ *of the proof-finding problem far easier algorithmically than the general proof-finding problem. That is why humans can do mathematics, and that is why if we can understand what is going on, then we should be able to program computers to do mathematics."* The project Gowers describes in the manifesto is devoted to better understanding:

(1) What distinguishes the pairs in $\mathbb{M}$ from general pairs in $\mathbb{B}$? and
(2) Why is it frequently feasible for humans to find a proof P of a statement S when $(S, P) \in \mathbb{M}$?

The perspective we have given here offers a preliminary high-level answer: if the italicized conjecture about algorithmic ease of proof-finding in $\mathbb{M}$ is true, then a necessary condition for $(S, P) \in \mathbb{B}$ to belong to $\mathbb{M}$ is that there exist a sparse function mapping established statements (that are known to be true) to S via "constituent" logical steps of certain acceptable kinds, and that P is an instance of such a sparse representation. This is of course an intuitive answer only, but it suggests that the principle of sparsity together with the detailed picture of $\mathbb{M}$ that will arise from Gowers' project can be useful not only for developing GOFAI — but also learning-based automated or interactive theorem provers.

Acknowledgments The first author thanks Fabio Anselmi, Sophie Langer, Tomer Galanti, Akshay Rangamani, Shimon Ullman, Yaim Cooper, Gitta Kutyniok, Lorenzo Rosasco, the Compositional Sparsity (CoSp) Collaboration (Santosh Vempala, Hrushikesh Mhaskar, Eran Malach, Seth Lloyd) for illuminating discussions. The second author thanks participants of the Fields Medal Symposium 2022, held in honour of Akshay Venkatesh, for stimulating discussions and talks on "The changing face of mathematics". The material in the present paper 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), the Natural Sciences and Engineering Research Council of Canada (NSERC RGPIN-2017-06901), AFSOR-THRL (FA8650-05-C-7262) and Lockheed-Martin.

References

1. Markus Bachmayr, Anthony Nouy, and Reinhold Schneider, *Approximation by tree tensor networks in high dimensions: Sobolev and compositional functions*, 2021.
2. 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.
3. Benedikt Bauer and Michael Kohler, *On deep learning as a remedy for the curse of dimensionality in nonparametric regression*, The Annals of Statistics **47** (2019), no. 4, 2261 – 2285.
4. Richard Bellman, *Dynamic programming*, 1st ed., Princeton University Press, Princeton, NJ, USA, 1957.
5. Holger Boche, Adalbert Fono, and Gitta Kutyniok, *Limitations of deep learning for inverse problems on digital hardware*, 2022.
6. Wolfgang Dahmen, *Compositional sparsity, approximation classes, and parametric transport equations*, 2022.
7. R. A. DeVore, R. Howard, and C. A. Micchelli, *Optimal nonlinear approximation*, Manuscripta mathematica **63** (1989), no. 4, 469–478.
8. Tomer Galanti, Mengjia Xu, Liane Galanti, and Tomaso Poggio, *Norm-based generalization bounds for compositionally sparse neural networks*, 2023.
9. Timothy Gowers, *Announcing an automatic theorem proving project*, **https://gowers.wordpress.com/2022/04/28/announcing-an-automatic-theorem-proving-project** (2022).
10. _____, *How can it be feasible to find proofs?*, (54 page manifesto, available at) **https://gowers.wordpress.com/ 2022/04/28/announcing-an-automatic-theorem-proving-project** (2022).
11. Lars Grasedyck, *Hierarchical Singular Value Decomposition of Tensors*, SIAM J. Matrix Anal. Appl. (2010), no. 31,4, 2029–2054.
12. Wolfgang Hackbusch and Stephan Kühn, *A new scheme for the tensor representation*, Journal of Fourier Analysis and Applications **15** (2009), 706–722.
13. Chi Han, Ziqi Wang, Han Zhao, and Heng Ji, *In-context learning of large language models explained as kernel regression*, 2023.
14. Dan Hendrycks and Kevin Gimpel, *Bridging nonlinearities and stochastic regularizers with gaussian error linear units*, CoRR **abs/1606.08415** (2016).
15. Michael Kohler and Sophie Langer, *Discussion of: "Nonparametric regression using deep neural networks with ReLU activation function"*, The Annals of Statistics **48** (2020), no. 4, 1906 – 1910.
16. Gitta Kutyniok, *Discussion of: "Nonparametric regression using deep neural networks with ReLU activation function"*, The Annals of Statistics **48** (2020), no. 4, 1902 – 1905.
17. Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang, *Transformers learn shortcuts to automata*, 2023.
18. V.E Maiorov, *On best approximation by ridge functions*, Journal of Approximation Theory **99** (1999), no. 1, 68–94.
19. John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon, *A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955*, AI Magazine **27** (2006), no. 4, 12.
20. H. Mhaskar, Q. Liao, and T. Poggio, *Learning real and boolean functions: When is deep better than shallow?*, Center for Brains, Minds and Machines (CBMM) Memo No. 45, also in arXiv (2016).
21. H. N. Mhaskar, *Neural networks for optimal approximation of smooth and analytic functions*, Neural Computation **8** (1996), no. 1, 164–177.
22. H.N. Mhaskar and T. Poggio, *Deep vs. shallow networks: An approximation theory perspective*, Analysis and Applications (2016), 829– 848.
23. Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D. Manning, *Characterizing intrinsic compositionality in transformers with tree projections*, 2022.
24. Allan Pinkus, *Approximation theory of the mlp model in neural networks*, Acta Numerica **8** (1999), 143–195.
25. T. Poggio, *How deep sparse networks avoid the curse of dimensionality: Efficiently computable functions are compositionally sparse*, CBMM memo 138 (2022).
26. F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological review **65** (1958), no. 6, 386–408.

27. Johannes Schmidt-Hieber, *Nonparametric regression using deep neural networks with ReLU activation function*, The Annals of Statistics **48** (2020), no. 4, 1875 – 1897.
28. M. Sipser, *Introduction to the theory of computation*, International Thomson Publishing, 1996.
29. L. G. Valiant, *A theory of the learnable*, Communications of the ACM **27** (1984), 1134–1142.
30. V. N. Vapnik and A. Y. Chervonenkis, *On the uniform convergence of relative frequences of events to their probabilities*, Th. Prob. and its Applications **17** (1971), no. 2, 264–280.
31. V.N. Vapnik and A. Ya. Chervonenkis, *The necessary and sufficient conditions for the uniform convergence of averages to their expected values*, Teoriya Veroyatnostei i Ee Primeneniya **26** (1981), no. 3, 543–564.
32. ———, *The necessary and sufficient conditions for consistency in the empirical risk minimization method*, Pattern Recognition and Image Analysis **1** (1991), no. 3, 283–305.
33. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, CoRR **abs/1706.03762** (2017).
34. Norbert Wiener, *Cybernetics: Or control and communication in the animal and the machine*, Wiley, 1948.
35. ———, *Men, machines, and the world about them*, Oct 1950.
36. Wikipedia contributors, *Big o notation — Wikipedia, the free encyclopedia*, 2023, [Online; accessed June-2023].
37. 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).
38. 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 **6** (2023), 0024.
39. Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar, *Leandojo: Theorem proving with retrieval-augmented language models*, 2023.

Center for Brains, Minds and Machines, MIT