# On efficiently computable functions, deep networks and sparse compositionality

tomaso poggio

February 2025

## 1 Introduction

In previous papers [1, 3] we have claimed that for each function which is efficiently Turing computable there exists a deep and sparse network which approximates it arbitrarily well. We also claimed a key role for compositional sparsity in this result. Though the general claims are correct some of of our statements may have been imprecise and thus potentially misleading. In this short paper we wish to formally restate our claims and provide definitions and proofs.

## 2 Main theorem

**Definition 1** (Efficient Turing computability)**.** There exists a deterministic Turing machine $M$ such that, given:

- A point $x \in [0,1]^d$ specified to $n$ bits of precision (i.e., a binary encoding of each coordinate of $x$ to $n$ bits),
- A precision parameter $\varepsilon > 0$ given by $\lceil \log_2(1/\varepsilon) \rceil$ bits,

the machine $M$ runs in time polynomial in $n + \log(1/\varepsilon)$ and produces a binary output (of length $\text{poly}(n, \log(1/\varepsilon))$) that is an $\varepsilon$-approximation to $f(x)$ (e.g., in supremum norm or Euclidean norm on $\mathbb{R}^m$).

**Theorem 2** (Polynomial-Time Computability Implies Neural Network Approximation Without Curse of Dimensionality)**.** *Let $f : [0,1]^d \to \mathbb{R}^m$ be a function that is* efficiently computable*. Then, for every $\varepsilon > 0$, there is a* feedforward neural network $\Phi_\varepsilon$ *(with size and depth at most polynomial in $n + \log(1/\varepsilon)$)—using any standard "universal" activation function (e.g. ReLU, sigmoid, etc.)—such that*

$$\max_{\substack{x \in [0,1]^d \\ x \text{ has an } n\text{-bit encoding}}} \left\| \Phi_\varepsilon(x) - f(x) \right\| \leq \varepsilon.$$

*In other words, $\Phi_\varepsilon$ approximates $f$ uniformly to within $\varepsilon$ on the discretely sampled inputs of precision $n$. By letting $n$ grow with the input precision (and letting $\varepsilon \to 0$), one obtains a family of deep neural networks that approximate $f$ arbitrarily closely, with overall size and depth scaling polynomially in the input-precision plus the approximation precision.*

**Proof.**

- Polynomial-time computability implies there is a Turing machine running in time $\text{poly}(n, \log(1/\varepsilon))$ for inputs $(x, \varepsilon)$ encoded with $n + \log(1/\varepsilon)$ bits.
- By standard results in circuit complexity, this yields a *polynomial-size (bounded-fan-in) Boolean circuit* family $\{C_{n,\varepsilon}\}$ that simulates the Turing machine on all inputs of length $n$.
- Each gate in the circuit can be replaced by a small neural "sub-network" that exactly or approximately realizes the corresponding Boolean function (e.g. via threshold or sigmoid units).
- Wiring these sub-networks together produces a *deep neural network* $\Phi_\varepsilon$ whose size (total number of neurons) and depth (number of layers) are polynomial in $n + \log(1/\varepsilon)$.
- On all $n$-bit encodings of $x$, this network's output matches (or approximates) the original function $f(x)$ to within $\varepsilon$.

# 3 Corollary

**Definition 3** (Compositional Sparsity)**.** A function $f : \mathcal{X}^d \to \mathcal{X}$ is *compositionally sparse* if it can be represented as the composition of at most $\mathcal{O}(d)$ constituent functions each of which is sparse, i.e., depends on at most a constant (small) number of variables $c$.

A compositionally sparse function can be represented in terms of a DAG (Directed Acyclic Graph), in which the leaves represent the inputs, the root denotes the output function, and the internal nodes represent the constituent functions. The maximum in-degree of the DAG is equivalent to $c$. It is always possible to show the DAG as a graph with multiple layers where there are nodes at each layer and connections only between one layer and the next one by using "dummy" nodes representing an identity transformation.

**Corollary 1** (Compositionally Sparse Approximation)**.** *Let $f : [0,1]^d \to \mathbb{R}^m$ be polynomial-time computable in the sense of Theorem 2. Then, for every $\varepsilon > 0$, there exists a compositionally sparse function*

$$g_\varepsilon \; : \; [0,1]^d \; \to \; \mathbb{R}^m$$

*such that:*

$$\max_{\substack{x \in [0,1]^d \\ x \text{ has an } n\text{-bit encoding}}} \| \, g_\varepsilon(x) \; - \; f(x) \| \; \leq \; \varepsilon,$$

*where each constituent function in the composition of $g_\varepsilon$ is itself realizable by a* shallow *(bounded-depth) neural network whose size (number of neurons) depends only polynomially on $n + \log(1/\varepsilon)$.*

*Proof of Corollary 1.* The statement follows directly from Theorem 2 and the standard correspondence between polynomial-size Boolean circuits and networks of shallow subnetworks. The steps in the proof are:

1. *Construct a Polynomial-Size Boolean Circuit.* By Theorem 2, $f$ is computed (to within $\varepsilon$) by a Turing machine running in time $\text{poly}(n, \log(1/\varepsilon))$. From classical circuit-complexity arguments, we can *unfold* this machine into a family of Boolean circuits $\{C_{n,\varepsilon}\}$, each of which has size (number of gates) bounded by a polynomial in $n + \log(1/\varepsilon)$. A similar statement is that the constituent functions of a Boolean function are sparse if the Boolean function corresponds to a program which is efficiently computable (i.e., in P). This is basically the definition of having a polynomial-size, bounded-fan-in circuit computing the function.

2. *Replace Each Gate or Set of Gates by a Shallow Neural Sub-Network.* Each gate in the circuit $C_{n,\varepsilon}$ is a small Boolean function (e.g. fan-in 2, such as AND, OR, NOT, or threshold). We convert an appropriately small set of such gates into a small (often depth-1 or depth-2) neural network:

   - *Threshold Gate Example*: A single neuron with a linear threshold activation can exactly replicate an AND/OR/NOT on $\{0, 1\}$ inputs.
   - *Sigmoid/ReLU Approximation*: If using continuous activations, we approximate these Boolean gates on $\{0, 1\}$ to arbitrary precision by adjusting the slope or bias parameters.

   Hence, each gate $g_j$ is replaced by a *shallow sub-network* $N_j$ with $O(1)$ or $O(\log n)$ neurons (depending on gate type).

3. *Layered Composition Yields a Compositionally Sparse Function.* The circuit $C_{n,\varepsilon}$ is a directed acyclic graph of gates, which can be topologically ordered in layers. Substituting each gate by its corresponding shallow network $N_j$ gives us a *layered composition*:

$$g_\varepsilon(x) \;=\; N_L\big(\cdots N_2(N_1(x))\cdots\big),$$

   where each $N_j$ has small depth (e.g. 1 or 2) and accepts only a bounded number of inputs from the previous stage (since each gate in the Boolean circuit had bounded fan-in). Consequently, $g_\varepsilon$ is *compositionally sparse* in the stated sense.

4. *Approximation Guarantee.* On the set of all $n$-bit encodings of $x \in [0, 1]^d$, the above composition $g_\varepsilon$ exactly replicates (or arbitrarily closely approximates) the circuit $C_{n,\varepsilon}$, and hence by Theorem 2, it lies within $\varepsilon$ of $f(x)$. Thus

$$\max_{\substack{x\in[0,1]^d \\ x \text{ has an } n\text{-bit encoding}}} \|g_\varepsilon(x) - f(x)\| \;\le\; \varepsilon.$$

5. *Size and Depth Bounds.* Because $C_{n,\varepsilon}$ has polynomial size in $n+\log(1/\varepsilon)$, the total number of shallow sub-networks (and total neurons across all layers) is also polynomial in $n + \log(1/\varepsilon)$. The final composition $g_\varepsilon$ is typically *deep* overall, but each individual subfunction in the composition is shallow, maintaining the *compositionally sparse* structure.

Hence, we conclude that for every $\varepsilon > 0$, there is a compositionally sparse function $g_\varepsilon$ (built from the functions corresponding to shallow neural sub-networks) that approximates $f$ to within $\varepsilon$. This completes the proof. □

The assumption in Theorem 2 (and in the corollary) that $f$ (or rather, its approximation) is in $P$ can be relaxed. In fact, $f$ only needs to be P/poly, that is, in the class of languages/functions such that there exists polynomial $p(n)$ such that for every input size $n$, there exists a circuit $C_n$ of size at most $p(n)$ that computes $f$ restricted to inputs of length $n$. $P$ is also called "uniform computation" and P/poly "non-uniform computation". Uniform computation has the very powerful property that the same constant sized program can generate the circuit computing the function for any input length.

# 4 Connection with Compositional Sparsity

Suppose we consider a Turing computable function $f : [0,1]^d \to \mathbb{R}$. Suppose the Boolean function corresponding to the Turing program that computes the function up to accuracy $\epsilon$ is compositionally sparse. This does not automatically imply that the original continuous and real-valued function is compositionally sparse. However, the following is true:

**Proposition 1** (Sparse Boolean Function Approximation by Sobolev Functions)**.** *Given a* sparse *Boolean function*

$$F \ : \ \{0,1\}^n \ \to \ \{0,1\}^m,$$

*there exists a set (or family) of Sobolev functions*

$$\{ f_\varepsilon \}$$

*on a continuous domain (say $\Omega \subset \mathbb{R}^d$) such that, for inputs encoded in $\{0,1\}^n$ and up to a desired approximation $\varepsilon$, each $f_\varepsilon$ yields outputs close to $F$ (interpreted in $\{0,1\}^m$).*

*In other words, for each finite bit-precision of input and output, one can pick (or construct) a Sobolev function $f_\varepsilon$ so that $f_\varepsilon$ closely matches the discrete outputs of $F$ on those discretized inputs.*

However, one must keep in mind that this "identification" does not imply a deep, structural equivalence between the continuous function and the Boolean function. It simply shows that, for each desired $\epsilon$, a suitable $f_\epsilon$ can be constructed so that it is in an appropriate Sobolev space and it is close to the Boolean function in a relevant norm. In particular, a different $\epsilon$ may lead to a different Sobolev function or even space. Corollary 1 is a more direct route to the statement that *efficiently Turing computable functions correspond to compositionally sparse functions*, where the statement has to mean that *each efficiently Turing computable function can be approximated arbitrarily well by a compositionally sparse function.*

*Remark:* The logic implicit in [1, 3] was that a) efficiently computable functions are compositionally sparse because their Boolean representation is compositionally sparse and that b) we can then use the main theorem in [2] to claim they can be approximated by deep sparse networks. The weak step in this argument is that a compositionally sparse Boolean representation of a continuous real-valued function on the reals, does not *directly* imply that the original function is compositionally sparse.

**Acknowledgments** I thank Dan Mitropolsky for explaining to me P/poly and other computer science concepts. This material is based upon work supported by the Center for Minds, Brains and Machines (CBMM), funded by NSF STC award CCF-1231216.

# References


[1] Tomaso Poggio and Maia Fraser, *Compositional sparsity of learnable functions*, Bulletin of the American Mathematical Society **61** (2024), no. 3, 438–456.

[2] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao, *Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review*, International Journal of Automation and Computing **14** (2017), no. 5, 503–519 (en).

[3] Tomaso A. Poggio, *How deep sparse networks avoid the curse of dimensionality: Efficiently computable functions are compositionally sparse*, 2023.

# 5 Appendices

## Appendix 1: Can Any Boolean Function Be Written as the Composition of Boolean Functions?

**A**ny Boolean function

$$F : \{0,1\}^n \to \{0,1\}^m$$

can be expressed as a finite composition of sparse Boolean functions (often called *gates*), each depending on only a limited subset of bits or intermediate outputs.

### 1. The General Idea

- A *Boolean circuit* (or *combinational logic*) representation accomplishes precisely this: it expresses a target Boolean function $F$ as a *layered composition* of sparser subfunctions $g_j$, each of which might be a standard gate like AND, OR, NOT, NAND, or a small Boolean block $\{0,1\}^k \to \{0,1\}$.
- Concretely, one builds a directed acyclic graph of gates. The input bits $x_1, \ldots, x_n$ feed into these gates, and each gate's output might feed into other gates. Eventually, the final layer of gates produces the $m$-bit output $\{0,1\}^m$.
- Symbolically, you can write

  $$F(x) \;=\; f_L\big(\ldots f_2\big(f_1(x)\big)\ldots\big),$$

  where each $f_\ell$ is a sparse Boolean function combining some of the inputs or partial outputs from earlier layers.

### 2. Efficiency and Circuit Complexity

- While every Boolean function *can* be represented in this way, the *size* of the circuit (the total number of gates) can become very large in the worst case. For a completely generic Boolean function, one might need exponentially many gates.
- However, for "structured" Boolean functions—e.g. those in the class **P** (polynomial-time computable)—there exist polynomial-size circuit families. This is a main theme of *circuit complexity* theory: understanding how large a circuit must be to compute (or approximate) a given function.

### 3. Related Constructions

- **Normal Forms.** Every Boolean function can be written in *disjunctive normal form (DNF)* or *conjunctive normal form (CNF)*. These forms are themselves compositions of smaller logical components (e.g. ANDs of literals, ORs of clauses).
- **Threshold Circuits.** Another style is to build circuits from *threshold gates* (gates that output 1 if a weighted sum of inputs exceeds a threshold). Again, these threshold gates are smaller Boolean subfunctions.
- **Multi-output Functions.** If $F$ has $m$ output bits, we can view each bit of $F(x)$ as a separate single-output function, all of which can be implemented by separate or partially shared subcircuits.

### Conclusion

Every Boolean function from $n$-bit inputs to $m$-bit outputs can indeed be written as a composition of sparser Boolean functions. We typically call this a *Boolean circuit*. Thus, the answer is *yes*: there is always some finite composition of subfunctions (gates) that realizes any desired overall mapping

$$\{0,1\}^n \to \{0,1\}^m.$$