The staircase property: How hierarchical structure can guide deep learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

This paper identifies a structural property of data that enables deep neural networks to learn hierarchically. We define the "staircase" property for Boolean functions, which posits that high-order Fourier coefficients are reachable from low-order Fourier coefficients along nested chains. We prove that functions satisfying this staircase property can be learned in polynomial time by homogeneous neural network architectures trained layerwise with stochastic coordinate descent. Our analysis rigorously shows that, during training, high-level features are built along the depth of the network by greedily combining lower-level features. We back our theoretical results with experiments.

1 Introduction

It has been observed empirically that neural networks can learn hierarchically. For example, a 'car' may be detected by first understanding simpler concepts like 'door', 'wheel', and so forth [1] in intermediate layers, which are then combined in deeper layers (c.f. [2]).

On the theoretical side, there is a push to understand how hierarchical learning occurs and to give guarantees on learning in neural networks. In this paper we are motivated by the following question:

> *Can we identify naturally structured and interpretable classes of hierarchical functions, and show how homogeneous DNNs are able to learn them?*

This is a refinement of the generic objective of trying to understand DNNs: We identify several key desiderata for any theoretical result in this direction. (1) *Natural structure*: We aim to capture naturally occurring data of interest, so the structural assumption must make conceptual sense. (2) *Interpretability*: If we hope to clearly interpret the inner workings of neural networks, understanding both how they classify and also how they learn, then we need a model for data that is interpretable to begin with. Interpretation of the representations occurring within a neural network is most clearly expressed with respect to structural properties of the data. Finally, (3) *Homogeneity of the network*: The network architecture and initialization should be symmetric and isotropic in a precise sense defined later on. This prevents using a carefully-crafted architecture and initialization to emulate general learning algorithms [3]. We view this style of restriction as being partway towards considering practical neural networks. The results in this paper aim to satisfy all three high-level objectives. The relation with prior work is discussed in Section 1.1.

This paper proposes a new structurally defined class of hierarchical functions and proves guarantees for learning by homogeneous neural networks. The function class is motivated by the following observation regarding two closely related functions, one of which cannot be learned efficiently and one of which can.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

In order to describe the functions, we first recall that any function on the n-dimensional hypercube $f : \{+1, -1\}^n \to \mathbb{R}$ can always be decomposed in the Fourier-Walsh basis as

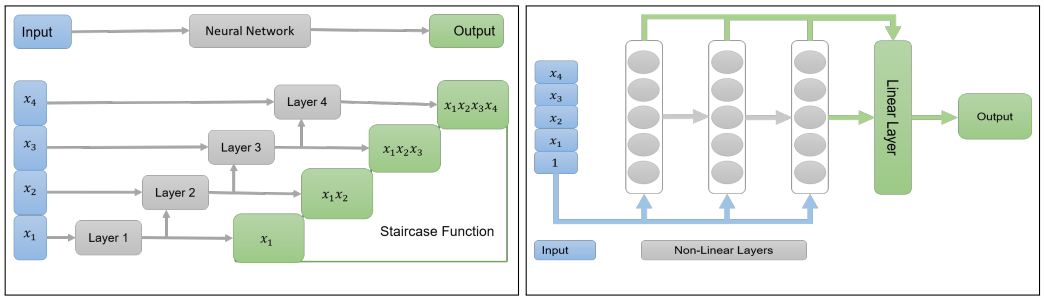
$$f = \sum_{S \subseteq [n]} \langle f, \chi_S \rangle \chi_S(x), \quad \text{where} \quad \chi_S(x) := \prod_{i \in S} x_i \tag{1}$$

and the inner product between any two functions is given by $\langle f, g \rangle = \mathbb{E} f(X) g(X)$ for $X \sim \mathrm{Unif}(\{+1, -1\}^n)$. The components of f are of the form $\chi_S(x)$. The class of functions χ_S for $|S| = k$ are efficiently SQ learnable if and only if k is a constant, and the same holds[1] for GD on neural nets with polynomial parameters [4] and for noisy SGD with sufficiently large batch [6]. In fact, [7] shows that gradients carry very little information required to reconstruct χ_S for moderately large S and hence gradient based training is expected to fail. Thus, we think of the component χ_S as *simple* if $|S|$ is small and *complex* if $|S|$ is large. Now consider instead the following *staircase* function:

$$S_k(x) = x_1 + x_1 x_2 + x_1 x_2 x_3 + x_1 x_2 x_3 x_4 + \cdots + \chi_{1:k} \,. \tag{2}$$

$S_k(x)$ has a hierarchical structure, where x_1 builds up to $x_1 x_2$, which builds up to $x_1 x_2 x_3$, and so on. In this case, one can hope for the neural network to first learn a low-degree approximation, picking up the feature x_1 and then proceeding to increment the degree approximation, learning the coefficient $x_1 x_2$, and eventually 'climbing the staircase' up to the large degrees. We refer to Fig. 1a for an illustration. We note that in the simple case of $S_k(x)$, we can instead take any (unknown) permutation of the indices $\{1, \ldots, n\}$ and include non-zero coefficients $\alpha_S \in \mathbb{R}$.

Our experiments in Fig. 2 show a dramatic difference between learning a parity and a staircase function - even with $n = 30$ and $k = 10$, the same network (with 5 ReLU ResNet layers) and the same hyperparameters can easily learn $S_k(x)$ to a vanishing error (Fig. 2b) whereas it cannot learn $\chi_{1:k}$ even up to any non-trivial error (Fig. 2a). The mechanism becomes clearer once we plot the corresponding Fourier coefficients of the network output against training iteration. We can see in Fig. 2c that the network cannot learn *any* Fourier coefficient relevant to $\chi_{1:10}$ whereas from Fig. 2d it is clear that the network learns them one by one with increasing complexity.



(a) An illustration of hierarchical learning where successive layers build upon the features from previous layers.

(b) An illustration of the proposed architecture. The solid blue and grey arrows represent sparse random connections

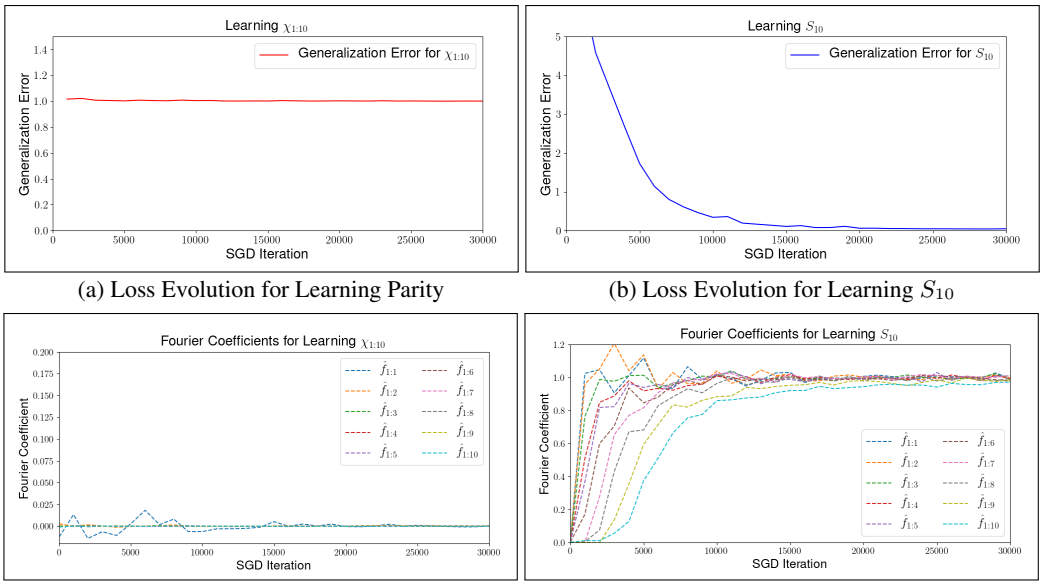
Figure 1: Hierarchical learning method and proposed architecture.

Our theoretical results shed light on this phenomenon, proving that certain homogeneous networks efficiently learn functions of the form of the staircase function $S_k(x)$, and more generally functions satisfying the following structural property. Given $S \subseteq [n]$, we denote by $\hat{g}(S) := \langle \chi_S, g \rangle$, the Boolean Fourier coefficient corresponding to the set S.

Definition 1.1. A function $g : \{-1, 1\}^n \to \mathbb{R}$ is a $[M_1, M_2]$-HBF (hierarchical Boolean function) if:

- for all $S \subset [n]$, if $\hat{g}(S) \neq 0$ then $|\hat{g}(S)| \in [M_1, M_2]$.
- for all $S \subset [n]$, if $\hat{g}(S) \neq 0$ and $|S| \geq 2$, there is $S' \subset S$ such that $|S \setminus S'| = 1$ and $\hat{g}(S') \neq 0$.

[1] One needs to prevent infinite precision like in SQ [4, 5], so gradients are given only up to a small error.



(a) Loss Evolution for Learning Parity

(b) Loss Evolution for Learning S_{10}

(c) Fourier Coefficients for Parity

(d) Fourier Coefficients for S_{10}

Figure 2: Comparison between training $\chi_{1:10}$ and S_{10} with the same network and $n = 30$. Here $\hat{f}_{1:i}$ denotes the Fourier coefficient $\langle \chi_{1:i}, f \rangle$ corresponding to the network output f.

Furthermore, g is said to be s-sparse (in the Fourier basis) if $|\{S : \hat{g}(S) \neq 0\}| \leq s$.

Even though we only consider hierarchical functions over the Boolean hypercube $\{-1, 1\}^n$ in our theoretical result, we believe that the techniques used in this work can be extended to other function spaces of interest, and exploit orthonormality of the corresponding Fourier basis functions. For this reason we give a fairly general definition of hierarchical functions in Section 2 which suggests that the relevant properties of the Boolean hypercube used in this work generalize naturally. We consider this notion of hierarchical functions to be an important technical contribution of this work.

Takeaways: Stairway to Heaven. In this work, we identify a natural definition of hierarchical structure in data that can aid a deep neural network to learn efficiently. A function with hierarchical structure has *simple* and *complex* components. This structure in the data is then reflected in the trained NN, where the simple components slowly build up to the complex ones. Specifically, we consider a neural network architecture with a homogeneous initialization and show theoretically that while the complex components in themselves can be hard to learn, the simpler components aid the neural network in learning the *entire* hierarchical structure. Our theoretical analysis is supplemented by experimental results with deep ReLU networks which show that training just the complex components alone results in slow learning whereas the neural networks exploit the hierarchical structure to quickly learn components of increasing complexity *one by one*.

1.1 Related Work

Statistical query emulation results For the general class of polynomial size neural network architecture (in the input dimension) with any choice of initialization, it is known that SGD can learn[2] any function class (including HBFs) that is efficiently learnable from samples (or efficiently PAC-learnable) [6], while GD can learn any function class that is efficiently learnable from statistical queries (SQ) [3]. However, these results rely on highly non-homogeneous architectures and initializations, with different parts of the nets responsible for different tasks that emulate the computations of general learning algorithms. In particular, it is not known how to obtain the emulation results of [3] for "homogeneous" architecture and initialization, e.g., when (1) the weights distribution at initialization is *exchangeable*, (2) the layers[3] are *isotropic*, i.e., for every pair of layers, the edges

[2]These reductions are for polynomial-time algorithms and for polynomial precisions on the gradients.

[3]A layer contains all the vertices at the same graph-distance from the input, where the graph is induced by the neural network architecture (we focus on directed networks without loops).

between the pair of layers have identically distributed weights and are exchangeable. In contrast, our architecture in Theorem 3.2 is closer to practice since it is a homogeneous neural network in the above sense, and our result further illustrates how features are built greedily over depth.

Hierarchical models of data Explicitly adding hierarchical structures into machine learning algorithms such as hierarchical Bayesian modeling and hierarchical linear modeling has proved successful in various machine learning tasks beyond deep learning [8–11]. For image data, [12, 13] propose hierarchical generative models of images and use them to motivate deep convolutional architectures, although these works do not provide proof that deep learning learns these generative models. [14] gives a layer-wise training algorithm for deep convolutional networks that provably learns a deep generative model of images, and demonstrates that it has comparable accuracy on CIFAR10 to training a convolutional network end-to-end. [15] similarly proposes a 'deep rendering model' which hierarchically models various levels of abstraction present in data (e.g., images), but does not have theoretical guarantees of learnability. The paper [16] proposes a generative model of data motivated by evolutionary processes, and proves in a formal sense that "deep" algorithms can learn these models, whereas shallow algorithms cannot. In contrast to our work, the "deep" algorithms considered by [16] are not descent algorithms on homogeneous deep neural network architectures.

Results closely related to ours can be found in [17] where they show that when two layer ReLU networks are trained with SGD, during the training process, the lower frequency components of the target function are learned first. Unfortunately, their results have an exponential dependence on the dimension. In our work, we leverage *depth* and the hierarchical Boolean function structure to ensure that higher-level Fourier coefficients are learned efficiently. Finally, [18], considers the problem of learning Boolean circuits of depth $O(\log n)$ via neural networks under product distributions using layer-wise gradient descent. While [18] requires the architecture to match the Boolean circuit being learned, in contrast, our architecture is homogeneous and independent of the function learned.

Power of depth Several works have studied hierarchical learning theoretically from the perspective of how representation power depends on depth. [19] shows that deep networks can represent a certain class of compositionally-created functions much more efficiently than shallow networks. [20] shows that certain smooth radial functions can be easily represented by 3 layer neural networks but need exponential (in dimension) number neurons to be represented by a two layer network. Based on a theoretical analysis for learning fractal distributions related to the Cantor set, [21] conjectures that if shallow networks are poor at representing a target function, then a deep network cannot be trained efficiently using gradient based methods even though it is easy to represent the target function with a deep network. [22] presents a depth separation result by showing that deep networks can produce highly oscillatory functions by building on the oscillations layer by layer. [23] uses this phenomenon to show a sharp representation theorem for arbitrary-depth ReLU networks.

Other theoretical works have proved depth separation theorems for training. [24] prove that a two hidden layer neural network where the first hidden layer is kept random and only the second layer (which is a quadratic Taylor model) is trained, provably outperforms just one hidden layer network. In [25, 26] it is proved that deep networks trained end-to-end with SGD and quadratic activation functions can efficiently learn a non-trivial concept class hierarchically, whereas kernel methods and lower-depth networks provably fail to do so. The class of functions studied by [25, 26] are those representable as the sum of neurons in a teacher network that is well-conditioned, has quadratic activations and has a depth of at most $\log\log n$, where n is the number of inputs. This function class is expressive but incomparable to the hierarchical function class studied in our work (e.g., we can learn polynomials up to degree n, whereas [26] is limited to degree $\log(n)$). Furthermore, our function class has the advantage of being naturally interpretable, with complex features (the high-order monomials) being built in a transparent way from simple features (the low-order monomials).

Gradient dynamics have been explored for the simplified case of deep linear networks in [27–29], where an 'incremental learning' phenomenon is observed in which the singular values are learnt sequentially, one after the other. This phenomenon is reminiscent of the incremental learning of the Fourier coefficients of the Boolean function in our setting (Fig. 2).

For real world data sets, [30] empirically shows that in many data sets of interest, simple neural networks trained with SGD first fit a linear classifier to the data and then progressively improve the approximation. Further, they demonstrate that shallow and deep networks have similar performance during the initial stages of training but that the performance of the simple networks tapers off before the complex ones.

Neural Tangent Kernel and random features A sequence of papers have studied convergence of overparametrized (or Neural Tangent Kernel regime) neural networks to the global minimizer of empirical loss when trained via gradient descent. In this regime, they show that neural networks

behave like kernel methods and give training and/or generalization guarantees. Because of the reduction to kernels, these results are essentially non-hierarchical. [31, 32] in fact show that deep networks in the NTK regime behave no better than shallow networks. We refer to [26] for a review of the literature related to NTK and shallow learning. Finally, we mention the related works [33–35] which consider the problem of learning low-degree (degree q) polynomials over $\mathbb{R}^n$, without a hierarchical structure assumption. They require $n^{\Omega(q)}$ neurons to learn such functions, which is super polynomial once $q \gg 1$. The results hold in the random features regime which is known to be weaker than neural tangent kernels.

1.2 Organization

In Section 2 we define hierarchical functions over the Boolean hypercube and in a greater level of generality. In Section 3, we give the problem setup, network architecture and the training algorithm. We state our rigorous guarantee that this training algorithm learns hierarchical Boolean functions in Section 3.4, and give an overview of the proof in Section 3.5, deferring the formal proof to the appendix. Finally, we show experimental results in Section 4 which validate our theoretical results.

2 General Hierarchical Structure

In this section, we give a natural extension of hierarchical Boolean functions (Definition 1.1) to more general function classes. Neural networks are efficient at representing affine transforms of the data and smooth low-dimensional functions [35]. Using this as a starting point, we introduce hierarchical functions over the space of continuous, real valued functions $[-R, R]^n \subseteq \mathbb{R}^n; R \in \mathbb{R}^+ \cup \{\infty\}$ without any reference to underlying measures but is flexible enough to add additional structures like measures and the corresponding L^2 norms. Set $R \in \mathbb{R}^+ \cup \{\infty\}$ and consider any sequence of functions $\mathcal{H} := \{h_k : k \in \mathbb{N} \cup \{0\}\}$ such that $h_k : \mathbb{R} \to \mathbb{R}$ such that h_0 is the constant function 1, and an affine transform $\mathcal{A} : \mathbb{R}^n \to \mathbb{R}^n$ such that $\mathcal{A}(x) = Ax + b$ for $A \in \mathbb{R}^{n\times n}; b \in \mathbb{R}^n$ We call a function $f : [-R, R]^d \to \mathbb{R}$ to be $(\mathcal{H}, \mathcal{A})$-polynomial if there exists a finite index set $I_f \subset (\mathbb{N} \cup \{0\})^n$ such that for some real numbers $(\alpha_{\mathbf{k}})_{\mathbf{k}\in I_f}$:

$$f(x) = \sum_{\mathbf{k}:=(k_1,\ldots,k_n)\in I_f} \alpha_{\mathbf{k}} \prod_{i=1}^{n} h_{k_i}(y_i)$$

We also define $\mathsf{Ord}(\mathbf{k}) := |\{i : k_i \neq 0\}|$ and a partial order '$\preceq$' over $(\mathbb{N} \cup \{0\})^n$ such that $\mathbf{k}' \preceq \mathbf{k}$ iff $k'_i \in \{0, k_i\}$ for every $i \in [n]$. We will call a $(\mathcal{H}, \mathcal{A})$-polynomial to be (M_1, M_2) hierarchical if

1. $M_1 \leq |\alpha_{\mathbf{k}}| \leq M_2$ for every $\mathbf{k} \in I_f$.
2. For every $\mathbf{k} \in I_f$ such that $\mathsf{Ord}(\mathbf{k}) \geq 2$, there exists $\mathbf{k}' \in I_f$ such that $\mathsf{Ord}(\mathbf{k}') = \mathsf{Ord}(\mathbf{k}) - 1$[4] and $\mathbf{k}' \preceq \mathbf{k}$.

We now extend the definition to general continuous functions. Suppose $d_{\mathcal{S}}$ is a pseudo-metric on the space of bounded continuous functions $\mathcal{C}^b([-R, R]^n; \mathbb{R})$. We call $f \in \mathcal{C}^b([-R, R]^n; \mathbb{R})$ to be $(\alpha, \beta, \mathcal{S})$ hierarchical if for every $\epsilon > 0$, there exists an (M_1, M_2) hierarchical $(\mathcal{H}, \mathcal{A})$-polynomial f_ϵ such that: $d_{\mathcal{S}}(f, f_\epsilon) < \epsilon$. We note some examples below:

1. Let μ be the uniform measure over $\{-1, 1\}^n$, $d_{\mathcal{S}}$ be the L^2 norm induced by μ, $\mathcal{H} = \{1, x\}$ and $\mathcal{A}$ be identity the identity mapping . We note that $[M_1, M_2]$-HBF in Definition 1.1 corresponds to $(M_1, M_2, L^2(\mu))$ hierarchical functions as defined above.
2. In the case when μ is the biased product measure over $\{-1, 1\}^n$, we can take $\mathcal{H} = \{1, x\}$ and $\mathcal{A}(x) = x - \mathbb{E}x$.
3. When μ is the isotropic Gaussian measure, we can take $R = \infty$, $\mathcal{H}$ to be the set of 1-D Hermite polynomials and $\mathcal{A}$ to be the identity. In case $\mu = \mathcal{N}(m, \Sigma)$, we instead take $\mathcal{A} = \Sigma^{-1/2}(x - m)$.
4. When $R < \infty$ and $\mathcal{S}$ is the L^2 norm with respect to the Lebesgue measure. $\mathcal{C}([-R, R]^n; \mathbb{R})$, we can take $\mathcal{H} = \{\exp(i\frac{\pi k x}{R}) : k \in \mathbb{Z}\}$ and $\mathcal{A} = I$. This allows us to interpret $(\mathcal{H}, \mathcal{A})$-polynomial approximations as Fourier series approximations.

[4] We may instead consider $\mathsf{Ord}(\mathbf{k}) - O(1)$

5. When $R < \infty$ and $\mathcal{S}$ is the uniform norm (or sup norm) over $\mathcal{C}([-R, R]^n; \mathbb{R})$, we can take $\mathcal{H} = \{1, x, x^2, \ldots, \}$ and $\mathcal{A} = I$. Since any continuous function can be approximated by a polynomial, this presents a large class of functions of interest.

In items 1-4, we consider these specific function classes $\mathcal{H}$ in order to make $(\mathcal{H}, \mathcal{A})$ monomials orthonormal under $L^2(\mu)$. We leave it as a direction of future work to extend our theoretical learning results in Theorem 3.2 to such function classes, due to the similarities of these to HBFs.

3 Homogeneous networks provably learn hierarchical Boolean functions

We now state our main theoretical result, which proves that a homogeneous neural network architecture with homogeneous initialization trained with a descent algorithm learns hierarchical Boolean functions in polynomial time.

3.1 Architecture

Our network architecture has neuron set V and edge set E, and is defined as follows. The neuron set is $V = V_{\text{in}} \sqcup V_1 \sqcup \ldots V_L$. Here $V_{\text{in}} = \{v_{\text{in},0}, v_{\text{in},1}, \ldots, v_{\text{in},n}\}$ is a set of $n+1$ inputs, and $|V_i| = W$ consists of W neurons on each layer. Furthermore, the edge set E is a sparse, random subset of all possible directed edges:

- each $(v_0, v_i) \in V_{\text{in}} \times V_i$ is in the edge set E independently with probability p_1, and
- each $(v_i, v_{i+1}) \in V_i \times V_{i+1}$ for $i \in [L-1]$ is in the edge set E independently with probability p_2.

For each edge $e \in E$, let there be a weight parameter a_e. And for each neuron $v \in V \setminus V_{\text{in}}$, let there be a bias parameter b_v. The parameters of the network are therefore $a \in \mathbb{R}^E$ and $b \in \mathbb{R}^{V \setminus V_{\text{in}}}$. For simplicity of notation, we concatenate these two vectors into one vector of parameters

$$w = [a \quad b] \in \mathbb{R}^E \oplus \mathbb{R}^{V \setminus V_{\text{in}}}.$$

For each $i \in [n]$, the ith input, $v_{\text{in},i} \in V_{\text{in}}$, computes x_i, and the 0th input, $v_{\text{in},0}$, computes a constant:

$$f_{v_{\text{in},i}}(x; w) = x_i, \text{ and } f_{v_{\text{in},0}}(x; w) = 1.$$

Given a neuron $v \in V \setminus V_{\text{in}}$, the function computed at that neuron is a quadratic function of a linear combination of neurons with edges to v, (i.e., the activation function is quadratic):

$$f_v(x; w) = \left(\sum_{e=(u,v) \in E} a_e f_u(x; w) \right)^2 + b_v.$$

The output of the neural network is the sum of the values of the neurons at the layers:

$$f(x; w) = \sum_{v \in V \setminus V_{\text{in}}} f_v(x; w).$$

We refer to Fig. 1b for an illustration.

Remark 3.1 (Homogeneity)**.** The architecture is homogeneous in the following sense: for any distinct pair of potential edges $(v_i, v_j), (v_i', v_j') \in V_i \times V_j$, the events that these edges are in E are independent and of equal probability. The same holds for any distinct pair of potential edges $(u, v_j), (u', v_j') \in V_{\text{in}} \times V_j$. Furthermore, the initialization during training (Algorithm 1) is perfectly homogeneous with all parameters are initialized to zero.

3.2 Loss function

Let the population loss function be the population mean-squared-error between the output f of the network and a function $g : \{-1, 1\}^n \to \mathbb{R}$ that we wish to learn. For any $x \in \{-1, 1\}^n$, $a \in \mathbb{R}^E$ and $b \in \mathbb{R}^{V \setminus V_{\text{in}}}$, define the point-wise loss and population loss functions (resp.):

$$\ell(x; w) = \frac{1}{2}(f(x; w) - g(x))^2; \ell(w) = \mathbb{E}_{x \sim \{-1,1\}^n} \ell(x; w)\,. \tag{3}$$

We will train the neural network parameters to minimize an L^2-regularized version of the loss function. Let $\lambda_1, \lambda_2 > 0$ be regularization parameters, and define:

$$\ell_R(w) = \ell(w) + \frac{1}{2} \sum_{\substack{e=(u,v)\in E \\ u\in V_{\text{in}}}} \lambda_1 a_e^2 + \frac{1}{2} \sum_{\substack{e=(u,v)\in E \\ u\notin V_{\text{in}}}} \lambda_2 a_e^2.$$

The distinct regularization parameters $\lambda_1, \lambda_2 > 0$ for the weights of edges from the input and prior neurons respectively are set for purely technical reasons. We refer to Section 3.5 for a brief explanation.

3.3 Training

We train the neural network to learn a function $g : \{-1, 1\}^n \to \mathbb{R}$ by running Algorithm 1. This algorithm trains layer-wise from layer 1 to layer L. The ith layer is trained with stochastic block coordinate descent, iterating through the neurons in V_i in an arbitrary fixed order, and training the parameters of each neuron $v \in V_i$ using the TRAINNEURON subroutine. Each call of TRAINNEURON runs stochastic gradient descent to train the subset of neural network parameters $w_v = \{a_e\}_{e=(u,v)\in E} \cup \{b_v\}$ directly associated with neuron v (i.e., the weights of the edges that go into v, and the bias of v), keeping the other parameters $w_{-v} = \{a_e\}_{e=(u',v')\in E \text{ s.t. } v\neq v'} \cup \{b_{v'}\}_{v'\in V\setminus(\{v\}\cup V_{\text{in}})}$ fixed.

Algorithm 1: TRAINNETWORKLAYERWISE

Input: Sample access to the distribution $\{(x, g(x))\}_{x\sim\{-1,1\}^n}$. Hyperparameters $W, L, p_1, p_2, \lambda_1, \lambda_2, \eta, B, \epsilon_1, \alpha, \tau$.

Output: Trained parameters of neural network after training layer-wise.

```
(V, E) ← random network constructed as in Section 3.1.
w^0 ← 0⃗, t ← 0                    // Initialize all weights and biases to zero.
for layer i = 1 to L do
    for neuron v ∈ V_i do
        // Train the neuron parameters, w_v, fixing other parameters
        w^{t+1} ← TRAINNEURON(v, w^t)
        t ← t + 1
    end
end
Return w^t
```

3.4 Result

We prove that Algorithm 1 learns hierarchical Boolean functions. For the sake of clarity, we defer the exact bounds on the parameters considered to Appendix B.

Theorem 3.2. *Let $g : \{-1, 1\}^n \to \mathbb{R}$ be an unknown s-sparse $[1/M, M]$-Hierarchical Boolean Function. Given $s, M > 1$, an accuracy parameter $\epsilon > 0$, a soundness parameter $0 < \delta < 1$, and access to random samples from $\{(x, g(x))\}_{x\sim\{-1,1\}^n}$, there is a setting of hyperparameters for Algorithm 1 that is polynomially-bounded, i.e.,*

$$1/\operatorname{poly}(n, s, M, 1/\epsilon, 1/\delta) \le W, L, p_1, p_2, \lambda_1, \lambda_2, \eta, B, \epsilon_1, \alpha, \tau \le \operatorname{poly}(n, s, M, 1/\epsilon, 1/\delta),$$

such that Algorithm 1 runs in $\operatorname{poly}(n, s, M, 1/\epsilon, 1/\delta)$ time and with probability $\ge 1 - \delta$ returns trained weights w satisfying that the population loss is bounded to the desired accuracy:

$$\ell(w) \le \epsilon.$$

3.5 Proof overview

We now briefly describe how Algorithm 1 learns a Hierarchical Boolean Function, giving a high-level depiction of the training process in the case that the target function is the staircase function $S_3(x) = x_1 + x_1x_2 + x_1x_2x_3$. We refer to Fig. 3 for an illustration of the training procedure, where

Algorithm 2: TRAINNEURON(v, w^0)

Input: Neuron $v \in V \setminus V_{\text{in}}$. Initial network parameters w^0. Access to random samples $(x, g(x))$ for $x \sim \{-1, 1\}^n$. Hyperparameters $\lambda_1, \lambda_2, \eta, B, \epsilon_1, \alpha, \tau$.

Output: Parameters of network after the subset of parameters w_v corresponding to neuron v is trained, while all other parameters w_{-v} remain fixed.

`// To avoid saddle points, randomly perturb the neuron parameters`

$w_v^1 \leftarrow w_v^0 + z$, where z is a noise vector whose entries are i.i.d. in $\text{Unif}([-\eta, \eta])$.

`// Run stochastic gradient descent on the neuron parameters`

$t \leftarrow 1$

while *true* **do**

  `// Approximate` $\nabla_{w_v} \ell_R$ `with minibatch size` B

  Draw i.i.d. data samples $(x^{t,1}, g(x^{t,1})), \ldots, (x^{t,B}, g(x^{t,B}))$

  $\xi^t \leftarrow \frac{1}{B} \sum_{i=1}^{B} \nabla_{w_v} \ell_R(x^{t,i}; w_{-v}^0, w_v^t)$

  `// Stop if we have reached an approximate stationary point`

  if $\|\xi^t\| \leq \epsilon_1$ **then** break out of the loop

  `// Update` w_v `in direction of the approximate gradient`

  $w_v^{t+1} \leftarrow w_v^t - \alpha \xi^t$

  $t \leftarrow t + 1$

end

`// Prune small network weights`

$w_v^{t+1} \leftarrow w_v^t$, rounding to 0 every entry of magnitude less than τ

Return $w = (w_{-v}^0, w_v^{t+1})$

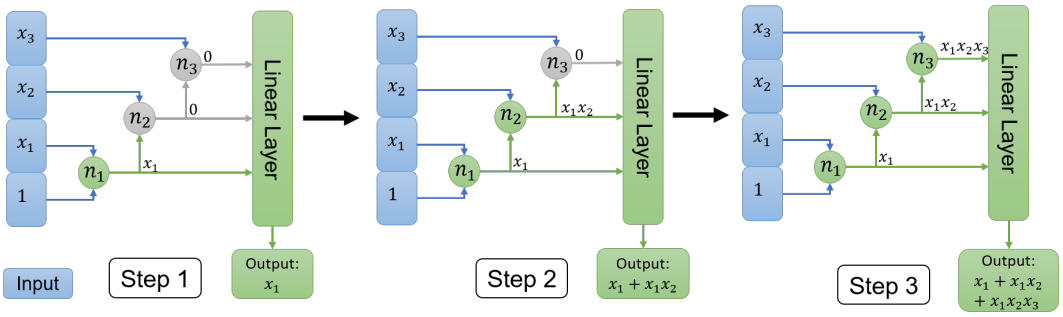


Figure 3: An illustration of the training procedure for learning $S_3(x) = x_1 + x_1x_2 + x_1x_2x_3$. The grey neurons are 'blank' and the green neurons are 'active'.

grey neurons are 'blank' (i.e., have identically zero output) and the green neurons are 'active' (i.e., compute a non-zero function). Initially all neurons are blank and the total output of the network is 0.

We set the random network topology connectivity hyperparameters p_1 and p_2 to be small, so that the network is sparse. We can thus show that the following invariant is maintained throughout training: any neuron has at most two active parents. This allows us to carefully analyze the stochastic gradient descent procedure each time a neuron is trained: in a key technical lemma, we show that every active neuron v has *exactly* two active parents u and u', and approximately computes a monomial given by the products of the parents' values, $f_v(x) \approx \chi_{S_v} \approx f_u(x) f_{u'}(x)$. We now present a run of the algorithm, breaking it up into "steps" for exposition.

Step 1: The algorithm iterates over neurons and trains them one by one using TRAINNEURON. Most of the neurons trained are left blank: for example, if a neuron v has the two inputs x_2 and 1, then by our key technical lemma the neuron could either remain blank or learn the product of the inputs, $x_2 = x_2 \cdot 1$. But the mean-squared error cannot decrease by learning x_2, since x_2 is orthogonal to the staircase function in the L^2 sense (i.e., $\langle S_3(x), x_2 \rangle = 0$, because the staircase function does not have x_2 as a monomial), so the neuron v remains blank.

Let t_1 be the first iteration at which the algorithm reaches a neuron $n_1 \in V_1$ that has x_1 and 1 as inputs. When the network trains n_1 using the sub-routine, we show that it learns to output $x_1 = x_1 \cdot 1$, since that is the highest-correlated function to $S_3(x)$ that n_1 can output. Combined with the linear layer, the overall neural network output becomes $f(x; w^{t_1}) \approx x_1$.

Step 2: The error function after Step 1 is $E_1(x) = S_3(x) - f(x; w^{t_1}) \approx x_1x_2 + x_1x_2x_3$. Now again, for many iterations the training procedure keeps neurons blank and keeps the error function unchanged, until at iteration t_2 it reaches a neuron n_2 with inputs x_1 (due to neuron n_1) and x_2 (directly from the input). Similarly to Step 1, when we train n_2, we show that it learns to output x_1x_2, which is the function with highest correlation to $E_1(x)$ which n_2 can output. Again, combined with the linear layer, the neural network now learns to output $f(x; w^{t_2}) \approx x_1 + x_1x_2$, so the error function has decreased to $E_2(x) = S_3(x) - f(x; w^{t_2}) \approx x_1x_2x_3$. The training proceeds until all the monomials in $S_3(x)$ are learned by the network.

Error Propagation and Regularization: A significant obstacle in analyzing layer-wise training is that outputs of neurons are inherently noisy because of incomplete training, and the error may grow exponentially along the depth of the network. In order to avoid this issue, we have two distinct regularization parameters λ_1, λ_2 and connectivity parameters p_1, p_2 for edges from inputs versus edges from neurons. In our proof of Theorem 3.2, we set $\lambda_1 \ll \lambda_2$ and $p_1 \gg p_2$, which ensures that after training a neuron (say n_2 above) the weight from the neuron n_1 (which has regularization parameter λ_2) is much smaller than the weight directly from the input x_2 (which has regularization parameter λ_1). Since the inputs are noise-free, this disallows exponential growth of errors along the depth. We conjecture that if the network is trained end-to-end instead of layer-wise, then one can avoid this technical difficulty and set $\lambda_1 = \lambda_2$ and $p_1 = p_2$, because of a backward feature correction phenomenon [26] where the lower layers' accuracy improves as higher levels are trained.

4 Experiments

The experiments in Fig. 2 were run on the same ReLU ResNet [36] with 5 non-linear layers, each 40 neurons wide. It was trained with SGD with the same step size on the square loss. We plot the generalization error and the necessary Fourier coefficients of the network every 10^3 iterations (computed with 10^5 fresh i.i.d samples from the hyper-cube). We refer to Appendix A for more details of the experimental parameters and additional experiments with varied depth and function structure.

5 Limitations and societal impacts

For simplicity of the proofs, the architecture and training algorithm that we choose are not common in practice: quadratic activations, a sparse connectivity graph, and layer-wise training[5] with stochastic block coordinate descent. We also perturb the weights with noise in order to avoid saddle points (although this has theoretical justification [40]), and we prune the low-magnitude weights to simplify the analysis (although it is known this may not deteriorate performance much in practice [41]). We emphasize that these limitations purely technical as they make the analysis more tractable, and we conjecture from our experiments that ReLU ResNets trained with vanilla SGD efficiently learn functions satisfying the staircase property.

This work is theoretical and does not deal directly with real world data, so it is not clear how this work would have direct societal impacts. However, this work takes a step towards developing natural and interpretable notions of what learning occurs in deep neural networks. Rigorously understanding and interpreting deep learning may aid us in better designing neural networks and hence in preventing unexpected, unjust and unexplainable behavior.

[5] Note that recent work has scaled layer-wise training to some degree of practical feasibility [37–39].

References

[1] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

[2] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[3] E. Abbe, P. Kamath, E. Malach, C. Sandon, and N. Srebro. On the power of differentiable learning versus pac and sq learning. *Preprint.*, 2021.

[4] Michael Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM (JACM)*, 45(6):983–1006, 1998.

[5] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning dnf and characterizing statistical query learning using fourier analysis. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 253–262, 1994.

[6] Emmanuel Abbe and Colin Sandon. On the universality of deep learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20061–20072. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/e7e8f8e5982b3298c8addedf6811d500-Paper.pdf`.

[7] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, pages 3067–3075. PMLR, 2017.

[8] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, 2016.

[9] Heather Woltman, Andrea Feldstain, J Christine MacKay, and Meredith Rocchi. An introduction to hierarchical linear modeling. *Tutorials in quantitative methods for psychology*, 8(1):52–69, 2012.

[10] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2):131–163, 1997.

[11] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[12] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

[13] Jong Chul Ye, Yoseob Han, and Eunju Cha. Deep convolutional framelets: A general deep learning framework for inverse problems. *SIAM Journal on Imaging Sciences*, 11(2):991–1048, 2018.

[14] Eran Malach and Shai Shalev-Shwartz. A provably correct algorithm for deep learning that actually works. *arXiv preprint arXiv:1803.09522*, 2018.

[15] Ankit B Patel, Tan Nguyen, and Richard G Baraniuk. A probabilistic theory of deep learning. *arXiv preprint arXiv:1504.00641*, 2015.

[16] Elchanan Mossel. Deep learning and hierarchal generative models. *arXiv preprint arXiv:1612.09057*, 2016.

[17] Ronen Basri, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *arXiv preprint arXiv:1906.00425*, 2019.

[18] Eran Malach and Shai Shalev-Shwartz. The implications of local correlation on learning some deep functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[19] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. Learning functions: when is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.

[20] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

[21] Eran Malach and Shai Shalev-Shwartz. Is deeper better only when shallow is good? *arXiv preprint arXiv:1903.03488*, 2019.

[22] Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

[23] Guy Bresler and Dheeraj Nagaraj. Sharp representation theorems for relu networks with precise dependence on depth. *Advances in Neural Information Processing Systems*, 33, 2020.

[24] Minshuo Chen, Yu Bai, Jason D Lee, Tuo Zhao, Huan Wang, Caiming Xiong, and Richard Socher. Towards understanding hierarchical learning: Benefits of neural representations. *arXiv preprint arXiv:2006.13436*, 2020.

[25] Zeyuan Allen-Zhu and Yuanzhi Li. What can resnet learn efficiently, going beyond kernels? *arXiv preprint arXiv:1905.10337*, 2019.

[26] Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. *arXiv preprint arXiv:2001.04413*, 2020.

[27] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[28] Gauthier Gidel, Francis Bach, and Simon Lacoste-Julien. Implicit regularization of discrete gradient dynamics in linear neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL `https://proceedings.neurips.cc/paper/2019/file/f39ae9ff3a81f499230c4126e01f421b-Paper.pdf`.

[29] Daniel Gissin, Shai Shalev-Shwartz, and Amit Daniely. The implicit bias of depth: How incremental learning drives generalization. *arXiv preprint arXiv:1909.12051*, 2019.

[30] Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin L Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. {SGD} on neural networks learns functions of increasing complexity. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, 2019.

[31] Alberto Bietti and Francis Bach. Deep equals shallow for relu networks in kernel regimes. *arXiv preprint arXiv:2009.14397*, 2020.

[32] Kaixuan Huang, Yuqing Wang, Molei Tao, and Tuo Zhao. Why do deep residual networks generalize better than deep feedforward networks?—a neural tangent kernel perspective. *Advances in Neural Information Processing Systems*, 33, 2020.

[33] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. Learning polynomials with neural networks. In *International conference on machine learning*, pages 1908–1916. PMLR, 2014.

[34] Gilad Yehudai and Ohad Shamir. On the power and limitations of random features for understanding neural networks. *arXiv preprint arXiv:1904.00687*, 2019.

[35] Guy Bresler and Dheeraj Nagaraj. A corrective view of neural networks: Representation, memorization and learning. In *Conference on Learning Theory*, pages 848–901. PMLR, 2020.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pages 583–593. PMLR, 2019.

[38] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In *International Conference on Machine Learning*, pages 4839–4850. PMLR, 2019.

[39] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pages 736–745. PMLR, 2020.

[40] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *International Conference on Machine Learning*, pages 1724–1732. PMLR, 2017.

[41] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Definition 1.1 and Theorem 3.2
   (b) Did you include complete proofs of all theoretical results? [Yes] See supplemental material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Supplemental material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] Do not run experiment multiple times.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplemental material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]