HOW CAN DEEP LEARNING PERFORMS DEEP (HIERARCHICAL) LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning is also known as hierarchical learning, where the learner *learns* to represent a complex target function by decomposing it into a sequence of simpler functions to reduce sample and time complexity. This paper formally analyzes how multi-layer neural networks can perform such hierarchical learning *efficiently* and *automatically* by applying stochastic gradient descent (SGD) or its variants.

On the conceptual side, we present a characterizations of how certain deep (i.e. super-constantly many layers) neural networks can still be sample and time efficiently trained on hierarchical learning tasks, when no known existing algorithm (including layer-wise training, kernel method, etc) is efficient. We establish a new principle called "backward feature correction", where *the errors in the lower-level features can be automatically corrected when training together with the higher-level layers*. We believe this is a key behind how deep learning is performing deep (hierarchical) learning, as opposed to layer-wise learning or simulating some known non-hierarchical method. [1]

1 INTRODUCTION

Deep learning is also known as hierarchical (feature) learning.[2] The term hierarchical learning can be defined as learning to represent the *complex* target function $g(x)$ using a composition of *much simpler* functions: $g(x) = h_L(h_{L-1}(\cdots h_1(x)\cdots))$. In deep learning, for example, each $h_\ell(\cdot)$ is usually represented by a linear operator followed with activation function. Empirically, the training process of deep learning is done by stochastic gradient descent (SGD) or its variants. After training, one can verify that the complexity of the learned *features* (i.e., $h_\ell(h_{\ell-1}(\cdots x \cdots))$ indeed increases as ℓ goes deeper — see (Zeiler & Fergus, 2014) or Figure 2. It has also been discovered for a long time that hierarchical learning, in many applications, requires fewer training examples (Bouvrie, 2009) when compared with non-hierarchical methods that learn $g(x)$ in one shot.

Hierarchical learning from a theoretical perspective. It is well-known that neural networks can *represent* a wide range of complicated functions using the composition of much simpler layers. Instead of learning a degree 2^L function from scratch, can hierarchical learning learn to represent it as a composition of L-quadratic functions, and thus learning one quadratic function at a time? The **main difficulty** is that being able to *represent* a complex target function in a hierarchical network does not necessarily guarantee *efficient* learning. For example, L layers of quadratic networks can represent all parity functions up to degree 2^L; but in the deep $L = \omega(1)$ setting, it is unclear if one can learn parity functions over $x \in \{-1,1\}^d$ with noisy labels via any efficient $\mathsf{poly}(d)$-time algorithm (Feldman et al., 2006), not to say via training neural networks.

So, for what type of functions can we formally *prove* that deep neural networks can hierarchically learn them? And, *how* can deep learning perform hierarchical learning to greatly improve learning efficiency in these cases?

[1]Note: although this paper has been circulated for a while, this is our first time to submit to an ML venue. This is a theory paper but we tried to make it suitable for a wider audience. We included many figures to support the connection between our theory and practice, but due to space limitation, many of the figures are deferred to Appendix A starting on Page 14.

[2]Quoting Bengio (2009), "deep learning aim at *learning feature hierarchies* with features from higher levels of the hierarchy formed by the composition of lower level features." Quoting Goodfellow et al. (2016) "the hierarchy of concepts allows the computer to learn complicated concepts by *building them out of simpler ones*."

Hierarchical learning and layer-wise learning. Motivated by the large body of theory works for two-layer networks, a tentative approach to analyze hierarchical learning in deep learning is via layer-wise training. Consider the example of using a multi-layer network with quadratic activation, to learn the following target function.

$$g(x) = \underbrace{x_1^2 + 2x_2^2}_{\text{low-complexity signal}} + 0.1 \underbrace{(x_1^2 + 2x_2^2 + x_3)^2}_{\text{high-complexity signal}} . \tag{1.1}$$

In this example, one may hope that the first train a two-layer quadratic network to learn simple, quadratic features (x_1^2, x_2^2), and the train another two-layer quadratic network *on top of the first one* learns a quadratic function over (x_1^2, x_2^2, x_3). In this way, one can hope for never needing to learn a degree-4 polynomial in one shot, but simply learning two quadratic functions in two steps. Is hierarchical learning in deep learning really this simple?

In fact, layer-wise training is known to perform poorly in practical deep learning, see Figure 7. A common sense is that when we train lower-level layers, it might over-fit to higher-level features. Using the example of (1.1), if one uses a quadratic network to fit $g(x)$, then the first-layer features may be trained too greedily and over-fit to high-complexity signals: for instance, the best quadratic network to fit $g(x)$ may learn features $(x_1 + \sqrt{0.1}x_3)^2$ and x_2^2, instead of (x_1^2, x_2^2). Now, if we freeze the first layer and train a second layer quadratic network on top of it (and the input), this "error" of $\sqrt{0.1}x_3$ can no longer be fixed thus we cannot fit the target function perfectly.

Our main message. On the conceptual level, we show (both theoretically and *empirically*) although lower-level layers in a neural network indeed tend to over-fit to higher complexity signals at the beginning of training, when training all the layers together — using simple variants of SGD — the presence of higher-level layers can eventually help reduce this type of over-fitting in lower-level layers. For example, in the above case the quality of lower-level features can improve from $(x_1 + \sqrt{0.1}x_3)^2$ again to get closer and closer to x_1^2 when trained together with higher-level layers. We call this *backward feature correction*. More generally, we identify *two critical steps* in the hierarchical learning process of a multi-layer network.

- The ***forward feature learning*** step, where a higher-level layer can learn its features using the simple combinations of the learned features from lower-level layers. This is an analog of layer-wise training, but a bit different (see discussions in (Allen-Zhu & Li, 2019a)) since all the layers are still trained *simultaneously*.
- The ***backward feature correction*** step, where a lower-level layer can learn to further *improve* its feature quality with the help of the learned features in higher-level layers. We are not aware of this being recorded in the theory literature, and believe it is *a most critical reason* for why hierarchical learning goes *beyond* layer-wise training in deep learning. We shall mathematically characterize this in Theorem 2.

Remark. When all the layers of a neural network are trained together, the aforementioned two steps actually occur *simultaneously*. For interested readers, we also design experiments to separate them and visualize, see Figure 3, 5, and 10 in Section A. On the theoretical side, we also give toy examples with mathematical intuitions in Section 1.2 to further explain the two steps.

Our technical results. With the help of the discovered conceptual message, we show the following technical results. Let input dimension d be sufficiently large, there exist a non-trivial class of "well-conditioned" L-layer neural networks with $L = \omega(1)$ and quadratic activations[3] so that:

- Training such networks by a variant of SGD *efficiently* and *hierarchically* learns this concept class. Here, by "efficiently" we mean time/sample complexity is $\mathsf{poly}(d/\varepsilon)$ where ε is the generalization error; and by "hierarchically" we mean the network learns to represent the concept class by decomposing it into a composition of simple (i.e. quadratic) functions, via forward feature learning and backward feature correction, to significantly reduce sample/time complexity.
- We are unaware of existing algorithm that can achieve the same result in polynomial time. For completeness, we prove super-polynomial lower bounds for shallow learning methods such as (1) kernel method, (2) regression over feature mappings, (3) two-layer networks with degree

[3]It is easy to measure the network's growing representation power in depth using quadratic activations (Livni et al., 2014). As a separate note, quadratic networks can perform as well as ReLU networks in practice (see Figure 11 on Page 25), and has particular cryptographic advantage (Mishra et al., 2020).

$\leq 2^L$ activations, or (4) the previous three with any regularization. Although proving separation is *not our main message*, we still illustrate in Section 1.2 that neither do we believe layer-wise training, or applying kernel method multiple (even $\omega(1)$ many) times can achieve poly-time.

To this extent, we have shown, at least for this class of L-layer networks with $L = \omega(1)$, *deep learning can indeed perform efficient hierarchical learning* when trained by a variant of SGD to learn functions not known to be learnable by "shallow learners" (including layer-wise training which can be viewed as applying two-layer networks multiple times). Thus, we believe that hierarchical learning (especially with backward feature correction) is *critical* to learn this concept class.

Difference from existing theory. Many prior works have studied the theory of deep learning. We try to cover them all in Section F but we summarize our difference from them as follows.

- Starting from Jacot et al. (2018), there is a rich literature that reduces multi-layer neural networks to kernel methods (e.g. neural tangent kernels, or NTKs). They approximate neural networks by *linear models* over (hierarchically defined) random features — which are not *learned* through training. They do not show the power of deep learning beyond kernel methods.
- Many other theories focus on two-layer networks but they do not have the *deep hierarchical* structure. In particular, some have studied *feature learning* as a process (Daniely & Malach, 2020; Li et al., 2020; Allen-Zhu & Li, 2021), but still cannot cover how the features of the second layer can help backward correct the first layer; thus naively repeating them for multi-layer networks may only give rise to layer-wise training as opposed to the full hierarchical learning.
- Allen-Zhu et al. (2019a) shows that 3-layer neural networks can learn the so-called "second-order NTK," which is not a linear model; however, second-order NTK is also learnable by doing a nuclear-norm constrained linear regression, which is still not truly hierarchical.
- Allen-Zhu & Li (2019a) shows that 3-layer ResNet can learn a concept class otherwise not learnable by kernel methods (within the same level of sample complexity). We discuss more in Section F, but most importantly, that concept class is learnable by applying kernel method twice.

In sum, prior works may have only studied a simpler but already non-trivial question: "can multi-layer neural networks efficiently learn simple functions that are *already learnable* by non-hierarchical models." While the cited works shed great light on the learning process of neural networks, in the language of this paper, they cannot justify how deep learning performs *deep hierarchical feature learning*. Our work is motivated by this huge gap between theory and practice.

Admittedly, with a more ambitious goal we have to sacrifice something. Notably, we study quadratic activations while some cited works can handle ReLU. Note this may be still fine: in practice, deep learning with quadratic networks perform very closely to ReLU ones, and significantly better than two-layer networks or neural kernel methods (see Figure 11). Hence, our theoretical result may also serve as a provisional step towards understating the deep learning process in ReLU networks.

1.1 Our Theorem

We give an overview of our theoretical result. The learner networks we consider are like DenseNets:

$$G(x) = \sum_{\ell=2}^{L} \langle u_\ell, G_\ell(x) \rangle \in \mathbb{R} \qquad \text{where} \quad G_0(x) = x \in \mathbb{R}^d, \qquad G_1(x) = \sigma(x) - \mathbb{E}[\sigma(x)] \in \mathbb{R}^d$$

$$G_\ell(x) = \sigma\left(\sum_{j \in \mathcal{J}_\ell} \mathbf{M}_{\ell,j} G_j(x)\right) \qquad \text{for } \ell \geq 2 \text{ and } \mathcal{J}_\ell \subseteq \{0, 1, \cdots, \ell-1\} \tag{1.2}$$

Here, σ is the activation function and we pick $\sigma(z) = z^2$ in this paper, $\mathbf{M}_{\ell,j}$'s are weight matrices, and the final output $G(x) \in \mathbb{R}$ is a weighted summation of the outputs of all the layers. The set $\mathcal{J}_\ell$ defines the connection graph. We can handle *any* connection graph with the only restriction being there is at least one "skip link."[4] To illustrate the main idea, we focus here on a regression problem in the teacher-student setting, although our result applies to classification as well as the *agnostic learning setting* (where the target network may also have label error). In this teacher-student regression setting, the goal is to learn some unknown target function $G^\star(x)$ in some concept class given samples $(x, G^\star(x))$ where $x \sim \mathcal{D}$ follows some distribution $\mathcal{D}$. In this paper, we consider

[4]In symbols, for every $\ell \geq 3$, we require $(\ell-1) \in \mathcal{J}_\ell$, $(\ell-2) \notin \mathcal{J}_\ell$ but $j \in \mathcal{J}_\ell$ for some $j \leq \ell-3$. As comparisons, the vanilla feed-forward network corresponds to $\mathcal{J}_\ell = \{\ell-1\}$, while ResNet (He et al., 2016) (with skip connection) corresponds to $\mathcal{J}_\ell = \{\ell-1, \ell-3\}$ with weight sharing (namely, $\mathbf{M}_{\ell,\ell-1} = \mathbf{M}_{\ell,\ell-3}$).

the target functions $G^\star(x) \in \mathbb{R}$ coming from the *same class* as the learner network:

$$G^\star(x) = \sum_{\ell=2}^{L} \alpha_\ell \cdot \langle u_\ell^\star, G_\ell^\star(x)\rangle \in \mathbb{R} \qquad \text{where} \quad G_0^\star(x) = x \in \mathbb{R}^d, \qquad G_1^\star(x) = \sigma(x) - \mathbb{E}[\sigma(x)] \in \mathbb{R}^d$$
$$G_\ell^\star(x) = \sigma\left(\sum_{j\in\mathcal{J}_\ell} \mathbf{W}_{\ell,j}^\star G_j^\star(x)\right) \in \mathbb{R}^{k_\ell} \qquad \text{for } \ell \geq 2 \text{ and } \mathcal{J}_\ell \subseteq \{0,1,\cdots,\ell-1\} \tag{1.3}$$

Since $\sigma(z)$ is degree 2-homogenous, without loss of generality we assume $\|\mathbf{W}_{\ell,j}^\star\|_2 = O(1)$, $u_\ell^\star \in \{-1,1\}^{k_\ell}$ and let $\alpha_\ell \in \mathbb{R}_{>0}$ be a scalar to control the contribution of the ℓ-th layer.

In the teacher-student setting, our main theorems can be sketched as follows:

Theorem (sketched)**.** *For every input dimension $d > 0$ and every $L = o(\log\log d)$, for certain concept class consisting of certain L-layer target networks defined in Eq. (1.3), over certain input distributions (such as standard Gaussian, certain mixture of Gaussians, etc.), we have:*

- *Within* $\mathsf{poly}(d/\varepsilon)$ *time/sample complexity, by a variant of SGD starting from random initialization, the L-layer quadratic DenseNet can learn this concept class with any generalization error ε, using* **forward feature learning + backward feature correction**. *(See Theorem 1.)*
- *As side result, we show any kernel method, any linear model over prescribed feature mappings, or any two-layer neural networks with arbitrary degree-2^L activations, require $d^{\Omega(2^L)}$ sample or time complexity, to achieve non-trivial generalization error such as $\varepsilon = d^{-0.01}$. (See Section O.)*

Remark. As we shall formally introduce in Section 2, the concept class in our theorem — the class of target functions to be learned — comes from Eq. (1.3) with additional width requirement $k_\ell \approx d^{1/2^\ell}$ and *information gap* requirement $\alpha_{\ell+1} \ll \alpha_\ell$ with $\alpha_2 = 1$ and $\alpha_L \geq \frac{1}{\sqrt{d}}$. The requirement $L = o(\log\log d)$ is very natural: a quadratic network even with constant condition number can output 2^{2^L} and we need this to be at most $\mathsf{poly}(d)$ to prove any efficient training result.

Remark. We refer the assumption $\alpha_{\ell+1} \ll \alpha_\ell$ as ***information gap***. In a classification problem, it can be understood as "α_ℓ is the marginal accuracy improvement when using ℓ-layer networks to fit the target function comparing to $(\ell-1)$-layer ones." We discuss more in Section B.1. For example, in Figure 4, we see $> 75\%$ of the CIFAR-10 images can be classified correctly using a two-hidden-layer network; but going from depth 7 to depth 8 only gives $< 1\%$ accuracy gain. Information gap was also pointed out in natural language processing applications (Tenney et al., 2019).

1.2 High-Level Intuitions

Intuitively, learning a *single* quadratic function is easy, but our concept class consists of a sufficiently rich set of degree $2^L = 2^{\omega(1)}$ polynomials over d dimensions. Using non-hierarchical learning methods, typical sample/time complexity is $d^{\Omega(2^L)} = d^{\omega(1)}$ — and we prove such lower bound for kernel (and some other) methods, even when all $k_\ell = 1$. This *is not surprising*, since kernel methods *do not* perform hierarchical learning so have to essentially "write down" all the monomials of degree 2^{L-1}, which suffers a lot in the sample complexity. Even if the learner performs kernel method $O(1)$ times, since the target function has width $k_\ell = d^{\Omega(1)}$ for any constant ℓ, this cannot avoid learning in one level a degree-$\omega(1)$ polynomial that depends on $d^{\Omega(1)}$ variables, resulting again in sample/time complexity $d^{\omega(1)}$.

Now, the *hope* for training a quadratic DenseNet with $\mathsf{poly}(d)$ time, is because it may decompose a degree-2^L polynomial into learning one quadratic function at a time. Easier said than done, let us provide intuition by considering an extremely simplified example: $L = 3$, $d = 4$, and

$$G^\star(x) = x_1^4 + x_2^4 + \alpha((x_1^4 + x_3)^2 + (x_2^4 + x_4)^2) \quad \text{for some } \alpha = o(1).$$

(Recall $L = 3$ refers to having two trainable layers that we refer to as the second and third layers.)

Forward feature learning: richer representation by over-parameterization. Since $\alpha \ll 1$, one may hope for the second layer $G_2(x)$ to learn x_1^4 and x_2^4 — which is quadratic over $G_1(x)$ — through some representation of its neurons; then feed this as input to the third layer. If so, the third layer $G_3(x)$ could learn a quadratic function over x_1^4, x_2^4, x_3, x_4 to fit the remainder $\alpha((x_1^4 + x_3)^2 + (x_2^4 + x_4)^2)$ in the objective. This logic has a critical flaw:

- *Instead of learning x_1^4, x_2^4, the second layer may as well learn $\frac{1}{5}(x_1^2 + 2x_2^2)^2$, $\frac{1}{5}(2x_1^2 - x_2^2)^2$.*

Indeed, $\frac{1}{5}(x_1^2 + 2x_2^2)^2 + \frac{1}{5}(2x_1^2 - x_2^2)^2 = x_1^4 + x_2^4$; however, *no* quadratic function over $\frac{1}{5}(x_1^2 + 2x_2^2)^2, \frac{1}{5}(2x_1^2 - x_2^2)^2$ and x_3, x_4 can produce $(x_1^4 + x_3)^2 + (x_2^4 + x_4)^2$. Therefore, the second layer needs to learn not only how to fit $x_1^4 + x_2^4$ but also the "correct basis" x_1^4, x_2^4 for the third layer.

To achieve this goal, we let the learner network to use (quadratically-sized) over-parameterization with random initialization. Instead of having only two hidden neurons, we will let the network have $m > 2$ hidden neurons. We show a critical lemma that the neurons in the second layer of the network can learn *a richer representation* of the same function $x_1^4 + x_2^4$, given by:

$$\{(\alpha_i x_1^2 + \beta_i x_2^2)^2\}_{i=1}^m$$

In each hidden neuron, the coefficients α_i, β_i behave like i.i.d. Gaussians. Indeed, $\mathbb{E}[(\alpha_i x_1^2 + \beta_i x_2^2)^2] \approx x_1^4 + x_2^4$, and w.h.p. when $m \geq 3$, we can show that a quadratic function of $\{(\alpha_i x_1^2 + \beta_i x_2^2)^2\}_{i=1}^m, x_3, x_4$ can be used to fit $(x_1^4 + x_3)^2 + (x_2^4 + x_4)^2$, so the algorithm can proceed. Note this is a completely different view comparing to prior works: here over-parameterization is not to make training easier in the current layer; instead, it enforces the network to learn a richer set of hidden features (to represent the same target function) that can be better used for higher layers.

Backward feature correction: improvement in lower layers after learning higher layers. The second obstacle in this toy example is that the second layer might not even learn the function $x_1^4 + x_2^4$ *exactly*. It is possible to come up with a distribution where the best quadratic over $G_1(x)$ (i.e., $x_1^2, x_2^2, x_3^2, x_4^2$) to fit $G^\star(x)$ is instead $(x_1^2 + \alpha x_3^2)^2 + (x_2^2 + \alpha x_4^2)^2$, which is only of magnitude α close to the ideal function $x_1^4 + x_2^4$. This is *over-fitting*, and the error $\alpha x_3^2, \alpha x_4^2$ *cannot* be corrected by over-parameterization. (More generally, this error in the lower-level features can propagate layer after layer, if one keeps performing forward feature learning without going back to correct them. This why we do not believe applying kernel method sequentially even $\omega(1)$ times can possibly learn our concept class in poly-time. We discuss more in Section 3.)

Let us proceed to see how this over-fitting on the second layer can be corrected by learning the third layer together. Say the **second layer has an "α-error"** and feeds the over-fit features $(x_1^2 + \alpha x_3^2)^2, (x_2^2 + \alpha x_4^2)^2$ to the third layer. The third layer can therefore use $\Delta' = \alpha((x_1^2 + \alpha x_3^2)^2 + x_3)^2 + \alpha((x_2^2 + \alpha x_4^2)^2 + x_4)^2$ to fit the remainder term $\Delta = \alpha((x_1^4 + x_3)^2 + (x_2^4 + x_4)^2)$ in $G^\star(x)$.

A very neat observation is that Δ' is only of magnitude α^2 away from Δ. Therefore, when the second and third layers are trained together, this "α^2-error" remainder Δ' will be subtracted from the training objective, so the **second layer can learn up to accuracy α^2, instead of α.** In other words, the amount of over-fitting is now reduced from α to α^2. We call this "backward feature correction" (see Figure 3, 5, and 10 in Section A). (This is also consistent with what we discover on *ReLU* networks in real-life experiments, see Figure 5 where we *visualize* such "over-fitting.")

In fact, this process $\alpha \to \alpha^2 \to \alpha^3 \to \cdots$ keeps going and the second layer can feed better and better features *to* the third layer (forward learning), via the reduction of over-fitting *from* the third layer (via backward correction). We can eventually learn $G^\star$ to arbitrarily small error $\varepsilon > 0$. When there are more than two trainable layers, the process is slightly more involved, and we summarize this *hierarchical* learning process in Figure 6 on Page 15.

Hierarchical learning in deep learning goes *beyond* layer-wise training. Our results also shed lights on the following observation in practice: typically layer-wise training (i.e. train layers one by one starting from lower levels) performs much worse than training all the layers together, see Figure 7. The fundamental reason is due to the missing piece of "backward feature correction."

2 Target Network and Learner Network

Target network. Recall we have defined the layers $G_2^\star(x), \dots, G_L^\star(x)$ of target networks in (1.3). The weights $\mathbf{W}_{\ell,j}^\star \in \mathbb{R}^{k_\ell \times k_j}$ for $j \in \mathcal{J}_\ell$ and we write $\mathbf{W}_{\ell,j}^\star = 0$ for $j \notin \mathcal{J}_\ell$. Our concept class to be learned consists of functions $G^\star \colon \mathbb{R}^d \to \mathbb{R}$ written as coordinate summation of each layer:[5]

$$G^\star(x) = \sum_{\ell=2}^L \alpha_\ell \cdot \mathbf{Sum}(G_\ell^\star(x)) := \sum_{\ell=2}^L \alpha_\ell \sum_{i\in[k_\ell]} G_{\ell,i}^\star(x)$$

[5]Our result trivially extends to the case when $\mathbf{Sum}(v)$ is replaced with $\sum_i p_i v_i$ where $p_i \in \{\pm 1\}$ for half of the indices. We refrain from proving that version for notational simplicity.

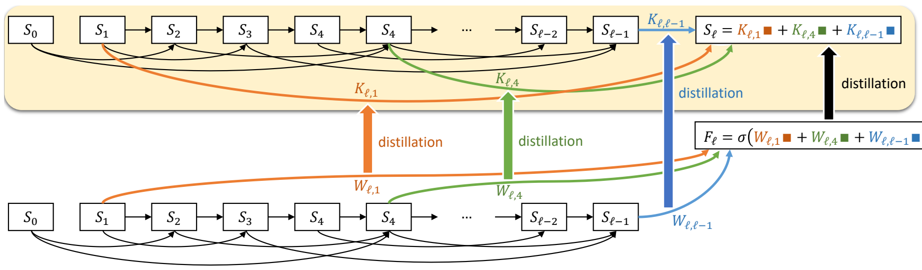

Figure 1: learner network structure with distillation

where $\mathbf{Sum}(v) := \sum_i v_i$, and it satisfies $\alpha_2 = 1$ and $\alpha_{\ell+1} < \alpha_\ell$. We will provide more explanation of the meaningfulness and necessity of information-gap $\alpha_{\ell+1} < \alpha_\ell$ in Section B.1.

It is convenient to define $S^\star_\ell(x)$ as the *hidden features* of target network (and $G^\star_\ell(x) = \sigma(S^\star_\ell(x))$).

$$S^\star_0(x) = G^\star_0(x) = x, \quad S^\star_1(x) = G^\star_1(x), \quad S^\star_\ell(x) := \textstyle\sum_{j=0}^{\ell-1} \mathbf{W}^\star_{\ell,j} G^\star_j(x) \quad \forall \ell \geq 2$$

Learner network. Recall we have constructed the learner network to be of the same structure (with over-parameterization, see (1.2)) as $G^\star$. We choose $\mathbf{M}_{\ell,0}, \mathbf{M}_{\ell,1} \in \mathbb{R}^{\binom{k_\ell+1}{2}\times d}$ and $\mathbf{M}_{\ell,j} \in \mathbb{R}^{\binom{k_\ell+1}{2}\times\binom{k_j+1}{2}}$ for every $2 \leq j \leq \ell-1$. In other words, the amount of over-parameterization is quadratic (i.e., from $k_j \to \binom{k_j+1}{2}$) per layer. Using samples $(x, G^\star(x))$ from an unknown target network $G^\star(x)$, our goal is to learn weight matrices $\mathbf{M}_{\ell,j}$ to satisfy

$$G(x) := \textstyle\sum_{\ell=2}^{L} \alpha_\ell \cdot \mathbf{Sum}(G_\ell(x)) \approx G^\star(x) \enspace .$$

2.1 Learner Network Re-parameterization

In this paper, for theoretical efficient training purpose, we work on a re-parameterization of the learner network. We use the following function to fit the target $G^\star(x)$:

$$F(x) = \textstyle\sum_{\ell=2}^{L} \alpha_\ell \cdot \mathbf{Sum}(F_\ell(x))$$

where the layers are defined as: $S_0(x) = G^\star_0(x)$, $S_1(x) = G^\star_1(x)$, and for $\ell \geq 2$:

$$S_\ell(x) = \textstyle\sum_{j\in\mathcal{J}_\ell, j\geq 2} \mathbf{K}_{\ell,j}\sigma\left(\mathbf{R}_j S_j(x)\right) + \sum_{j\in\{0,1\}\cap\mathcal{J}_\ell} \mathbf{K}_{\ell,j} S_j(x) \in \mathbb{R}^{k_\ell} \tag{2.1}$$

$$F_\ell(x) = \sigma\Big(\textstyle\sum_{j\in\mathcal{J}_\ell, j\geq 2} \mathbf{W}_{\ell,j}\sigma\left(\mathbf{R}_j S_j(x)\right) + \sum_{j\in\{0,1\}\cap\mathcal{J}_\ell} \mathbf{W}_{\ell,j} S_j(x) \Big) \in \mathbb{R}^{m} \tag{2.2}$$

Above, we shall choose m to be polynomially large and let

- $\mathbf{R}_\ell \in \mathbb{R}^{\binom{k_\ell+1}{2}\times k_\ell}$ be randomly initialized for every layer ℓ, not changed during training; and
- $\mathbf{W}_{\ell,j} \in \mathbb{R}^{m\times q}$, $\mathbf{K}_{\ell,j} \in \mathbb{R}^{k_\ell\times q}$ be trainablefor every ℓ and $j \in \mathcal{J}_\ell$, and the dimension $q = \binom{k_j+1}{2}$ for $j \geq 2$ and $q = d$ for $j = 0, 1$.

It is easy to verify that when $\mathbf{R}_\ell^\top \mathbf{R}_\ell = \mathbf{I}$ and when $\mathbf{W}_{\ell,j} = \mathbf{K}_{\ell,j}$, by defining $\mathbf{M}_{\ell,j} = \mathbf{R}_\ell \mathbf{K}_{\ell,j}$ we have $F_\ell(x) = G_\ell(x)$ and $F(x) = G(x)$. We remark that the hidden dimension k_ℓ can also be learned during training, see Algorithm 1 in Section C.[6]

Why this re-parameterization. We work with this re-parameterization $F(x)$ for *efficient training purpose*. It is convenient to think of $S_\ell(x)$ as the "*hidden features*" used by the learner network. Since $S_\ell(x)$ is of the same dimension k_ℓ as $S^\star_\ell(x)$, our goal becomes to prove that the hidden features $S_\ell(x)$ and $S^\star_\ell(x)$ are close up to unitary transformation (i.e. Theorem 2).

One may also consider $F_\ell(x) = \sigma(\mathbf{W}\cdots)$ and treat the pre-activation part $(\mathbf{W}\cdots) \in \mathbb{R}^m$ in (2.2) — instead of $S_\ell(x) \in \mathbb{R}^{k_\ell}$ — as the "over-parameterized hidden features." This over-parameterization is used to make the training *provably efficient*. As we shall see, we will impose regularizers during training to enforce $\mathbf{K}^\top\mathbf{K} \approx \mathbf{W}^\top\mathbf{W}$; and this idea of using a larger unit $\mathbf{W}$

[6]From this definition, it seems the learner needs to know $\{\alpha_\ell\}_\ell$ and $\{\mathcal{J}_\ell\}_\ell$; as we point out in Section C, performing grid search over them is efficient in $\mathsf{poly}(d)$ time. This can be viewed as neural architecture search. As a consequence, in the agnostic setting, our theorem can be understood as: *"the learner network can fit the labeling function using the **best** $G^\star$ from the concept class as well as the **best** choices of $\{\alpha_\ell\}_\ell$ and $\{\mathcal{J}_\ell\}_\ell$."*

for training and using a smaller unit $\mathbf{K}$ to learn the larger one can be viewed as *knowledge distillation*. One can then argue that the "over-parameterized hidden features" are also close to $S^\star_\ell(x)$ up to knowledge distillation and unitary transformation. Knowledge distillation is commonly used in practice (Hinton et al., 2015), and we illustrate this by Figure 1.

Truncated quadratic activation. To make our theory simpler, during training, it would be easier to work with an activation function that has bounded derivatives in the entire space (recall the derivative $|\sigma'(z)| = |z|$ is unbounded). We make a theoretical choice of a *truncated quadratic activation* $\widetilde{\sigma}(z)$ that is sufficiently close to $\sigma(z)$. Accordingly, we rewrite $F(x)$, $F_\ell(x)$, $S_\ell(x)$ as $\widetilde{F}(x)$, $\widetilde{F}_\ell(x)$, $\widetilde{S}_\ell(x)$ whenever we replace $\sigma(\cdot)$ with $\widetilde{\sigma}(\cdot)$. (For completeness we still include the formal definition in Appendix H.1.) Our lemma — see Appendix J.1 — shall ensure that $F(x) \approx \widetilde{F}(x)$ and $S_\ell(x) \approx \widetilde{S}_\ell(x)$. Thus, our *final learned network* $F(x)$ *is truly quadratic*. In practice, people use batch/layer normalizations to make sure activations stay bounded, but truncation is more theory-friendly.

Notation simplification. We concatenate the weight matrices used in the same layer ℓ as follows:

$$\mathbf{W}_\ell = (\mathbf{W}_{\ell,j})_{j\in\mathcal{J}_\ell} \qquad \mathbf{K}_\ell = (\mathbf{K}_{\ell,j})_{j\in\mathcal{J}_\ell} \qquad \mathbf{W}^\star_\ell = \left(\mathbf{W}^\star_{\ell,j}\right)_{j\in\mathcal{J}_\ell}$$

$$\mathbf{W}_{\ell\triangleleft} = (\mathbf{W}_{\ell,j})_{j\in\mathcal{J}_\ell, j\neq \ell-1} \qquad \mathbf{K}_{\ell\triangleleft} = (\mathbf{K}_{\ell,j})_{j\in\mathcal{J}_\ell, j\neq \ell-1} \qquad \mathbf{W}^\star_{\ell\triangleleft} = \left(\mathbf{W}^\star_{\ell,j}\right)_{j\in\mathcal{J}_\ell, j\neq\ell-1}$$

2.2 Training Objective

We focus our notation for the regression problem in the realizable case. We will introduce notations for the agnostic case and for classification in Section B.1 when we need them.

As mentioned earlier, to perform knowledge distillation, we add a regularizer to ensure $\mathbf{W}_\ell^\top \mathbf{W}_\ell \approx \mathbf{K}_\ell^\top \mathbf{K}_\ell$ so that $\mathbf{K}_\ell^\top \mathbf{K}_\ell$ is a low-rank approximation of $\mathbf{W}_\ell^\top \mathbf{W}_\ell$. (This also implies $\mathbf{Sum}(F_\ell(x)) \approx \mathbf{Sum}(\sigma(S_\ell(x)))$.) Specifically, we use the following training objective:

$$\widetilde{\mathbf{Obj}}(x; \mathbf{W}, \mathbf{K}) = \widetilde{\mathbf{Loss}}(x; \mathbf{W}, \mathbf{K}) + \mathbf{Reg}(\mathbf{W}, \mathbf{K})$$

where the ℓ_2 loss is $\widetilde{\mathbf{Loss}}(x; \mathbf{W}, \mathbf{K}) = (G^\star(x) - \widetilde{F}(x))^2$ and

$$\mathbf{Reg}(\mathbf{W}, \mathbf{K}) = \textstyle\sum_{\ell=2}^L \lambda_{3,\ell} \left\| \mathbf{K}_{\ell,\ell-1}^\top \mathbf{K}_{\ell\triangleleft} - \mathbf{W}_{\ell,\ell-1}^\top \mathbf{W}_{\ell\triangleleft} \right\|_F^2 + \sum_{\ell=2}^L \lambda_{4,\ell} \left\| \mathbf{K}_{\ell,\ell-1}^\top \mathbf{K}_{\ell,\ell-1} - \mathbf{W}_{\ell,\ell-1}^\top \mathbf{W}_{\ell,\ell-1} \right\|_F^2$$
$$+ \textstyle\sum_{\ell=2}^L \lambda_{5,\ell} \left\| \mathbf{K}_\ell^\top \mathbf{K}_\ell - \mathbf{W}_\ell^\top \mathbf{W}_\ell \right\|_F^2 + \sum_{\ell=2}^L \lambda_{6,\ell} \left(\|\mathbf{K}_\ell\|_F^2 + \|\mathbf{W}_\ell\|_F^2 \right) \quad .$$

For a given set $\mathcal{Z}$ consisting of N i.i.d. samples from the true distribution $\mathcal{D}$, the *training process* minimizes the following objective ($x \sim \mathcal{Z}$ denotes x is uniformly sampled from the training set $\mathcal{Z}$)

$$\widetilde{\mathbf{Obj}}(\mathcal{Z}; \mathbf{W}, \mathbf{K}) = \mathop{\mathbb{E}}_{x\sim\mathcal{Z}} [\widetilde{\mathbf{Obj}}(x; \mathbf{W}, \mathbf{K})] \tag{2.3}$$

The regularizers we used are just (squared) Frobenius norm on the weight matrices, which are common in practice. The regularizers associated with $\lambda_{3,\ell}, \lambda_{4,\ell}, \lambda_{5,\ell}$ are for *knowledge distillation propose* to make sure $\mathbf{K}$ is close to $\mathbf{W}$ (they are simply zero when $\mathbf{K}_\ell^\top \mathbf{K}_\ell = \mathbf{W}_\ell^\top \mathbf{W}_\ell$). They play *no role* in backward feature corrections (since layers ℓ and ℓ' for $\ell' \neq \ell$ are optimized *independently* in these regularizers). These corrections are done solely by SGD automatically.

For the original, non-truncated quadratic activation network, we also denote by

$$\mathbf{Loss}(x; \mathbf{W}, \mathbf{K}) = (G^\star(x) - F(x))^2 \text{ and } \mathbf{Obj}(x; \mathbf{W}, \mathbf{K}) = \mathbf{Loss}(x; \mathbf{W}, \mathbf{K}) + \mathbf{Reg}(\mathbf{W}, \mathbf{K}).$$

3 Statements of Main Result

We assume the *input distribution* $x \sim \mathcal{D}$ satisfies random properties such as isotropy and hyper-contractivity. We defer the details to Section D, while pointing out that not only standard Gaussian but even some mixtures of non-spherical Gaussians satisfy these properties (see Proposition D.1). For simplicity, the readers can think of $\mathcal{D} = \mathcal{N}(0, \mathbf{I})$ in this section.

We consider a concept class consisting of target networks satisfying the following parameters

1. (monotone) $d \geq k := k_2 \geq k_3 \geq \cdots \geq k_L$.

2. (normalized) $\mathbb{E}_{x\sim\mathcal{D}}\left[\mathbf{Sum}(G^\star_\ell(x))\right] \le B_\ell$ for some $B_\ell \ge 1$ for all ℓ and $B := \max_\ell\{B_\ell\}$.
3. (well-conditioned) the singular values of $\mathbf{W}^\star_{\ell,j}$ are between $\frac{1}{\kappa}$ and κ for all $\ell, j \in \mathcal{J}_\ell$ pairs.

Remark 3.1. Properties $1, 3$ are satisfied for many practical networks; in fact, many practical networks have weight matrices close to unitary, see (Huang et al., 2018). For property 2, although there may exist some worst case $\mathbf{W}^\star_{\ell,j}$, at least when each $\mathbf{W}^\star_{\ell,j}$ is of the form $\mathbf{U}_{\ell,j}\mathbf{\Sigma}\mathbf{V}_{\ell,j}$ for $\mathbf{U}_{\ell,j}, \mathbf{V}_{\ell,j}$ being random orthonormal matrices, with probability at least 0.9999, it holds $B_\ell = \kappa^{2^{O(\ell)}}k_\ell$ for instance for standard Gaussian inputs — this is small since $L \le o(\log\log d)$. Another view is that practical networks are equipped with batch/layer normalizations, which ensure $B_\ell = O(k_\ell)$.

Our results. In the main body of this paper, we state a special case of our main (positive result) Theorem 1 which is sufficiently interesting and has simpler notations. The full Theorem 1' is in Appendix H. In this special case, we assume there are absolute integer constants $C > C_1 \ge 2$ such that, the concept class consists of target networks $G^\star(x)$ satisfies the above three properties with parameters $\kappa \le 2^{C_1^L}, B_\ell \le 2^{C_1^\ell}k_\ell, k_\ell \le d^{\frac{1}{C^{\ell+C_1}}}$ and there is an information gap $\frac{\alpha_{\ell+1}}{\alpha_\ell} \le d^{-\frac{1}{C^\ell}}$ for $\ell \ge 2$; furthermore, suppose in the connection graph $\{2, 3, \cdots, \ell - C_1\} \cap \mathcal{J}_\ell = \varnothing$, meaning that the skip connections do not go very deep, unless directly connected to the input.

Theorem 1 (special case of Theorem 1')**.** *In the special case as defined above, for every sufficiently large $d > 0$, every $L = o(\log\log d)$, every $\varepsilon \in (0,1)$, consider any target network $G^\star(x)$ satisfying the above parameters. Then, given $N = \mathsf{poly}(d/\varepsilon)$ i.i.d. samples x from $\mathcal{D}$ with corresponding labels $G^\star(x)$, by applying Algorithm 1 (a variant of SGD) with over-parameterization $m = \mathsf{poly}(d/\varepsilon)$ and learning rate $\eta = \frac{1}{\mathsf{poly}(d/\varepsilon)}$ over the training objective (2.3), with probability at least 0.99, we can find a learner network F in time $\mathsf{poly}(d/\varepsilon)$ such that:*

$$\mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left(G^\star(x) - F(x)\right)^2 \le \varepsilon^2 \quad \textit{and} \quad \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left(G^\star(x) - \widetilde{F}(x)\right)^2 \le \varepsilon^2 \ .$$

We defer the detailed pseudocode of Algorithm 1 to Section C but make several remarks:

- Note $\alpha_{\ell+1} = \alpha_\ell d^{-\frac{1}{C^\ell}}$ implies $\alpha_L \ge d^{-\frac{1}{C}} \ge \frac{1}{\sqrt{d}}$. Hence, to achieve for instance $\varepsilon \le \frac{1}{d^4}$ error, the learning algorithm has to truly learn *all* the layers of $G^\star(x)$, as opposed to for instance ignoring the last layer which will incur error $\alpha_L \gg \varepsilon$.
- The reason we focus on $L = o(\log\log d)$ and well-conditioned target networks should be natural. Since the target network is of degree 2^L, we wish to have $\kappa^{2^L} \le \mathsf{poly}(d)$ so the output of the network is bounded by $\mathsf{poly}(d)$ for efficient learning.

The main conceptual and technical contribution of our paper is the "backward feature correction" process. To illustrate this, we highlight a critical lemma in our proof and state it as a theorem:

Backward Feature Correction Theorem

Theorem 2 (highlight of Corollary L.3d)**.** *In the setting of Theorem 1, during the training process, suppose the first ℓ-layers of the learner network* has achieved *ε generalization error,*

$$\textit{or in symbols,} \quad \mathbb{E}\left[\left(G^\star(x) - \textstyle\sum_{\ell'\le\ell}\alpha_{\ell'}\mathbf{Sum}(F_{\ell'}(x))\right)^2\right] \le \varepsilon^2 \ , \tag{3.1}$$

then for every $\ell' \le \ell$, there is unitary matrix $\mathbf{U}_{\ell'} \in \mathbb{R}^{k_{\ell'}\times k_{\ell'}}$ such that (we write $\alpha_{L+1} = 0$)

$$\mathbb{E}\left[\alpha_{\ell'}^2\left\|S^\star_{\ell'}(x) - \mathbf{U}_{\ell'}S_{\ell'}(x)\right\|^2\right] \lesssim (\alpha_{\ell+1}^2 + \varepsilon^2) \ .$$

In other words, once we have trained the first ℓ layers well enough, for some lower-level layer $\ell' \le \ell$, the "error in the learned features $S_{\ell'}(x)$ comparing to $S^\star_{\ell'}(x)$" is *proportional to* $\alpha_{\ell+1}$. Recall for fixed ℓ', as we increase ℓ the value $\alpha_{\ell+1}$ decreases, thus Theorem 2 suggests that

the lower-level features can actually get improved when we train higher-level layers together.

Remark 3.2. Theorem 2 is not a "representation" theorem. There might be other networks F such that (3.1) is satisfied but $S_{\ell'}(x)$ is not close to $S^\star_{\ell'}(x)$ at all. Theorem 2 implies *during the training process*, as long as we following carefully the training process of SGD, such "bad F" will be automatically avoided. We give more details in our intuition and sketched proof Section E.

Comparing to sequential kernel methods. Recall we have argued in Section 1.2 that our concept class is not likely to be efficiently learnable, if one applies kernel method $O(1)$ times sequentially. Even if one applies kernel method for $\omega(1)$ rounds, this is similar to *layer-wise training* and misses "backward feature correction." As we pointed out using examples in Section 1.2, this is unlikely to learn the target function to good accuracy either. In fact, one may consider "sequential kernel" together with "backward feature correction", but even this may not always work, since small generalization error does not necessarily imply sufficient accuracy on intermediate features *if we do not follow the SGD training process* (see Remark 3.2).[7]

Importance of hierarchical learning. To the best of our knowledge, for the concept class considered in this paper, we do not know any other simple algorithm to learn it in polynomial time, and the only simple learning algorithm we are aware of is to train a neural network to perform hierarchical learning. In other words, we believe we have presented a setting where we prove that training a neural network via a simple SGD variant can perform hierarchical learning, to solve an underlying problem that is not known solvable by existing algorithms, including applying kernel methods sequentially multiple times, tensor decomposition methods, sparse coding.

3.1 Backward Feature Correction: How deep? How much?

How deep does it need for the neural network to perform backward feature correction? In our theoretical result, we studied an extreme case in which training the L-th layer can even backward correct the learned weights on the first layer for $L = \omega(1)$ (see Theorem 2). In practice, we demonstrate that backward feature correction may indeed need to be deep. For the 34-layer WideResNet architecture on CIFAR tasks, see Figure 8 on Page 16, we show that backward feature correction happens for *at least* 8 *layers*, meaning that if we first train all the $\leq \ell$ layers for some large ℓ (say $\ell = 21$), the features in layer $\ell - 8, \ell - 7, \cdots, \ell$ *still need to be (locally) improved* in order to become the best features comparing to training all the layers together.

We also give a characterization on **how much** the features need to be backward corrected using theory and experiments. On the empirical side, we measure the changes given by backward feature correction in Figure 8 and 9. We detect that these changes are *local*: meaning although the lower layers need to change when training with higher layers together to obtain the highest accuracy, they *do not change by much* (the correlation of layer weights before and after backward correction is more than 0.9). In Figure 10, we also visualize the neurons at different layers, so that one can easily see backward feature correction is indeed a *local correction process in practice*.

This is consistent with our theory. Theorem 2 shows at least for our concept class, backward feature correction is a local correction, meaning that the amount of feature change to the lower-level layers (when trained together with higher-level layers) is only little-$o(1)$ due to $\alpha_{\ell+1} \ll \alpha_{\ell'}$.

Intuitively, the locality comes from "information gap", which asserts that the lower layers in $G^\star$ can already fit a majority of the labels. When the lower layers in G are trained, their features will already be close to those "true" lower-level features in $G^\star$ and only a *local correction* is needed.

We believe that *the need for only local backward feature corrections is one of the main reasons that deep learning works in practice* on performing efficient (deep) hierarchical learning. We refer to (Allen-Zhu & Li, 2019a) for empirical evidence that deep learning fails to perform hierarchical learning when information gap is removed and the correction becomes non-local, even in the teacher-student setting with a hierarchical target network exactly generating the labels. The main contribution of our theoretical result is to show that such local "backward feature correction" can be done automatically when applying (a variant of) SGD to the training objective.

What's in Supplementary Materials. We include an Appendix I to cover some missing details: including missing Figures 2-13, and experiments to support the connection between our theory and practice, theorem statements in the agonistic or classification settings, formal specification of the SGD training algorithm, formal specification of the data distribution, as well as a sketched proof. Detailed detailed proofs to Appendix II.

[7]One may also want to connect this to (Allen-Zhu & Li, 2019a): according to Footnote 16, the analysis from (Allen-Zhu & Li, 2019a) is analogous to doing "sequential kernel" for 2 rounds, but even if one wants to backward correct the features of the first hidden layer, its error remains to be α and cannot be improved to arbitrarily small.

References

Zeyuan Allen-Zhu and Yuanzhi Li. LazySVD: even faster SVD decomposition yet without agonizing pain. In *NeurIPS*, pp. 974–982, 2016.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Efficiently, Going Beyond Kernels? In *NeurIPS*, 2019a. Full version available at `http://arxiv.org/abs/1905.10337`.

Zeyuan Allen-Zhu and Yuanzhi Li. Can SGD Learn Recurrent Neural Networks with Provable Generalization? In *NeurIPS*, 2019b. Full version available at `http://arxiv.org/abs/1902.01028`.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature purification: How adversarial training performs robust deep learning. In *FOCS*, 2021. Full version available at `http://arxiv.org/abs/2005.10190`.

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers. In *NeurIPS*, 2019a. Full version available at `http://arxiv.org/abs/1811.04918`.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *NeurIPS*, 2019b. Full version available at `http://arxiv.org/abs/1810.12065`.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via overparameterization. In *ICML*, 2019c. Full version available at `http://arxiv.org/abs/1811.03962`.

Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *International Conference on Machine Learning*, pp. 584–592, 2014.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019a.

Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *CoRR*, abs/1901.08584, 2019b. URL `http://arxiv.org/abs/1901.08584`.

Ainesh Bakshi, Rajesh Jayaram, and David P Woodruff. Learning two layer rectified neural networks in polynomial time. *arXiv preprint arXiv:1811.01885*, 2018.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International Conference on Machine Learning*, pp. 583–593, 2019.

Yoshua Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

Digvijay Boob and Guanghui Lan. Theoretical properties of the global optimizer of two layer neural network. *arXiv preprint arXiv:1710.11241*, 2017.

Jacob V Bouvrie. *Hierarchical learning: Theory with applications in speech and vision*. PhD thesis, Massachusetts Institute of Technology, 2009.

Alon Brutzkus and Amir Globerson. Globally optimal gradient descent for a convnet with gaussian inputs. *arXiv preprint arXiv:1702.07966*, 2017.

Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 10835–10845, 2019.

Amit Daniely. Sgd learns the conjugate kernel class of the network. In *Advances in Neural Information Processing Systems*, pp. 2422–2430, 2017.

Amit Daniely and Eran Malach. Learning parities with neural networks. *arXiv preprint arXiv:2002.07400*, 2020.

Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2253–2261, 2016.

Simon S Du and Wei Hu. Width provably matters in optimization for deep linear neural networks. *arXiv preprint arXiv:1901.08572*, 2019.

Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, November 2018a.

Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018b.

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

Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. New results for learning noisy parities and halfspaces. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 563–574. IEEE, 2006.

Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pp. 797–842, 2015.

Rong Ge, Jason D Lee, and Tengyu Ma. Learning one-hidden-layer neural networks with landscape design. *arXiv preprint arXiv:1711.00501*, 2017.

Rong Ge, Rohith Kuditipudi, Zhize Li, and Xiang Wang. Learning two-layer neural networks with symmetric inputs. *arXiv preprint arXiv:1810.06793*, 2018.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Linearized two-layers neural networks in high dimension. *arXiv preprint arXiv:1904.12191*, 2019.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Boris Hanin and Mihai Nica. Finite depth and width corrections to the neural tangent kernel. *arXiv preprint arXiv:1909.05989*, 2019.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

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*, pp. 770–778, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jiaoyang Huang and Horng-Tzer Yau. Dynamics of deep neural networks and neural tangent hierarchy. *arXiv preprint arXiv:1909.08156*, 2019.

Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in neural information processing systems*, pp. 103–112, 2019.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pp. 586–594, 2016.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Yuanzhi Li and Zehao Dou. When can wasserstein gans minimize wasserstein distance? *arXiv preprint arXiv:2003.04033*, 2020.

Yuanzhi Li and Yingyu Liang. Provable alternating gradient descent for non-negative matrix factorization with strong correlations. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2062–2070. JMLR. org, 2017.

Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, 2018.

Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation. In *Advances in Neural Information Processing Systems*, pp. 597–607.

http://arxiv.org/abs/1705.09886, 2017.

Yuanzhi Li, Yingyu Liang, and Andrej Risteski. Recovery guarantee of non-negative matrix factorization via alternating updates. In *Advances in neural information processing systems*, pp. 4987–4995, 2016.

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *COLT*, 2018.

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *arXiv preprint arXiv:1907.04595*, 2019a.

Yuanzhi Li, Tengyu Ma, and Hongyang R Zhang. Learning over-parametrized two-layer relu neural networks beyond ntk. *arXiv preprint arXiv:2007.04596*, 2020.

Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019b.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*, 2020a.

Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very deep transformers for neural machine translation. *arXiv preprint arXiv:2008.07772*, 2020b.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pp. 855–863, 2014.

Shachar Lovett. An elementary proof of anti-concentration of polynomials in gaussian variables. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 17, pp. 182, 2010.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL `https://www.usenix.org/conference/usenixsecurity20/presentation/mishra`.

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

Ido Nachum and Amir Yehudayoff. On symmetry and initialization for neural networks. In *LATIN 2020*, pp. 401–412, 2020.

Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks. *arXiv preprint arXiv:1902.04674*, 2019.

Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, pp. 11289–11300, 2019.

Warren Schudy and Maxim Sviridenko. Concentration and moment inequalities for polynomials of independent random variables. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pp. 437–446. Society for Industrial and Applied Mathematics, 2012.

Vaishaal Shankar, Alex Fang, Wenshuo Guo, Sara Fridovich-Keil, Ludwig Schmidt, Jonathan Ragan-Kelley, and Benjamin Recht. Neural kernels without tangents. *arXiv preprint arXiv:2003.02237*, 2020.

Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *arXiv preprint arXiv:1707.04926*, 2017.

Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

Matus Telgarsky. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

Yuandong Tian. An analytical formula of population gradient for two-layered relu network and its applications in convergence and critical point analysis. *arXiv preprint arXiv:1703.00560*, 2017.

Loc Quang Trinh. Greedy layerwise training of convolutional neural networks. Master's thesis, Massachusetts Institute of Technology, 2019.

Santosh Vempala and John Wilmes. Polynomial convergence of gradient descent for training one-hidden-layer neural networks. *arXiv preprint arXiv:1805.02677*, 2018.

Bo Xie, Yingyu Liang, and Le Song. Diversity leads to generalization in neural networks. *arXiv preprint Arxiv:1611.03131*, 2016.

Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.

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

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

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

Xiao Zhang, Yaodong Yu, Lingxiao Wang, and Quanquan Gu. Learning one-hidden-layer relu networks via gradient descent. *arXiv preprint arXiv:1806.07808*, 2018.

Kai Zhong, Zhao Song, Prateek Jain, Peter L Bartlett, and Inderjit S Dhillon. Recovery guarantees for one-hidden-layer neural networks. *arXiv preprint arXiv:1706.03175*, 2017.

Difan Zou and Quanquan Gu. An improved analysis of training over-parameterized deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2053–2062, 2019.

Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic gradient descent optimizes over-parameterized deep relu networks. *arXiv preprint arXiv:1811.08888*, 2018.