Proving the Lottery Ticket Hypothesis: Pruning is All You Need

Eran Malach [*1] Gilad Yehudai [*2] Shai Shalev-shwartz [1] Ohad Shamir [2]

Abstract

The lottery ticket hypothesis (Frankle and Carbin, 2018), states that a randomly-initialized network contains a small subnetwork such that, when trained in isolation, can compete with the performance of the original network. We prove an even stronger hypothesis (as was also conjectured in Ramanujan et al., 2019), showing that for every bounded distribution and every target network with bounded weights, a sufficiently over-parameterized neural network with random weights contains a subnetwork with roughly the same accuracy as the target network, without any further training.

1. Introduction

Neural network pruning is a popular method to reduce the size of a trained model, allowing efficient computation during inference time, with minimal loss in accuracy. However, such a method still requires the process of training an over-parameterized network, as training a pruned network from scratch seems to fail (see (Frankle and Carbin, 2018)). Recently, a work by Frankle and Carbin (2018) has presented a surprising phenomenon: pruned neural networks can be trained to achieve good performance, when resetting their weights to their initial values. Hence, the authors state *the lottery ticket hypothesis*: a randomly-initialized neural network contains a subnetwork such that, when trained in isolation, can match the performance of the original network.

This observation has attracted great interest, with various follow-up works trying to understand this intriguing phenomenon. Specifically, very recent works by Zhou et al. (2019); Ramanujan et al. (2019) presented algorithms to find subnetworks that already achieve good performance, without any training. (Ramanujan et al., 2019) stated the following conjecture: a sufficiently over-parameterized neural network with random initialization contains a subnetwork that achieves competitive accuracy (with respect to the large trained network), without any training. This conjecture can be viewed as a stronger version of the lottery ticket hypothesis.

In this work, we prove this stronger conjecture, in the case of over-parameterized neural networks. Moreover, we differentiate between two types of subnetworks: subnetworks where specific weights are removed (*weight-subnetworks*) and subnetworks where entire neurons are removed (*neuron-subnetworks*). First, we show that a ReLU network of arbitrary depth l can be approximated by finding a *weight-subnetwork* of a random network of depth $2l$ and sufficient width. Second, we show that depth-two (one hidden-layer) networks have *neuron-subnetworks* that are competitive with the best random-features classifier (i.e. the best classifier achieved when training only the second layer of the network). Hence, we imply that for shallow networks, training the second layer of the network is equivalent to pruning entire neurons of a sufficiently large random network. In all our results, the size of initial network is polynomial in the problem parameters. In the case of the *weight-subnetwork*, we show that the number of parameters in the pruned network is similar, up to a constant factor, to the number of parameters in the target network.

As far as we are aware, this is the first work that gives theoretical evidence to the existence of good subnetworks within a randomly initialized neural network (i.e., proving the strong lottery ticket hypothesis). Our results imply that fundamentally, pruning a randomly initialized network is as strong as optimizing the value of the weights. Hence, while the common method for finding a good network is to train its parameters, our work demonstrates that in fact, all you need is a good pruning mechanism. This gives a strong motivation to develop algorithms that focus on pruning the weights rather than optimizing their values.

1.1. Related Work

Neural Network Pruning Pruning neural networks is a popular method to compress large models, allowing them to run on devices with limited resources. Over the years, a

[*]Equal contribution [1]School of Computer Science, Hebrew University [2]Weizmann Institute of Science. Correspondence to: Eran Malach <eran.malach@mail.huji.ac.il>, Gilad Yehudai <gilad.yehudai@weizmann.ac.il>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

variety of pruning methods were suggested, showing that the number of parameters neural network models can be reduced by up to 90%, with minimal performance loss. These methods differ in two aspects: how to prune (the pruning criterion), and what to prune (specific weights vs. entire neurons or convolutional channels). Works by LeCun et al. (1990); Hassibi and Stork (1993); Dong et al. (2017) explored the efficiency of network pruning based on second derivative conditions. Another popular method is pruning based on the magnitude of the weights (Han et al., 2015). Other pruning techniques remove neurons with zero activation (Hu et al., 2016) (i.e., neurons that are always output zero), or other measures of redundancy (Mariet and Sra, 2015; Srinivas and Babu, 2015). While weight-based pruning achieves the best results in terms of network compression, the gain in terms of inference time is not optimal, as it cannot be efficiently utilized by modern hardware. To get an effective gain in performance, recent works suggested methods to prune entire neurons or convolutional channels (Yang et al., 2017; Li et al., 2016; Molchanov et al., 2017; Luo et al., 2017).

In our work, we show that surprisingly, pruning a random network achieves results that are competitive with optimizing the weights. Furthermore, we compare neuron-based pruning to weight-based pruning, and show that the latter can achieve strictly stronger performance. We are unaware of any theoretical work studying the power and limitation of such pruning methods.

Lottery Ticket Hypothesis In (Frankle and Carbin, 2018), Frankle and Carbin stated the original **lottery ticket hypothesis**: *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that — when trained in isolation — it can match the test accuracy of the original network after training for at most the same number of iterations*. This conjecture, if it is true, has rather promising practical implications - it suggests that the inefficient process of training a large network is in fact unnecessary, as one only needs to find a good small subnetwork, and then train it separately. While finding a good subnetwork is not trivial, it might still be simpler than training a neural network with millions of parameters.

A follow up work by Zhou et al. (2019) claims that the "winning-tickets", i.e., the good initial subnetwork, already has better-than-random performance on the data, without any training. With this in mind, they suggest an algorithm to find a good subnetwork within a randomly initialized network that achieves good accuracy. Building upon this work, another work by Ramanujan et al. (2019) suggests an improved algorithm which finds an untrained subnetwork that approaches state-of-the-art performance, for various architectures and datasets. Following these observations, (Ramanujan et al., 2019) suggested a complementary conjceture to the original lottery ticket hypothesis: *within a sufficiently overparameterized neural network with random weights (e.g. at initialization), there exists a subnetwork that achieves competitive accuracy*.

While these results raise very intriguing claims, they are all based on empirical observations alone. Our work aims to give theoretical evidence to these empirical results. We prove the latter conjecture, stated in (Ramanujan et al., 2019), in the case of deep and shallow neural networks. To the best of our knowledge, this is the first theoretical work aiming to explain the strong lottery ticket conjecture, as stated in (Ramanujan et al., 2019).

Over-parameterization and random features A popular recent line of works showed how gradient methods over highly over-parameterized neural networks can learn various target functions in polynomial time (e.g. (Allen-Zhu et al., 2019),(Daniely, 2017),(Arora et al., 2019),(Cao and Gu, 2019)). However, recent works (e.g. (Yehudai and Shamir, 2019), (Ghorbani et al., 2019), (Ghorbani et al., 2019)) show the limitations of the analysis in the above approach, and compare the power of the analysis to that of random features. In particular, (Yehudai and Shamir, 2019) show that this approach cannot efficiently approximate a single ReLU neuron, even if the distribution is standard Gaussian. In this work we show that finding a shallow *neuron-subnetwork* is equivalent to learning with random features, and that *weight-subnetworks* is a *strictly* stronger model in the sense that it can efficiently approximate ReLU neurons, under mild assumptions on the distribution (namely, that it is bounded).

1.2. Notations

We introduce some notations that will be used in the sequel. We denote by $\mathcal{X} = \{x \in \mathbb{R}^d : \|x\|_2 \le 1\}$ [1] our instance space. For a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, we denote the squared-loss of a hypothesis $h : \mathcal{X} \to \mathbb{R}$ by:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[(h(x) - y)^2\right] .$$

For two matrices $A, B \in \mathbb{R}^{m\times n}$, we denote by $A \odot B$ the $m \times n$ matrix which its i, j-th coordinate is equal to $[A_{i,j}B_{i,j}]$ the Hadamard (element-wise) product between A and B. We use $U([-c, c]^k)$ to denote the uniform distribution on some cube around zero, and by $\mathcal{N}(0, \Sigma)$ a normal distribution with mean zero and covariance matrix Σ. For a matrix H we denote by $\lambda_{\min}(H)$ its minimal eigenvalue. For a matrix A, we denote by $\|A\|_2$ the L_2 operator norm of A, namely $\|A\|_2 := \lambda_{\max}(A)$ where $\lambda_{\max}$ is the largest singular value of A. We denote by $\|A\|_{\max}$ the max norm of A, namely $\|A\|_{\max} := \max_{i,j} |A_{i,j}|$.

[1] The assumption that $\|x\| \le 1$ is made for simplicity. It can be readily extended to $\|x\| \le r$ for any r at the cost of having the network size depend polynomially on r.

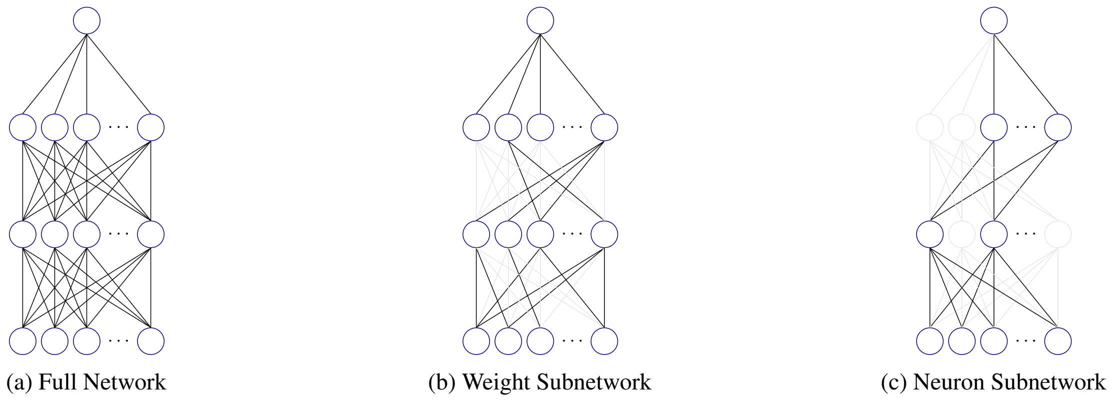

Figure 1. Different types of pruned subnetworks (weight/neuron subnetwork).

2. Approximating ReLU Networks by Pruning Weights

In this section we provide our main result, showing that a network of depth l can be approximated by pruning a random network of depth $2l$. We show this for a setting where we are allowed to prune specific weights, and are not limited to removing entire neurons (i.e. finding *weight-subnetworks*). Neuron-subnetworks are discussed in the next section. We further focus on networks with the ReLU activation, $\sigma(x) = \max\{x, 0\}$. We define a network $G : \mathbb{R}^d \to \mathbb{R}$ of depth l and width[2] n in the following way:

$$G(x) = G^{(l)} \circ \cdots \circ G^{(1)}(x)$$

Where we have:

- $G^{(1)}(x) = \sigma(W^{G(1)}x)$ for $W^{G(1)} \in \mathbb{R}^{d\times n}$.
- $G^{(i)}(x) = \sigma(W^{G(i)}x)$ for $W^{G(i)} \in \mathbb{R}^{n\times n}$, for every $1 < i < l$.
- $G^{(l)}(x) = W^{G(l)}x$ for $W^{G(l)} \in \mathbb{R}^{n\times 1}$

where $W^{G(i)}$ are the weights in the i-th layer. A **weight-subnetwork** $\widetilde{G}$ of G is a network of width n and depth l, with weights $W^{\widetilde{G}(i)} := B^{(i)} \odot W^{G(i)}$ for some mask $B^{(i)} \in \{0,1\}^{n_{in}\times n_{out}}$ (where n_{in}, n_{out} denote the input/output dimension of each layer, respectively). Our main theorem in this section shows that for every target network of depth l with bounded weights, a random network of depth $2l$ and polynomial width contains with high probability (over the randomness of the weights) a subnetwork that approximates the target network:

Theorem 2.1. *Fix some $\epsilon, \delta \in (0,1)$. Let F be some target network of depth l such that for every $i \in [l]$ we have $\|W^{F(i)}\|_2 \le 1, \|W^{F(i)}\|_{\max} \le \frac{1}{\sqrt{n_{in}}}$ (where $n_{in} = d$ for $i = 1$ and $n_{in} = n$ for $i > 1$). Let G be a network of width $\text{poly}(d, n, l, \frac{1}{\epsilon}, \log\frac{1}{\delta})$ and depth $2l$, where we initialize $W^{G(i)}$ from $U([-1,1])$. Then, w.p at least $1-\delta$ over the initialization of G there exists a weight-subnetwork $\widetilde{G}$ of G such that:*

$$\sup_{x\in\mathcal{X}} \left|\widetilde{G}(x) - F(x)\right| \le \epsilon$$

Furthermore, the number of active (non-zero) weights in $\widetilde{G}$ is $O(dn + n^2 l)$.

Remark 2.2. *We note that the initialization scheme of the network considered in Thm. 2.1 is not standard Xavier initialization. The reason is that in standard Xavier initialization the weights are normalized such that the gradient's variance at initialization will not depend on the network's size. Here we don't calculate the gradient but only prune some of the neurons. Thus, the magnitude of the weights does not depend on the width of the network. That said, the theorem can be easily extended to any initialization which is a uniform distribution on some interval around zero, by correctly scaling the network's output.*

Since the number of parameters in the function F is $dn + n^2(l-2) + n$, the above shows that the number of active weights in the pruned network is similar, up to a constant factor, to the number of parameters in F. Note that the width of the random network has polynomial dependence on the input dimension d, the width of the target network n and its depth l. While the dependence on the width and depth of the target network is unavoidable, the dependence on the input dimension may seem to somewhat weaken the result. Since neural networks are often used on high-dimensional inputs, such dependence on the input dimension might make our result problematic for practical settings in the high-dimension regimes. However, we note that such dependence could be avoided, when making some additional assumptions on the target network. Specifically,

[2]We consider all layers of the network to be of the same width for convenience of notations. Our results can easily be extended to cases where the size of the layers differ.

if we assume that the target network has sparse weights in the first layer, i.e. - each neuron in the first layer has at most s non-zero weights, then we get dependence on the sparsity s, rather than on the input dimension d. This is shown formally in the appendix.

We can derive a slightly stronger result in the case where the target network is a depth-two network. Specifically, we can show that a depth-two network can be approximated by pruning a depth-three random network (rather than pruning a depth-four network, as implied from Thm. 2.1):

Theorem 2.3. *Fix some target two-layer neural network F of width n, and fix $\epsilon, \delta \in (0,1)$. Let G be a random three-layer neural network of width $\mathrm{poly}\left(d, n, \frac{1}{\epsilon}, \log\left(\frac{1}{\delta}\right)\right)$, with weights initialized from $U([-1,1])$. Then, with probability at least $1-\delta$ over the initialization of G, there exists a weight-subnetwork $\widetilde{G}$ of G, such that:*

$$\sup_{x \in \mathcal{X}} \left| F(x) - \widetilde{G}(x) \right| \le \epsilon$$

Furthermore, the number of active (non-zero) weights in $\widetilde{G}$ is $O(dn)$.

2.1. Proof intuition

The full proof of Thm. 2.1 and Thm. 2.3 can be found in Appendix A. We start by giving a sketch of the main argument.

The basic building block of the proof is showing how to approximate a single ReLU neuron of the form $x \mapsto \sigma(\langle w^*, x \rangle)$ by a two layer network. Using the equality $a = \sigma(a) - \sigma(-a)$, we can write the neuron as:

$$\begin{aligned} x \mapsto \sigma\left(\sum_{i=1}^{d} w_i^* x_i\right) \\ = \sigma\left(\sum_{i=1}^{d} \sigma(w_i^* x_i) - \sum_{i=1}^{d} \sigma(-w_i^* x_i)\right). \end{aligned} \tag{1}$$

Now, consider a two layer network of width k and a single output neuron, with a pruning matrix B for the first layer. It can be written as

$$x \mapsto \sigma\left(\sum_{j=1}^{k} u_j \sigma\left(\sum_{t=1}^{d} B_{j,t} W_{j,t} x_t\right)\right).$$

Suppose we pick, for every i, two indexes $j_1(i), j_2(i)$, and set the matrix B s.t. $B_{j_1(i),i}, B_{j_2(i),i} = 1$ and all the rest of the elements of B are zero. It follows that the pruned network can be rewritten as

$$\begin{aligned} x \mapsto \sigma\left(\sum_{i=1}^{d} u_{j_1(i)} \sigma(W_{j_1(i),i} x_i) + \sum_{i=1}^{d} u_{j_2(i)} \sigma(W_{j_2(i),i} x_i)\right) \\ = \sigma\Bigg(\sum_{i=1}^{d} \mathrm{sign}(u_{j_1(i)}) \sigma(\left|u_{j_1(i)}\right| W_{j_1(i),i} x_i) + \\ + \sum_{i=1}^{d} \mathrm{sign}(u_{j_2(i)}) \sigma(\left|u_{j_2(i)}\right| W_{j_2(i),i} x_i)\Bigg) \end{aligned} \tag{2}$$

Comparing the right-hand sides of Equations 1 and 2, we observe that they will be at most ϵ away from each other provided that for every i:

- $\mathrm{sign}(u_{j_1(i)}) \neq \mathrm{sign}(u_{j_2(i)})$
- $\left| \left|u_{j_1(i)}\right| W_{j_1(i),i} - \mathrm{sign}(u_{j_1(i)}) w_i^* \right| \le \epsilon/2d$
- $\left| \left|u_{j_2(i)}\right| W_{j_2(i),i} - \mathrm{sign}(u_{j_2(i)}) w_i^* \right| \le \epsilon/2d$

Finally, fixing i and picking $u_{j_1(i)}, u_{j_2(i)}, W_{j_1(i),i}, W_{j_2(i),i}$ at random, the requirements would be fulfilled with probability of $\Omega(\epsilon/d)$. Hence, if $k \gg d/\epsilon$, for every i we would be able to find an appropriate $j_1(i), j_2(i)$ with high probability. Note that the number of weights we are actually using is $2d$, which is only factor of 2 larger than the number of original weights required to express a single neuron.

The construction above can be easily extended to show how a depth two ReLU network can approximate a single ReLU layer (simply apply the construction for every neuron). By stacking the approximations, we obtain an approximation of a full network. Since every layer in the original network requires two layers in the newly constructed pruned network, we require a twice deeper network than the original one.

2.2. Proof of Thm. 2.3

Now, we go over the details of the proof of Thm. 2.3. The proof of Thm. 2.1 is a slightly more involved version of this proof. The proofs of the lemmas are given in the appendix. Note that in the appendix the proofs are given in a more general manner, where we consider a sparsity parameter $s \le d$ such that every neuron in first layer of the target network has at most s non-zero weights. Here we show the proof for the simpler case of $s = d$.

First we show that it is possible to approximate the function $x \mapsto w_i^* x_i$, that is, approximating a projection on a single coordinate:

Lemma 2.4. *Fix some scalar $\alpha \in [-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}]$, index $i \in [d]$, and some $\epsilon, \delta > 0$. Let $w^{(1)}, \ldots, w^{(k)} \in \mathbb{R}^d$ chosen randomly from $U([-1,1]^d)$, and $u^{(1)}, \ldots, u^{(k)} \in$*

$[-1,1]$ *chosen randomly from* $U([-1,1])$. *Then, for* $k \geq O\left(\frac{1}{\epsilon^2}, \log\left(\frac{1}{\delta}\right)\right)$, *w.p at least* $1-\delta$ *there exists a binary mask* $b^{(1)}, \dots, b^{(k)} \in \{0,1\}^d$, *such that* $g(x) = \sum_j u^{(j)}\sigma(\langle w^{(j)} \odot b^{(j)}, x\rangle)$ *satisfies* $|g(x) - \alpha x_i| \leq 2\epsilon$, *for* $\|x\|_\infty \leq 1$. *Furthermore, we have* $\sum_j \|b^{(j)}\|_0 \leq 2$ *and* $\max_j \|b^{(j)}\|_0 \leq 1$.

The binary mask $b^{(j)}$ zeroes out all the weights, except for two single weights which are close to w_i^* and $-w_i^*$. By initializing the network with sufficient width we are guaranteed to have such weights w.h.p.

The next step is to approximate the linear function $x \mapsto \langle w^*, x\rangle = \sum_{i=1}^d w_i^* x_i$. This can be done by approximating the projection on the i-th coordinate d times:

Lemma 2.5. *Fix some* $w^* \in [-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}]^d$ *and some* $\epsilon, \delta > 0$. *Let* $w^{(1)}, \dots, w^{(k)} \in \mathbb{R}^d$ *chosen randomly from* $U([-1,1]^d)$, *and* $u \in [-1,1]^k$ *chosen randomly from* $U([-1,1]^k)$. *Then, for* $k = O\left(d^3, \frac{1}{\epsilon^2}, \log\left(\frac{1}{\delta}\right)\right)$, *w.p at least* $1-\delta$ *there exists a binary mask* $b^{(1)}, \dots, b^{(k)} \in \{0,1\}^d$, *such that* $g(x) = \sum_{i=1}^k u_i \sigma(\langle w^{(i)} \odot b^{(i)}, x\rangle)$ *satisfies* $|g(x) - \langle w^*, x\rangle| \leq \epsilon$, *for* $\|x\|_\infty \leq 1$. *Furthermore, we have* $\sum_i \|b^{(i)}\|_0 \leq 2d$ *and* $\max_i \|b^{(i)}\|_0 \leq 1$.

As in the previous lemma, for each coordinate the binary mask zeroes out every weight except two specifically chosen weights. In total we have only $2d$ non-zero weights.

Finally, we can approximate a single neuron $x \mapsto \sigma(\langle w^*, x\rangle)$ using a three layer network:

Lemma 2.6. *Fix some* $w^* \in [-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}]^d$, *some* $v^* \in [-1,1]$ *and* $\epsilon, \delta > 0$. *Let* $w^{(1)}, \dots, w^{(k_1)} \in \mathbb{R}^d$ *chosen randomly from* $U([-1,1]^d)$, $u^{(1)}, \dots, u^{(k_2)} \in [-1,1]^{k_1}$ *chosen randomly from* $U([-1,1]^{k_1})$, *and* $v \in [-1,1]^{k_2}$ *chosen randomly from* $U([-1,1]^{k_2})$. *Then, for* $k_1 = O\left(d^3, \frac{1}{\epsilon^2}, \log\left(\frac{1}{\delta}\right)\right)$, $k_2 = O\left(\frac{1}{\epsilon}, \log\left(\frac{1}{\delta}\right)\right)$, *w.p at least* $1-\delta$ *there exists a binary mask* $b^{(1)}, \dots, b^{(k_1)} \in \{0,1\}^d$, $\hat{b} \in \{0,1\}^{k_2}$, *such that* $g(x) = \sum_{i=1}^{k_2} \hat{b}_i v_i \sigma(\sum_{j=1}^{k_1} u_j^{(i)} \sigma(\langle w^{(j)} \odot b^{(j)}, x\rangle))$ *satisfies* $|g(x) - v^*\sigma(\langle w^*, x\rangle)| \leq \epsilon$, *for* $\|x\|_2 \leq 1$. *Furthermore, we have* $\sum_j \|b^{(j)}\|_0 \leq 2d$ *and* $\max_j \|b^{(j)}\|_0 \leq 1$.

The idea here is that we can use the third layer to initialize k_2 networks, each of width k_1. By choosing k_1 and k_2 large enough, w.h.p. we initialized a network of width k_1 such that its weight in the third layer is close the coefficient multiplying the target neuron. Then the binary mask zeroes out all other weights in the network such that the only weights remaining approximate our target neuron.

Using the above lemmas, we can prove Thm. 2.3. Note that the key argument here is simple: if a weight-subnetwork can approximate a single neuron with probability at least $1 - \frac{\delta}{n}$, then using the union bound, stacking n such subnetworks approximates all neurons with probability at least $1 - \delta$.

Proof. of Thm. 2.3.

Fix some $w^{(1)*}, \dots, w^{(n)*} \in [-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}]^d$, $v^* \in [-1,1]^n$ and let $f(x) = \sum_{i=1}^n v_i^* \sigma(\langle w^{(i)*}, x\rangle)$. Fix some $\epsilon, \delta > 0$. Let $w^{(1)}, \dots, w^{(k_1)} \in \mathbb{R}^d$ chosen randomly from $U([-1,1]^d)$, $u^{(1)}, \dots, u^{(k_2)} \in [-1,1]^{k_1}$ chosen randomly from $U([-1,1]^{k_1})$, and $v \in [-1,1]^{k_2}$ chosen randomly from $U([-1,1]^{k_2})$. So, our pruned network is defined by:

$$g(x) = \sum_{i=1}^{k_2} \hat{b}_i v_i \sigma(\sum_{j=1}^{k_1} \tilde{b}_j^{(i)} u_j^{(i)} \sigma(\langle w^{(j)} \odot b^{(j)}, x\rangle))$$

With masks $b^{(i)} \in \{0,1\}^d, \tilde{b}^{(i)} \in \{0,1\}^{k_1}, \hat{b} \in \{0,1\}^{k_2}$.

Denote $k_1' = \frac{k_1}{n}, k_2' = \frac{k_2}{n}$ and assume $k_1', k_2' \in \mathbb{N}$ (otherwise mask exceeding neurons). With slight abuse of notation, we denote $w^{(i,j)} := w^{(j+k_1'i)}$, $u^{(i,j)} := \left(u_{ik_1'}^{(j+ik_2')}, \dots, u_{(i+1)k_1'}^{(j+ik_2')}\right)$, $v^{(i,j)} := v_{j+ik_2'}$ and similarly $b^{(i,j)} := b^{(j+k_1'i)}$, $\tilde{b}^{(i,j)} = \left(\tilde{b}_{ik_1'}^{(j+ik_2')}, \dots, \tilde{b}_{(i+1)k_1'}^{(j+ik_2')}\right)$ and $\hat{b}^{(i,j)} = \hat{b}_{j+ik_2'}$. Define for every $i \in [n]$:

$$g_i(x) =$$
$$\sum_j \hat{b}^{(i,j)} v^{(i,j)} \sigma\left(\sum_l \tilde{b}_l^{(i,j)} u_l^{(i,j)} \sigma\left(\langle b^{(i,l)} \circ w^{(i,l)}, x\rangle\right)\right)$$

Now, by setting $\tilde{b}_l^{(j+k_1'i)} = 1$ if $ik_1' \leq l < (i+1)k_1'$ and $\tilde{b}_l^{(j+k_1'i)} = 0$ otherwise, we get that $g(x) = \sum_{i=1}^n g_i(x)$. From Lemma 2.6 we get that for $k_1' = O\left(d^3, \frac{n^2}{\epsilon^2} \log\left(\frac{n}{\delta}\right)\right)$ and $k_2' = O\left(\frac{n}{\epsilon}, \log\left(\frac{n}{\delta}\right)\right)$ with probability at least $1 - \frac{\delta}{n}$ we have $\left|g_i(x) - v_i^* \sigma(\langle w^{(i)*}, x\rangle)\right| \leq \frac{\epsilon}{n}$ for every $\|x\|_2 \leq 1$. Using the union bound, we get that with probability at least $1-\delta$, for $\|x\|_2 \leq 1$ we have:

$$|g(x) - f(x)| \leq \sum_{i=1}^n \left|g_i(x) - v_i^* \sigma(\langle w^{(i)*}, x\rangle)\right| \leq \epsilon$$

Finally, the number of active weights in the first layer is simply: $\sum_{i=1}^n \sum_{j=1}^{k_1'} \|b^{(j)}\|_0 \leq 2nd$. The number of weights in the final layer is at most the same as the first layer, and hence we get that the number of non-zero weights is $O(nd)$. $\square$

2.3. Universality and Computational Efficiency of Pruning

We showed that in terms of expressive power, pruning weights in a randomly initialized over-parameterized network can approximate a target ReLU network of any depth. Using well-known results in the literature of neural networks, this result implies two interesting corollaries:

Universal Approximation Using Weight Pruning It has been long known that neural networks are universal approximators: they are able to approximate any continuous function up to arbitrary accuracy (for example, see (Stinchcombe and White, 1989; Scarselli and Tsoi, 1998)). Since we show that pruning a network with random weights can approximate any target network, this implies that pruning a random network is also a universal approximation scheme. That is, for every smooth function f, a large enough random network contains with high probability (over the randomness of the weights) a subnetwork that approximates f up to accuracy ϵ.

Pruning Weights is Computationally Hard It is well known in the literature of neural networks that learning even a depth-two ReLU network is computationally hard in the general case (see (Livni et al., 2014; Manurangsi and Reichman, 2018; Boob et al., 2018)). Such results show cases where it is computationally hard to find any hypothesis that is competitive with the best neural-network on the given data. From these results, it is immediate that weight-pruning of random ReLU networks, deep or shallow, is computationally hard as well. That is, no polynomial-time algorithm can find the optimal subnetwork of some given neural network Indeed, if we had an efficient algorithm that finds an optimal weight-subnetwork of a three-layer network, from Thm. 2.3 this algorithm approximates the best depth-two network (for some fixed width). But in general, approximating the best depth-two network on an arbitrary distribution is computationally hard (under certain hardness assumptions), which leads to a contradiction. So, there is no efficient algorithm that is guaranteed to return an optimal weight-subnetwork for any input distribution.

3. Equivalence Between Pruning Neurons and Random Features

In this section we analyze the power of pruning entire neurons in a depth-two network. The main result of this section is that pruning entire neurons is equivalent to the well known random features model (e.g. (Rahimi and Recht, 2008), (Rahimi and Recht, 2008)). Intuitively, we show that whenever training only the last layer of the network suffices, it is also possible to construct a good sub-network by pruning entire neurons. By recent results regarding the limitations of the random features model, we establish that neuron pruning is also a very limited model, in comparison to the model of weight pruning discussed in the previous section.

Formally, consider a width k two-layer neural network defined by $g : \mathbb{R}^d \to \mathbb{R}$ as follows:

$$g(x) = u^\top \sigma(Wx) = \sum_{i=1}^{k} u_i \sigma(\langle w_i, x\rangle)$$

where u_i is the i-th coordinate of u and w_i is the i-th row of W. A network $\tilde{g}$ is a **neuron-subnetwork** of g if there exists a vector $b \in \{0,1\}^k$ such that:

$$\tilde{g}(x) = (u \odot b)^\top \sigma(Wx) = \sum_{i=1}^{k} (u_i \cdot b_i)\sigma(\langle w_i, x\rangle).$$

So, $\tilde{g}$ is also a 2-layer neural network, which contains a subset of the neuron of g. Next, we define the random features model:

Definition 3.1. *Suppose we sample $w_1, \dots, w_k \sim D$ from some distribution D, a* ***random features model*** *over $w_1, \dots, w_k$ and activation σ is any function of the form:*

$$f(x) = \sum_{i=1}^{k} u_i \sigma(\langle w_i, x\rangle)$$

for $u_1, \dots, u_k \in \mathbb{R}$.

Training a 2-layer random features model is done by training only the second layer, i.e. training only the weights $u_1, \dots, u_k$. This is equivalent to training a linear model over the features $\sigma(\langle w_i, x\rangle)$, which are chosen randomly. We show that neuron-subnetworks are competitive with random features:

Theorem 3.2. *Let D be any distribution over $\mathcal{X} \times [-1, +1]$, and let $\sigma : \mathbb{R} \to \mathbb{R}$ be L-Lipschitz with $\sigma(0) \le L$. Let $\epsilon, \delta > 0$, $n \in \mathbb{N}$ and D^* a distribution over $\{w : \|w\| \le 1\}$ such that for $w_1, \dots, w_n \sim D^*$ w.p $> 1 - \delta$ there exist $u_1, \dots, u_n \in \mathbb{R}$ such that $|u_i| \le C$ and the function $f(x) = \sum_{i=1}^{n} u_i \sigma(\langle w_i, x\rangle)$ satisfies that $L_D(f) \le \epsilon$. Let $k \ge poly\left(C, n, L, \frac{1}{\epsilon}, \frac{1}{\delta}\right)$, and suppose we initialize a 2-layer neural network g with width k where $w_i \sim D^*$, and $u_i \sim U([-1, 1])$. Then there exists a neuron-subnetwork $\tilde{g}$ of g and constant $c > 0$ such that $L_D(c\tilde{g}) \le \epsilon$.*

The full proof can be found in Appendix B. Thm. 3.2 shows that for any distribution over the data, if a random features model can achieve small loss, then it is also possible to find a neuron-subnetwork of a randomly initialized network (with enough width) that achieves the same loss. This means that pruning neurons is competitive with the random features model. On the other hand, if for some distribution over the data it is possible to find a neuron-subnetwork of a randomly initialized network that achieves small loss, then clearly it is possible to find a random features model that achieves the same loss. Indeed, we can set the weights of the random features model to be the same as in the neuron-subnetwork, where pruned weights are equal to zero.

To summarize, Thm. 3.2 and the argument above shows an equivalence between random features and neuron-subnetworks: For a distribution D, there is a random features model f with k features such that $L_D(f) \le \epsilon$ if-and-only-if for a randomly initialized network with width polynomial in k, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$, w.p $> 1-\delta$ there exists a neuron-subnetwork $\tilde{g}$ such that $L_D(\tilde{g}) \leq \epsilon$.

A few recent works (e.g. (Yehudai and Shamir, 2019), (Ghorbani et al., 2019), (Ghorbani et al., 2019)) studied the limitations of random features. In particular, (Yehudai and Shamir, 2019) show that a random features model cannot approximate a single ReLU neuron even under standard Gaussian distribution, unless the amount of features or the magnitude of the weights (or both) are exponential in the input dimension. Thus, the above equivalence also shows a limitation of neuron-subnetworks - they cannot efficiently approximate a single ReLU neuron, just as random features can't. This means that the weight-subnetwork model shown in Sec. 2 is significantly stronger than the neuron-subnetwork model.

The intuition behind the proof of Thm. 3.2 is the following: Assume we initialize a 2-layer neural network of width $n = k \cdot m$ where k is as in the theorem, and m is some large number (that depends on $\frac{1}{\epsilon}, \frac{1}{\delta}$). We think of it as initializing m different 2-layer networks of width k. From the assumption of the theorem, for most of these networks there exists a random features model that achieves small loss. For each of these networks we prune a neuron if its randomly initialized weight in the second layer is far from its corresponding random features model's weight. Note that since we initialize the weights i.i.d., then we prune each neuron with the same probability and independently of the other neurons.

The main lemma in the proof of Thm. 3.2 is the following:

Lemma 3.3. *Let $k_1 \in \mathbb{N}$, $\epsilon, \delta > 0$ and assume that σ is L-Lipschitz with $\sigma(0) \leq L$. Let $k_2 > O\left(k_1^4, L^4, \frac{1}{\epsilon^4}, \log\left(\frac{k_1}{\delta}\right)\right)$, and for every $i \in [k_1]$, $j \in [k_2]$ initialize $w_i^{(j)} \sim \mathcal{D}$ for any distribution $\mathcal{D}$ with $\mathbb{P}(\|w_i\| \leq 1) = 1$ and $u_i^{(j)} \sim U([-1,1])$. Let $v^{(1)}, \dots, v^{(k_2)} \in \mathbb{R}^{k_1}$ with $\|v^{(j)}\|_\infty \leq 1$ for every $j \in [k_2]$, and define $f^{(j)}(x) = \sum_{i=1}^{k_1} v_i^{(j)} \sigma\left(\langle w_i^{(j)}, x\rangle\right)$. Then there exist $b^{(1)}, \dots, b^{(k_2)} \in \{0,1\}^{k_1}$ such that for the functions $\tilde{g}^{(j)}(x) = \sum_{i=1}^{k_1} b_i^{(j)} \cdot u_i^{(j)} \sigma\left(\langle w_i^{(j)}, x\rangle\right)$ w.p $> 1-\delta$ we have:*

$$\sup_{x:\|x\|\leq 1} \left| c_1 \cdot \frac{1}{k_2} \sum_{j=1}^{k_2} \tilde{g}^{(j)}(x) - \frac{1}{k_2} \sum_{j=1}^{k_2} f^{(j)}(x) \right| \leq \epsilon$$

where $c_1 > 0$

Intuitively, we are given k_2 neural networks, each of width k_1. The lemma shows that if we randomly initialize the second layer of each of the k_2 networks, and zero out the relevant neuron if it is too far away from the target neuron, then by averaging over these k_2 networks, we have a good approximation of the target network. In the proof we use a concentration of measure argument and the fact that we zero out each neuron i.i.d, because we initialize the weights i.i.d.

3.1. Learning Finite Datasets and RKHS Functions via Neuron-Subnetworks

In this subsection we show that pruning entire neurons may prove beneficial, despite the inherent limitations discussed previously. We focus on two popular families of problems, which are known to be solvable by training depth-two networks:

1. Overfitting a finite sample:

   $$S = \{(x_1, y_1), \dots, (x_m, y_m) \in \mathcal{X} \times [-1,1]\}.$$

   This is equivalent to finding a neuron-subnetwork which minimizes the empirical risk on the sample S. This setting is considered in various recent works (for example in (Du et al., 2018), (Du et al., 2018), (Allen-Zhu et al., 2018)).

2. Learning RKHS: given an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ we consider a target function from the set of functions $\mathcal{F}_C$ which consists of functions of the form

   $$c_d \int_{w \in \left[-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right]^d} h(w)\sigma(\langle w, x\rangle) dw$$

   where $\sup_w |h(w)| \leq C$, and $c_d = \left(\frac{\sqrt{d}}{2}\right)^d$ is a normalization term. The set $\mathcal{F}_\infty$ is actually the RKHS of the kernel

   $$K(x,y) = \mathbb{E}_{w \in U\left(\left[-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right]^d\right)} \left[\sigma(\langle w, x\rangle) \cdot \sigma(w, y)\right].$$

   In particular, for σ which is not a polynomial, the set $\mathcal{F}_\infty$ contains all continuous functions (see (Leshno et al., 1993)). This setting is considered in (Cao and Gu, 2019), (Sun et al., 2018).

The main theorem of this section is the following:

Theorem 3.4. *Let $\epsilon, \delta > 0$ and let $\sigma : \mathbb{R} \to \mathbb{R}$ be L-Lipschitz with $\sigma(0) \leq L$. Let g be a randomly initialized 2-layer neural network of width k such that $w_i \sim U\left(\left[-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right]^d\right)$, and $u_i \sim U([-1,1])$.*

1. *(Finite dataset) Let*

   $$S = \{(x_1, y_1), \dots, (x_m, y_m) \in \mathcal{X} \times [-1,+1]\},$$

   and let H be the $m \times m$ matrix defined by $H_{i,j} = \mathbb{E}_w\left[\sigma(\langle w, x_i\rangle)\sigma(\langle w, x_j\rangle)\right]$ and assume that

$\lambda_{\min}(H) = \lambda > 0$. If $k \geq poly\left(m, \frac{1}{\lambda}, L, log\left(\frac{1}{\delta}\right), \frac{1}{\epsilon}\right)$ then w.p $> 1-\delta$ there exists a neuron-subnetwork $\tilde{g}$ and a constant $c > 0$ such that:

$$\sup_{i=1,\dots,m} |c\tilde{g}(x_i) - y_i| \leq \epsilon$$

2. *(RKHS function) Let $f \in \mathcal{F}_C$. If $k \geq poly\left(C, L, log\left(\frac{1}{\delta}\right), \frac{1}{\epsilon}\right)$ then w.p $> 1-\delta$ there exists a neuron-subnetwork $\tilde{g}$ and a constant $c > 0$ such that:*

$$\sup_{x\in\mathcal{X}} |c\tilde{g}(x) - f(x)| \leq \epsilon$$

Remark 3.5. *For the finite dataset case, the assumption on the minimal eigenvalue λ of the matrix H is standard and assumed in similar forms in other works which approximate a finite dataset using random features approach (see (Du et al., 2018), (Du et al., 2018), (Panigrahi et al., 2019)).*

In both versions of the theorem, the network's width does not depend on the dimension of the input data. It does depend on the "complexity" of the target distribution. In the finite dataset case the network's width depends on the number of examples m and on the value of $\frac{1}{\lambda}$. In the RKHS function case, it depends on the constant C which defines the size of the function class $\mathcal{F}_C$ from which the target function is taken.

Note that in a binary classification task (where that labels are ± 1) over a finite dataset, Thm. 3.4 shows that we can achieve zero loss (with respect to the $0-1$ loss), even if we don't scale $\tilde{g}(x)$ by a constant c. To show this, we use Thm. 3.4 with $\epsilon = 1/2$ to get that for every pair (x, y) in the finite dataset we have $|c\tilde{g}(x) - y| \leq 1/2$, since $c > 0$ and $y \in \{1, -1\}$ we get that $\text{sign}(\tilde{g}(x)) = \text{sign}(y)$.

We give a short proof intuition for Thm. 3.4, the full proof is in appendix C. We initialize a 2-layer neural network of width $k = k_1 \cdot k_2$, this can be thought as initializing k_2 different networks, each of width k_1. The idea is to choose k_1 large enough so that w.h.p. a random features model with k_1 features would be able to approximate the target (either finite dataset or RKHS function). Next, for each network of size k_1 we prune a neuron if it is far from its corresponding random features model. We finish by using a concentration of measure argument to conclude that averaging over k_2 such networks (for a large enough k_2) yields a good approximation of the target.

Remark 3.6. *The proof of Thm. 3.4 actually provides an algorithm for pruning 2-layer neural networks:*

- *Randomly initialize a 2-layer neural network of width $k = k_1 \cdot k_2$.*
- *For each subnetwork of width k_1 - optimize a linear predictor over the random weights from the first layer.*
- *Let ϵ be a confidence parameter, prune each neuron if its distance from the corresponding weight of the trained linear predictor is more than ϵ.*

This algorithm runs in polynomial time, but it is obviously very naive. However, it does demonstrate that there exists a polynomial time algorithm for pruning neurons in shallow networks. We leave a study of more efficient algorithms for future work.

4. Discussion/Future Work

We have shown strong positive results on the expressive power of pruned random networks. However, as we mentioned previously, our results imply that there is no efficient algorithm for weight-pruning of a random network, by reduction from hardness results on learning neural networks. Hence, weight-pruning is similar to weight-optimization in the following sense: in both methods there exists a good solution, but finding it is computationally hard in the worst case. That said, similarly to weight optimization, heuristic algorithms for pruning might work well in practice, as shown in (Zhou et al., 2019; Ramanujan et al., 2019). Furthermore, pruning algorithms may enjoy some advantages over standard weight-optimization algorithms. First, while weight-optimization requires training very large networks and results in large models and inefficient inference, weight-pruning by design achieves networks with preferable inference-time performance. Second, weight-optimization is largely done with gradient-based algorithms, which have been shown to be suboptimal in various cases (see (Shalev-Shwartz et al., 2017; Shamir, 2018)). Pruning algorithms, on the other hand, can possibly rely on very different algorithmic techniques, that might avoid the pitfalls of gradient-descent.

To conclude, in this work we showed some initial motivations for studying algorithms for pruning random networks, which we believe set the ground for numerous future directions. An immediate future research direction is to come up with a heuristic pruning algorithm that works well in practice, and provide provable guarantees under mild distributional assumptions. Other interesting questions for future research include understanding to what extent the polynomial dependencies of the size of the neural network before pruning can be improved, and generalizing the results to other architectures such as convolutional layers and ResNets.

Acknowledgements: This research is supported by the European Research Council (TheoryDL project), and by European Research Council (ERC) grant 754705.

References

[1] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*, 2018.

[2] Z. Allen-Zhu, Y. Li, and Y. Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems*, 2019.

[3] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019.

[4] D. Boob, S. S. Dey, and G. Lan. Complexity of training relu neural network. *arXiv preprint arXiv:1809.10787*, 2018.

[5] Y. Cao and Q. Gu. A generalization theory of gradient descent for learning over-parameterized deep ReLU networks. *arXiv preprint arXiv:1902.01384*, 2019.

[6] A. Daniely. SGD learns the conjugate kernel class of the network. In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2017.

[7] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[8] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018.

[9] S. S. Du, X. Zhai, B. Poczos, and A. Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

[10] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[11] B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari. Limitations of lazy training of two-layers neural networks. *arXiv preprint arXiv:1906.08899*, 2019.

[12] B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari. Linearized two-layers neural networks in high dimension. *arXiv preprint arXiv:1904.12191*, 2019.

[13] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[14] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[15] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[16] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[17] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[18] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[19] R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

[20] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[21] P. Manurangsi and D. Reichman. The computational complexity of training relu (s). *arXiv preprint arXiv:1810.04207*, 2018.

[22] Z. Mariet and S. Sra. Diversity networks: Neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015.

[23] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2498–2507. JMLR. org, 2017.

[24] A. Panigrahi, A. Shetty, and N. Goyal. Effect of activation functions on the training of overparametrized neural nets. *arXiv preprint arXiv:1908.05660*, 2019.

[25] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[26] A. Rahimi and B. Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561. IEEE, 2008.

[27] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? *arXiv preprint arXiv:1911.13299*, 2019.

[28] F. Scarselli and A. C. Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, 11(1):15–37, 1998.

[29] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[30] S. Shalev-Shwartz, O. Shamir, and S. Shammah. Failures of gradient-based deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3067–3075. JMLR. org, 2017.

[31] O. Shamir. Distribution-specific hardness of learning neural networks. *The Journal of Machine Learning Research*, 19(1):1135–1163, 2018.

[32] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[33] M. Stinchcombe and H. White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *IJCNN International Joint Conference on Neural Networks*, 1989.

[34] Y. Sun, A. Gilbert, and A. Tewari. Random ReLU features: Universality, approximation, and composition. *arXiv preprint arXiv:1810.04374*, 2018.

[35] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[36] G. Yehudai and O. Shamir. On the power and limitations of random features for understanding neural networks. In *Advances in Neural Information Processing Systems*, 2019.

[37] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067*, 2019.