Proceedings of Machine Learning Research vol 144:1–33, 2022

How much pre-training is enough to discover a good subnetwork?

Cameron R. Wolfe CRW13@RICE.EDU
Qihan Wang QW15@RICE.EDU
Junhyung Lyle Kim JK77@RICE.EDU
Anastasios Kyrillidis ANASTASIOS@RICE.EDU
Department of Computer Science, Rice University, Houston, TX, USA.

Abstract

Neural network pruning is useful for discovering efficient, high-performing subnetworks within pre-trained, dense network architectures. However, more often than not, it involves a three-step process—pre-training, pruning, and re-training—that is computationally expensive, as the dense model must be fully pre-trained. Luckily, several works have empirically shown that high-performing subnetworks can be discovered via pruning without fully pre-training the dense network. Aiming to theoretically analyze the amount of dense network pre-training needed for a pruned network to perform well, we discover a theoretical bound in the number of SGD pre-training iterations on a two-layer, fully-connected network, beyond which pruning via greedy forward selection (Ye et al., 2020) yields a subnetwork that achieves good training error. This threshold is shown to be logarithmically dependent upon the size of the dataset, meaning that experiments with larger datasets require more pre-training for subnetworks obtained via pruning to perform well. We empirically demonstrate the validity of our theoretical results across a variety of architectures and datasets, including fully-connected networks trained on MNIST and several deep convolutional neural network (CNN) architectures trained on CIFAR10 and ImageNet.

Keywords: Lottery Ticket Hypothesis, Early Bird Tickets, Greedy Selection

1. Introduction

The proposal of the Lottery Ticket Hypothesis (LTH) (Frankle and Carbin, 2018) has led to significant interest in using pruning techniques to discover small (sometimes sparse) models that perform well. LTH has been empirically validated and is even applicable in large-scale settings (Chen et al., 2020; Frankle et al., 2019; Gale et al., 2019; Liu et al., 2018; Morcos et al., 2019; Zhou et al., 2019; Zhu and Gupta, 2017), revealing that high-performing subnetworks can be obtained by pruning dense, pre-trained models and fine-tuning the pruned weights to convergence (i.e., either from their initial values or some later point) (Chen et al., 2020a; Girish et al., 2020; Renda et al., 2020).

Neural network pruning tends to follow a three step process, including pre-training, pruning, and re-training, where pre-training is the most costly step (Frankle and Carbin, 2018; You et al., 2019). To circumvent the cost of pre-training, several works explore the possibility of pruning networks directly from initialization (i.e., the "strong lottery ticket hypothesis") (Frankle et al., 2020; Ramanujan et al., 2019; Wang et al., 2020), but subnetwork performance could suffer. Adopting a hybrid approach, good subnetworks can also be obtained from models with minimal pre-training (Chen et al., 2020b; You et al., 2019) (i.e., "early-bird" tickets), providing hope that high-performing pruned models can be discovered without incurring the full training cost of the dense model.

Empirical analysis of pruning techniques has inspired associated theoretical developments. Several works have derived bounds for the performance and size of subnetworks discovered in randomly-initialized networks (Malach et al., 2020; Orseau et al., 2020; Pensia et al., 2020). Other theoretical works analyze pruning via greedy forward selection (Ye et al., 2020; Ye et al., 2020). In addition to enabling analysis with respect to subnetwork size, pruning via greedy forward selection was shown to work well in practice for large-scale architectures and datasets. Although some findings from these works apply to randomly-initialized networks given proper assumptions (Ye et al., 2020; Malach et al., 2020; Orseau et al., 2020; Pensia et al., 2020), *no work yet analyzes how different levels of pre-training impact the performance of pruned networks from a theoretical perspective.*

This paper. We adopt the greedy forward selection pruning framework and analyze subnetwork performance with respect to the number of SGD pre-training iterations performed on the dense model. From this analysis, *we discover a threshold in the number of pre-training iterations—logarithmically dependent upon the size of the dataset—beyond which subnetworks obtained via greedy forward selection perform well in terms of training error.* Such a finding offers theoretical insight into the early-bird ticket phenomenon and provides intuition for why discovering high-performing subnetworks is more difficult in large-scale experiments (You et al., 2019; Renda et al., 2020; Liu et al., 2018). We validate our theoretical analysis through extensive experiments (i.e., two-layer networks on MNIST and CNNs on CIFAR10 and ImageNet), *finding that the amount of pre-training required to discover a subnetwork that performs well is consistently dependent on the size of the dataset.*

2. Preliminaries

Notation. Vectors are represented with bold type (e.g., $\mathbf{x}$), while scalars are represented by normal type (e.g., x). $\|\cdot\|_2$ represents the ℓ_2 vector norm. Unless otherwise specified, vector norms are always considered to be ℓ_2 norms. $[N]$ is used to represent the set of positive integers from 1 to N (i.e., $[N] = \{1 \dots N\}$). We denote the ball of radius r centered at $\boldsymbol{\mu}$ as $\mathcal{B}(\boldsymbol{\mu}, r)$.

Network Parameterization. We consider two-layer neural networks of width N:

$$f(\mathbf{x}, \boldsymbol{\Theta}) = \tfrac{1}{N} \sum_{i=1}^{N} \sigma(\mathbf{x}, \boldsymbol{\theta}_i), \tag{1}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ \dots \ , \boldsymbol{\theta}_N\}$ represents all weights within the two-layer neural network. In (1), $\sigma(\cdot, \boldsymbol{\theta}_i)$ represents the activation of a single neuron as shown below:

$$\sigma(\mathbf{x}, \boldsymbol{\theta}_i) = b_i \sigma_+(\mathbf{a}_i^\top \mathbf{x}). \tag{2}$$

Here, σ_+ is an activation function with bounded 1st and 2nd-order derivatives; see Assumption 1. The weights of a single neuron are $\boldsymbol{\theta}_i = [b_i, \mathbf{a}_i] \in \mathbb{R}^{d+1}$, where d is the dimension of the input vector.

Two-layer networks are relevant to larger-scale architectures and applications (Belilovsky et al., 2019; Hettinger et al., 2017; Nøkland and Eidnes, 2019) and have the special property of separability between hidden neurons—the network's output can be derived by computing (2) separately for each neuron. Notably, we do not leverage any results from mean-field analysis of two-layer networks (Mei et al., 2018; Ye et al., 2020), choosing instead to analyze the network's performance when trained directly with stochastic gradient descent (SGD) (Oymak and Soltanolkotabi, 2019).

The Dataset. We assume that our network is modeling a dataset D, where $|D| = m$. $D = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}_{i=1}^{m}$, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$ for all $i \in [m]$. During training, we consider an ℓ_2-norm regression loss over the dataset:

$$\mathcal{L}[f] = \frac{1}{2} \cdot \mathbb{E}_{(\mathbf{x},y)\sim D} \left[(f(\mathbf{x}, \boldsymbol{\Theta}) - y)^2 \right]. \tag{3}$$

We define $\mathbf{y} = \left[y^{(1)}, y^{(2)}, \ldots, y^{(m)} \right] / \sqrt{m}$, which represents a concatenated vector of all labels within the dataset scaled by a factor $\sqrt{m}$. Similarly, we define $\phi_{i,j} = \sigma(\mathbf{x}^{(j)}, \boldsymbol{\theta}_i)$ as the output of neuron i for the j-th input vector in the dataset and construct the vector $\boldsymbol{\Phi}_i = [\phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,m}] / \sqrt{m}$, which is a scaled, concatenated vector of output activations for a single neuron across the entire dataset. We use $\mathcal{M}_N$ to denote the convex hull over such activation vectors for all N neurons:

$$\mathcal{M}_N = \texttt{Conv}\left\{ \boldsymbol{\Phi}_i : i \in [N] \right\}. \tag{4}$$

Here, $\texttt{Conv}\{\cdot\}$ denotes the convex hull. $\mathcal{M}_N$ forms a marginal polytope of the feature map for all neurons in the two-layer network across every dataset example. We use $\texttt{Vert}(\mathcal{M}_N)$ to denote the vertices of the marginal polytope $\mathcal{M}_N$ (i.e., $\texttt{Vert}(\mathcal{M}_N) = \{\boldsymbol{\Phi}_i : i \in [N]\}$). Using the construction $\mathcal{M}_N$, the ℓ_2 loss can be easily defined as follows:

$$\ell(\mathbf{z}) = \frac{1}{2} \|\mathbf{z} - \mathbf{y}\|^2, \tag{5}$$

where $\mathbf{z} \in \mathcal{M}_N$. Finally, we define the diameter of the space $\mathcal{M}_N$ as $\mathcal{D}_{\mathcal{M}_N} = \max_{\mathbf{u},\mathbf{v} \in \mathcal{M}_N} \|\mathbf{u} - \mathbf{v}\|_2$.

3. Pruning with greedy forward selection

We assume the existence of a dense, two-layer network of width N, from which the pruned model is constructed. For now, *no assumption is made regarding the amount of pre-training for the dense model.* Given a subset of neuron indices from the dense model $\mathcal{S} \subseteq [N]$, we denote the output of the pruned subnetwork that only contains this subset of neurons as follows:

$$f_{\mathcal{S}}(\mathbf{x}, \boldsymbol{\Theta}) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \sigma(\mathbf{x}, \boldsymbol{\theta}_i). \tag{6}$$

Beginning from an empty subnetwork (i.e., $\mathcal{S} = \emptyset$), we aim to discover a subset of neurons $\mathcal{S}^\star = \arg\min_{\mathcal{S} \subseteq [N]} \mathcal{L}[f_{\mathcal{S}}]$ such that $|\mathcal{S}^\star| \ll N$. Instead of discovering an exact solution to this difficult combinatorial optimization problem, greedy forward selection (Ye et al., 2020) is used to find an approximate solution. The pruned network is constructed by selecting at each iteration k the neuron that yields the largest decrease in loss:

$$\mathcal{S}_{k+1} = \mathcal{S}_k \cup i^\star, \quad i^\star = \arg\min_{i \in [N]} \mathcal{L}[f_{\mathcal{S}_k \cup i}]. \tag{7}$$

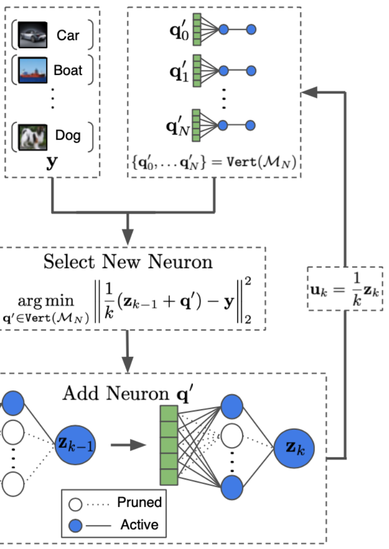
Car
Boat
Dog
$\mathbf{y}$
$\mathbf{q}'_0$
$\mathbf{q}'_1$
$\mathbf{q}'_N$
$\{\mathbf{q}'_0, \ldots \mathbf{q}'_N\} = \texttt{Vert}(\mathcal{M}_N)$
Select New Neuron
$\arg\min_{\mathbf{q}' \in \texttt{Vert}(\mathcal{M}_N)} \left\| \frac{1}{k}(\mathbf{z}_{k-1} + \mathbf{q}') - \mathbf{y} \right\|_2^2$
$\mathbf{u}_k = \frac{1}{k}\mathbf{z}_k$
Add Neuron $\mathbf{q}'$
$\mathbf{z}_{k-1}$
$\mathbf{z}_k$
Pruned
Active

Figure 1: Depiction of pruning with greedy forward selection as expressed in (8)-(10).

Using constructions from Section 2, we write the update rule for greedy forward selection as follows:

$$\text{(Select new neuron):} \quad \mathbf{q}_k = \arg\min_{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_n)} \ell\left(\tfrac{1}{k} \cdot (\mathbf{z}_{k-1} + \mathbf{q}) \right) \tag{8}$$

$$\text{(Add neuron to subnetwork):} \quad \mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{q}_k \tag{9}$$

$$\text{(Uniform average of neuron outputs):} \quad \mathbf{u}_k = \frac{1}{k} \cdot \mathbf{z}_k. \tag{10}$$

In words, (8)-(10) include the output of a new neuron, given by $\mathbf{q}_k$, within the current subnetwork at each pruning iteration based on a greedy minimization of the training loss $\ell(\cdot)$. Then, the output of the pruned subnetwork over the entire dataset at the k-th iteration, given by $\mathbf{u}_k \in \mathcal{M}_N$, is computed by taking a uniform average over the activation vectors of the k active neurons in $\mathbf{z}_k$. This pruning procedure, depicted in Figure 1, is the same as that of (Ye et al., 2020), but more explicitly matches the procedure in (7) by adopting a fixed, uniform average over selected neurons at each iteration. Within our analysis, rewriting the update rule as in (8)-(10) makes the comparison of subnetwork and dense network loss intuitive, *as the output of both dense and pruned networks is formulated as a uniform average over neuron activations*.

Notably, the greedy forward selection procedure in (8)-(10) can select the same neuron multiple times during successive pruning iterations. Such selection with replacement could be interpretted as a form of training during pruning—multiple selections of the same neuron is equivalent to modifying the neuron's output layer weight b_i in (2). Nonetheless, we highlight that such "training" does not violate the core purpose and utility of pruning: *we still obtain a smaller subnetwork with performance comparable to the dense network from which it was derived.*

4. How much pre-training do we really need?

As previously stated, no existing theoretical analysis has quantified the impact of pre-training on the performance of a pruned subnetwork. Here, we solve this problem by extending existing analysis for pruning via greedy forward selection to determine the relationship between SGD pre-training and subnetwork training loss. Full proofs are deferred to Appendix A, but we provide a sketch of the analysis within this section. We begin with all assumptions that are necessary for our analysis.

Assumption 1 *For some constant $\delta \in \mathbb{R}$, we assume the first and second derivatives of the network's activation function are bounded, such that $|\sigma'_+(\cdot)| \le \delta$ and $|\sigma''_+(\cdot)| \le \delta$ for σ_+ defined in (2).*

In comparison to (Ye et al., 2020), Assumption 1 lessens restrictions on the activation function[1] and removes boundedness assumptions on data and labels. Under these assumptions, we provide a bound for the loss of a subnetwork obtained after k iterations of greedy forward selection.

Lemma 1 *The following expression for the objective loss is true for iterates derived after k iterations of the update rule in (8)-(10) with a two-layer network of width N:*

$$\ell(\mathbf{u}_k) \le \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{k}\mathcal{L}_N. \tag{11}$$

Here, $\mathcal{L}_N$ represents the loss achieved by the dense network.

This result is similar in flavor to (Ye et al., 2020); yet, here it is derived under milder assumptions and modified to consider the loss of the dense network. Because assumptions in Lemma 1 match those of (Oymak and Soltanolkotabi, 2019), the expression above can be generalized to the stochastic case (i.e., with respect to randomness over SGD iterations) and modified to consider the number of SGD pre-training iterations performed on the dense network.

Theorem 2 *Assume Assumption 1 holds and that a two-layer network of width N was pre-trained for t iterations with SGD over a dataset D of size m. Additionally, assume that $Nd > m^2$ and*

1. We only require the first and second-order derivatives of the activation function to be bounded, whereas (Ye et al., 2020) imposes a Lipschitz bound and a bound on the maximum output value of the activation function.

$m > d$, where d represents the input dimension of data in D.[2] *Then, a subnetwork obtained from this dense network via k iterations of greedy forward selection satisfies the following:*

$$\mathbb{E}[\ell(\mathbf{u}_k)] \leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \mathcal{L}_0,$$

where $\mathcal{L}_0$ is the loss of the dense network at initialization, c and ζ are positive constants, and all expectations are with respect to randomness over SGD iterations on the dense network.

Trivially, the loss in Theorem 2 only decreases during successive pruning iterations if the rightmost $\mathcal{L}_0$ term does not dominate the expression (i.e., all other terms decay as $\mathcal{O}(\frac{1}{k})$). If this term does not decay with increasing k, the upper bound on subnetwork training loss deteriorates, thus eliminating any guarantees on subnetwork performance. Interestingly, we discover that the tightness of the upper bound in Theorem 2 depends on the number of dense network SGD pre-training iterations as follows.[3]

Theorem 3 *Adopting identical assumptions and notation as Theorem 2, assume the dense network is pruned via greedy forward selection for k iterations. The resulting subnetwork is guaranteed to achieve a training loss $\propto \mathcal{O}(\frac{1}{k})$ if t—the number of SGD pre-training iterations on the dense network—satisfies the following condition:*

$$t \gtrapprox \mathcal{O}\left(\frac{-\log(k)}{\log\left(1 - c\frac{d}{m^2}\right)}\right), \quad \text{where } c \text{ is a positive constant.} \tag{12}$$

Otherwise, the loss of the pruned network is not guaranteed to improve over successive iterations of greedy forward selection.

Discussion.[4] Our $\mathcal{O}(\frac{1}{k})$ rate in Lemma 1 matches that of previous work under mild assumptions, and explicitly expresses the loss of the subnetwork with respect to the loss of the dense network. Assumption 1 is chosen to align with the analysis of (Oymak and Soltanolkotabi, 2019). This combination enables the training loss of the pruned subnetwork to be expressed with respect to the number of pre-training iterations on the dense network as shown in Theorem 2.

Theorem 3 states that *a threshold exists in the number of SGD pre-training iterations of a dense network, beyond which subnetworks derived with greedy selection are guaranteed to achieve good training loss*. Denoting this threshold as $t^\star$, in the case that $m^2 \gg d$, $\lim_{m\to\infty} t^\star = \infty$, implying that larger datasets require more pre-training for high-performing subnetworks to be discovered. When $m^2 \approx d$, $\lim_{m\to cd} t^\star = 0$; i.e., minimal pre-training is required for subnetworks to perform well on small datasets. Interestingly, $t^\star$ scales logarithmically with the dataset size, implying that the amount of required pre-training will plateau as the underlying dataset becomes increasingly large.

Our finding provides theoretical insight regarding i) how much pre-training is sufficient to discover a subnetwork that performs well and ii) the difficulty of discovering high-performing

2. This overparameterization assumption is mild in comparison to previous work (Allen-Zhu et al., 2018; Du et al., 2019). Only recently was an improved, subquadratic requirement derived (Song et al., 2021).
3. We also derive a similar result using gradient descent (GD) instead of SGD; see Appendix A.
4. A faster rate can be achieved with greedy forward selection given Assumption 1 if $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$; see Appendix A.1.2. This result still holds under our milder assumptions, but the proof is similar to (Ye et al., 2020). It is true that such a result indicates that subnetworks perform well without pre-training when $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$, which seems to lessen the significance of Theorem 3. However, we analyze this assumption practically within Appendix B and show that, even for extremely small scale experiments, the assumption i) never holds at initialization and ii) tends to require more training for larger datasets, which aligns with findings in Theorem 3.

subnetworks in large-scale experiments (i.e., large datasets require more pre-training). Because several empirical heuristics for discovering high-performing subnetworks without full pre-training have already been proposed (You et al., 2019; Yin et al., 2019), we choose to not focus on deriving novel empirical methods. Additionally, our results specifically focus upon training accuracy, while generalization performance could be a more realistic assessment of subnetwork quality. It is possible that our analysis can be combined with theoretical results for neural network generalization performance (Arora et al., 2019; Li et al., 2018), but we leave such analysis as future work.

5. Related Work

Many variants have been proposed for both structured (Han et al., 2016; Li et al., 2016; Liu et al., 2017; Ye et al., 2020) and unstructured (Evci et al., 2019, 2020; Frankle and Carbin, 2018; Han et al., 2015) pruning. Generally, structured pruning, which prunes entire channels or neurons of a network instead of individual weights, is considered more practical, as it can achieve speedups without leveraging specialized libraries for sparse computation. Existing pruning criterion include the norm of weights (Li et al., 2016; Liu et al., 2017), feature reconstruction error (He et al., 2017; Luo et al., 2017; Ye et al., 2020; Yu et al., 2017), or even gradient-based sensitivity measures (Baykal et al., 2019; Wang et al., 2020; Zhuang et al., 2018). While most pruning methodologies perform backward elimination of neurons within the network (Frankle and Carbin, 2018; Frankle et al., 2019; Liu et al., 2017, 2018; Yu et al., 2017), some recent research has focused on forward selection structured pruning strategies (Ye et al., 2020; Ye et al., 2020; Zhuang et al., 2018). We adopt greedy forward selection within this work, as it has been previously shown to yield superior performance in comparison to greedy backward elimination.

Our analysis resembles that of the Frank-Wolfe algorithm (Frank et al., 1956; Jaggi, 2013), a widely-used technique for constrained, convex optimization. Recent work has shown that training deep networks with Frank-Wolfe can be made feasible in certain cases despite the non-convex nature of neural network training (Bach, 2014; Pokutta et al., 2020). Instead of training networks from scratch with Frank-Wolfe, however, we use a Frank-Wolfe-style approach to greedily select neurons from a pre-trained model. Such a formulation casts structured pruning as convex optimization over a marginal polytope, which can be analyzed similarly to Frank-Wolfe (Ye et al., 2020; Ye et al., 2020) and loosely approximates networks trained with standard, gradient-based techniques (Ye et al., 2020). Alternative methods of analysis for greedy selection algorithms could also be constructed with the use of sub-modular optimization techniques (Nemhauser et al., 1978).

Much work has been done to analyze the convergence properties of neural networks trained with gradient-based techniques (Chang et al., 2020; Hanin and Nica, 2019; Jacot et al., 2018; Zhang et al., 2019). Such convergence rates were originally explored for wide, two-layer neural networks using mean-field analysis techniques (Mei et al., 2019, 2018). Similar techniques were later used to extend such analysis to deeper models (Lu et al., 2020; Xiong et al., 2020). Generally, recent work on neural network training analysis has led to novel analysis techniques (Hanin and Nica, 2019; Jacot et al., 2018), extensions to alternate optimization methodologies (Jagatap and Hegde, 2018; Oymak and Soltanolkotabi, 2019), and even generalizations to different architectural components (Goel et al., 2018; Li and Yuan, 2017; Zhang et al., 2019). By adopting and extending such analysis, we aim to bridge the gap between the theoretical understanding of neural network training and LTH.

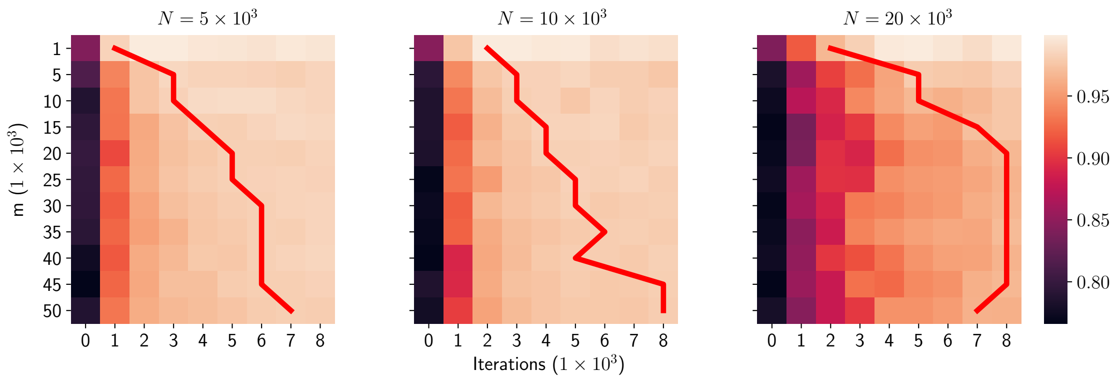

Figure 2: Pruned, two-layer models on MNIST. Sub-plots depict different dense network sizes, while the x and y axis depict the number of pre-training iterations and the sub-dataset size, respectively. Models are pruned to 200 hidden neurons every 1K iterations to measure subnetwork performance. Color represents training accuracy, and the red line depicts the point at which subnetworks surpass the performance of the best pruned model on the full dataset for different sub-dataset sizes.

6. Experiments

In this section, we empirically validate our theoretical results from Section 4, which predict that larger datasets require more pre-training for subnetworks obtained via greedy forward selection to perform well. Pruning via greedy forward selection has already been empirically analyzed in previous work. Therefore, *we differentiate our experiments by providing an in-depth analysis of the scaling properties of greedy forward selection with respect to the size and complexity of the underlying dataset.* In particular, we produce synthetic "sub-datasets" of different sizes and measure the amount of pre-training required to discover a high-performing subnetwork on each.

We perform analysis with two-layer networks on MNIST (Deng, 2012) and with deep CNNs (i.e., ResNet34 (He et al., 2015) and MobileNetV2 (Sandler et al., 2018)) on CIFAR10 and ImageNet (Krizhevsky et al., 2009; Deng et al., 2009). We find that i) *high-performing subnetworks can be consistently discovered without incurring the full pre-training cost*, and ii) *the amount of pre-training required for a subnetwork to perform well increases with the size and complexity of the underlying dataset in practice*. All experiments are run on an internal cluster with two Nvidia RTX 3090 GPUs using the public implementation of greedy forward selection (Ye, 2021).

6.1. Two-Layer Networks

We perform structured pruning experiments with two-layer networks on MNIST (Deng, 2012) by pruning hidden neurons via greedy forward selection. To match the single output neuron setup described in Section 2, we binarize MNIST labels by considering all labels less than five as zero and vice versa. Our model architecture matches the description in Section 2 with a few minor differences. Namely, we adopt a ReLU hidden activation and apply a sigmoid output transformation to enable training with binary cross entropy loss. Experiments are conducted with several different hidden dimensions (i.e., $N \in \{5\text{K}, 10\text{K}, 20\text{K}\}$). Hyperparameters are tuned using a hold out validation set; see Appendix C.1 for more details.

To study how dataset size impacts subnetwork performance, we construct sub-datasets of sizes 1K to 50K (i.e., in increments of 5K) from the original MNIST dataset by uniformly sampling

Model	Dataset Size	Pruned Accuracy				Dense Accuracy
		20K It.	40K It.	60K It.	80K It.	
MobileNetV2	10K	82.32	86.18	86.11	86.09	83.13
	30K	80.19	87.79	88.38	88.67	87.62
	50K	86.71	88.33	91.79	91.77	91.44
ResNet34	10K	75.29	85.47	85.56	85.01	85.23
	30K	84.06	91.59	92.31	92.15	92.14
	50K	89.79	91.34	94.28	94.23	94.18

Table 1: CIFAR10 test accuracy for subnetworks derived from dense networks with varying pre-training amounts (i.e., number of training iterations listed in top row) and sub-dataset sizes.

examples from the 10 original classes. The two-layer network is pre-trained for 8K iterations in total and pruned every 1K iterations to a size of 200 hidden nodes; see Appendix C.1 for a precise, algorithmic description of the two-layer network pruning process. After pruning, the accuracy of the pruned model over the entire training dataset is recorded (i.e., no fine-tuning is performed), allowing the impact of dataset size and pre-training length on subnetwork performance to be observed. See Figure 2 for these results, which are averaged across three trials.

Discussion. The performance of pruned subnetworks in Figure 2 matches the theoretical analysis provided in Section 4 for all different sizes of two-layer networks. Namely, *as the dataset size increases, so does the amount of pre-training required to produce a high-performing subnetwork.* To see this, one can track the trajectory of the red line, which traces the point at which the accuracy of the best performing subnetwork for the full dataset is surpassed at each sub-dataset size. This trajectory clearly illustrates that pre-training requirements for high-performing subnetworks increase with the size of the dataset. Furthermore, this increase in the amount of required pre-training is seemingly logarithmic, as the trajectory typically plateaus at larger dataset sizes.

Interestingly, despite the use of a small-scale dataset, high-performing subnetworks are never discovered at initialization, revealing that some minimal amount of pre-training is often required to obtain a good subnetwork via greedy forward selection. Previous work claims that high-performing subnetworks may exist at initialization in theory. In contrast, our empirical analysis shows that this is not the case even in simple experimental settings.

6.2. Deep Networks

We perform structured pruning experiments (i.e., channel-based pruning) using ResNet34 (He et al., 2015) and MobileNetV2 (Sandler et al., 2018) architectures on CIFAR10 and ImageNet (Krizhevsky et al., 2009; Deng et al., 2009). We adopt the same generalization of greedy forward selection to pruning deep networks as described in (Ye et al., 2020) and use ϵ to denote our stopping criterion; see Appendix C.2 for a complete algorithmic description. We follow the three-stage methodology—pre-training, pruning, and fine-tuning—and modify both the size of the underlying dataset and the amount of pre-training prior to pruning to examine their impact on subnetwork performance. Standard data augmentation and splits are adopted for both datasets.

CIFAR10. Three CIFAR10 sub-datasets of size 10K, 30K, and 50K (i.e., full dataset) are created using uniform sampling across classes. Pre-training is conducted for 80K iterations using SGD with

Model	**FLOP (Param) Ratio**	**Pruned Accuracy 50 Epoch**	**100 Epoch**	**150 Epoch**	**Dense Accuracy**
MobileNetV2	60% (80%)	70.05	71.14	71.53	71.70
	40% (65%)	69.23	70.36	71.10	
ResNet34	60% (80%)	71.68	72.56	72.65	73.20
	40% (65%)	69.87	71.44	71.33	

Table 2: Test accuracy on ImageNet of subnetworks with different FLOP levels derived from dense models with varying amounts of pre-training (i.e., training epochs listed in top row). We report the FLOP/parameter ratio after pruning with respect to the FLOPS/parameters of the dense model.

momentum and a cosine learning rate decay schedule starting at 0.1. We use a batch size of 128 and weight decay of $5 \cdot 10^{-4}$.[5] The dense model is independently pruned every 20K iterations, and subnetworks are fine-tuned for 2500 iterations with an intial learning rate of 0.01 prior to being evaluated. We adopt $\epsilon = 0.02$ and $\epsilon = 0.05$ for MobileNet-V2 and ResNet34, respecitvely, yielding subnetworks with a 40% decrease in FLOPS and 20% decrease in model parameters in comparison to the dense model.[6]

The results of these experiments are presented in Table 1. The amount of training required to discover a high-performing subnetwork consistently increases with the size of the dataset. For example, with MobileNetV2, a winning ticket is discovered on the 10K and 30K sub-datasets in only 40K iterations, while for the 50K sub-dataset a winning ticket is not discovered until 60K iterations of pre-training have been completed. Furthermore, subnetwork performance often surpasses the performance of the fully-trained dense network *without completing the full pre-training procedure*.

ImageNet. We perform experiments on the ILSVRC2012, 1000-class dataset (Deng et al., 2009) to determine how pre-training requirements change for subnetworks pruned to different FLOP levels.[7] We adopt the same experimental and hyperparameter settings as (Ye et al., 2020). Models are pre-trained for 150 epochs using SGD with momentum and cosine learning rate decay with an initial value of 0.1. We use a batch size of 128 and weight decay of $5 \cdot 10^{-4}$. The dense network is independently pruned every 50 epochs, and the subnetwork is fine-tuned for 80 epochs using a cosine learning rates schedule with an initial value of 0.01 before being evaluated. We first prune models with $\epsilon = 0.02$ and $\epsilon = 0.05$ for MobileNetV2 and ResNet34, respectively, yielding subnetworks with a 40% reduction in FLOPS and 20% reduction in parameters in comparison to the dense model. Pruning is also performed with a larger ϵ value (i.e., $\epsilon = 0.05$ and $\epsilon = 0.08$ for MobileNetV2 and ResNet34, respectively) to yield subnetworks with a 60% reduction in FLOPS and 35% reduction in model parameters in comparison to the dense model.

The results are reported in Table 2. Although the dense network is pre-trained for 150 epochs, subnetwork test accuracy reaches a plateau after only 100 epochs of pre-training in all cases. Furthermore, subnetworks with only 50 epochs of pre-training still perform well in many cases. E.g., the 60% FLOPS ResNet34 subnetwork with 50 epochs of pre-training achieves a testing accuracy

5. Our pre-training settings are adopted from a popular repository for the CIFAR10 dataset (Liu, 2017).
6. These settings are derived using a grid search over values of ϵ and the learning rate with performance measured over a hold-out validation set; see Appendix C.2.
7. We do not experiment with different sub-dataset sizes on ImageNet due to limited computational resources.

within 1% of the pruned model derived from the fully pre-trained network. Thus, *high-performing subnetworks can be discovered with minimal pre-training even on large-scale datasets like ImageNet*.

Discussion. These results demonstrate that the number of dense network pre-training iterations needed to reach a plateau in subnetwork performance i) consistently increases with the size of the dataset and ii) is consistent across different architectures given the same dataset. Discovering a high-performing subnetwork on the ImageNet dataset takes roughly 500K pre-training iterations (i.e., 100 epochs). In comparison, discovering a subnetwork that performs well on the MNIST and CIFAR10 datasets takes roughly 8K and 60K iterations, respectively. Thus, the amount of required pre-training iterations increases based on the size of dataset *even across significantly different scales and domains*. This indicates that dependence of pre-training requirements on dataset size may be an underlying property of discovering high-performing subnetworks no matter the experimental setting.

Per Theorem 3, the size of the dense network will not impact the number of pre-training iterations required for a subnetwork to perform well. This is observed to be true within our experiments; e.g., MobileNet and ResNet34 reach plateaus in subnetwork performance at similar points in pre-training for CIFAR10 and ImageNet in Tables 1 and 2. However, the actual loss of the subnetwork, as in Lemma 1, has a dependence on several constants that may impact subnetwork performance despite having no aymptotic impact on Theorem 3. E.g., a wider network could increase the width of the polytope $D_{\mathcal{M}_N}$ or initial loss $\ell(\mathbf{u}_1)$, leading to a looser upper bound on subnetwork loss. Thus, different sizes of dense networks, despite both reaching a plateau in subnetwork performance at the same point during pre-training, may yield subnetworks with different performance levels.

Interestingly, we observe that dense network size does impact subnetwork performance. In Figure 2, subnetwork performance varies based on dense network width, and subnetworks derived from narrower dense networks seem to achieve better performance. Similarly, in Tables 1 and 2, subnetworks derived from MobileNetV2 tend to achieve higher relative performance with respect to the dense model. Thus, subnetworks derived from smaller dense networks seem to achieve better *relative* performance in comparison to those derived from larger dense networks, suggesting that pruning via greedy forward selection may demonstrate different qualities in comparison to more traditional approaches (e.g., iterative magnitude-based pruning (Liu et al., 2018)). Despite this observation, however, the amount of pre-training epochs required for the emergence of the best-performing subnetwork is still consistent across architectures and dependent on dataset size.

7. Conclusion

In this work, we theoretically analyze the impact of dense network pre-training on the performance of a pruned subnetwork. By expressing pruned network loss with respect to the number of SGD iterations performed on its associated dense network, we discover a threshold in the number of pre-training iterations beyond which a pruned subnetwork achieves good training loss. Furthermore, we show that this threshold is logarithmically dependent upon the size of the dataset, which offers intuition into the early-bird ticket phenomenon and the difficulty of replicating pruning experiments at scale. We empirically verify our theoretical findings over several datasets (i.e., MNIST, CIFAR10, and ImageNet) with numerous network architectures (i.e., two-layer networks, MobileNetV2, and ResNet34), showing that the amount of pre-training required to discover a winning ticket is consistently dependent on the size of the underlying dataset. Several open problems remain, such as extending our analysis beyond two-layer networks, deriving generalization bounds for subnetworks pruned with greedy forward selection, or even using our theoretical results to discover new heuristic methods for identifying early-bird tickets in practice.

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *arXiv preprint arXiv:1811.04918*, 2018.

Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019.

Francis Bach. Breaking the Curse of Dimensionality with Convex Neural Networks. *arXiv e-prints*, art. arXiv:1412.8690, December 2014.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. SiPPing Neural Networks: Sensitivity-informed Provable Pruning of Neural Networks. *arXiv e-prints*, art. arXiv:1910.05422, October 2019.

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

Xiangyu Chang, Yingcong Li, Samet Oymak, and Christos Thrampoulidis. Provable benefits of overparameterization in model compression: From double descent to pruning neural networks. *arXiv preprint arXiv:2012.08749*, 2020.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. *arXiv preprint arXiv:2012.06908*, 2020.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The Lottery Ticket Hypothesis for Pre-trained BERT Networks. *arXiv e-prints*, art. arXiv:2007.12223, July 2020a.

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. EarlyBERT: Efficient BERT Training via Early-bird Lottery Tickets. *arXiv e-prints*, art. arXiv:2101.00063, December 2020b.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685. PMLR, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the Lottery: Making All Tickets Winners. *arXiv e-prints*, art. arXiv:1911.11134, November 2019.

Utku Evci, Yani A. Ioannou, Cem Keskin, and Yann Dauphin. Gradient Flow in Sparse Neural Networks and How Lottery Tickets Win. *arXiv e-prints*, art. arXiv:2010.03533, October 2020.

Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *arXiv e-prints*, art. arXiv:1803.03635, March 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Stabilizing the Lottery Ticket Hypothesis. *arXiv e-prints*, art. arXiv:1903.01611, March 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

Trevor Gale, Erich Elsen, and Sara Hooker. The State of Sparsity in Deep Neural Networks. *arXiv e-prints*, art. arXiv:1902.09574, February 2019.

Sharath Girish, Shishira R. Maiya, Kamal Gupta, Hao Chen, Larry Davis, and Abhinav Shrivastava. The Lottery Ticket Hypothesis for Object Recognition. *arXiv e-prints*, art. arXiv:2012.04643, December 2020.

Surbhi Goel, Adam Klivans, and Raghu Meka. Learning One Convolutional Layer with Overlapping Patches. *arXiv e-prints*, art. arXiv:1802.02547, February 2018.

Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv e-prints*, art. arXiv:1510.00149, October 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *arXiv e-prints*, art. arXiv:1602.01528, February 2016.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, art. arXiv:1512.03385, December 2015.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel Pruning for Accelerating Very Deep Neural Networks. *arXiv e-prints*, art. arXiv:1707.06168, July 2017.

Chris Hettinger, Tanner Christensen, Ben Ehlert, Jeffrey Humpherys, Tyler Jarvis, and Sean Wade. Forward thinking: Building and training neural networks one layer at a time. *arXiv preprint arXiv:1706.02480*, 2017.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

Gauri Jagatap and Chinmay Hegde. Learning relu networks via alternating minimization. *arXiv preprint arXiv:1806.07863*, 2018.

Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 427–435, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/jaggi13.html.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. *arXiv e-prints*, art. arXiv:1608.08710, August 2016.

Xingguo Li, Junwei Lu, Zhaoran Wang, Jarvis Haupt, and Tuo Zhao. On tighter generalization bound for deep neural networks: Cnns, resnets, and beyond. *arXiv preprint arXiv:1806.05159*, 2018.

Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. *arXiv preprint arXiv:1808.01204*, 2018.

Yuanzhi Li and Yang Yuan. Convergence Analysis of Two-layer Neural Networks with ReLU Activation. *arXiv e-prints*, art. arXiv:1705.09886, May 2017.

Kuang Liu. Pytorch-cifar. https://github.com/kuangliu/pytorch-cifar, 2017.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient Convolutional Networks through Network Slimming. *arXiv e-prints*, art. arXiv:1708.06519, August 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the Value of Network Pruning. *arXiv e-prints*, art. arXiv:1810.05270, October 2018.

Yiping Lu, Chao Ma, Yulong Lu, Jianfeng Lu, and Lexing Ying. A Mean-field Analysis of Deep ResNet and Beyond: Towards Provable Optimization Via Overparameterization From Depth. *arXiv e-prints*, art. arXiv:2003.05508, March 2020.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *arXiv e-prints*, art. arXiv:1707.06342, July 2017.

Eran Malach, Gilad Yehudai, Shai Shalev-Shwartz, and Ohad Shamir. Proving the Lottery Ticket Hypothesis: Pruning is All You Need. *arXiv e-prints*, art. arXiv:2002.00585, February 2020.

Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A Mean Field View of the Landscape of Two-Layers Neural Networks. *arXiv e-prints*, art. arXiv:1804.06561, April 2018.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit. *arXiv e-prints*, art. arXiv:1902.06015, February 2019.

Ari S Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *arXiv preprint arXiv:1906.02773*, 2019.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

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

Laurent Orseau, Marcus Hutter, and Omar Rivasplata. Logarithmic Pruning is All You Need. *arXiv e-prints*, art. arXiv:2006.12156, June 2020.

Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks. *arXiv e-prints*, art. arXiv:1902.04674, February 2019.

Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. Optimal lottery tickets via subsetsum: Logarithmic over-parameterization is sufficient. *arXiv preprint arXiv:2006.07990*, 2020.

Sebastian Pokutta, Christoph Spiegel, and Max Zimmer. Deep Neural Network Training with Frank-Wolfe. *arXiv e-prints*, art. arXiv:2010.07243, October 2020.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's Hidden in a Randomly Weighted Neural Network? *arXiv e-prints*, art. arXiv:1911.13299, November 2019.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing Rewinding and Fine-tuning in Neural Network Pruning. *arXiv e-prints*, art. arXiv:2003.02389, March 2020.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Chaehwan Song, Ali Ramezani-Kebrya, Thomas Pethick, Armin Eftekhari, and Volkan Cevher. Subquadratic overparameterization for shallow neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking Winning Tickets Before Training by Preserving Gradient Flow. *arXiv e-prints*, art. arXiv:2002.07376, February 2020.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On Layer Normalization in the Transformer Architecture. *arXiv e-prints*, art. arXiv:2002.04745, February 2020.

Mao Ye. Network-pruning-greedy-forward-selection. `https://github.com/lushleaf/Network-Pruning-Greedy-Forward-Selection`, 2021.

Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. In *International Conference on Machine Learning*, pages 10820–10830. PMLR, 2020.

Mao Ye, Lemeng Wu, and Qiang Liu. Greedy Optimization Provably Wins the Lottery: Logarithmic Number of Winning Tickets is Enough. *arXiv e-prints*, art. arXiv:2010.15969, October 2020.

Shihui Yin, Kyu-Hyoun Kim, Jinwook Oh, Naigang Wang, Mauricio Serrano, Jae-Sun Seo, and Jungwook Choi. The sooner the better: Investigating structure of early winning lottery tickets. 2019.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: Pruning Networks using Neuron Importance Score Propagation. *arXiv e-prints*, art. arXiv:1711.05908, November 2017.

Xiao Zhang, Yaodong Yu, Lingxiao Wang, and Quanquan Gu. Learning one-hidden-layer relu networks via gradient descent. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1524–1534. PMLR, 2019.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask. *arXiv e-prints*, art. arXiv:1905.01067, May 2019.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv e-prints*, art. arXiv:1710.01878, October 2017.

Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware Channel Pruning for Deep Neural Networks. *arXiv e-prints*, art. arXiv:1810.11809, October 2018.

Appendix A. Proofs

A.1. Convergence Analysis

Prior to presenting the proofs of the main theoretical results, we introduce several relevant technical lemmas.

Lemma 4 *Consider $\mathbf{u}_k, \mathbf{u}_{k-1} \in \mathcal{M}_N$, representing adjacent iterates of (8)-(10) at step k. Additionally, consider an arbitrary update $\mathbf{q} \in \texttt{Vert}(\mathcal{M}_N)$, such that $\mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{q}$ and $\mathbf{u}_k = \frac{1}{k}\mathbf{z}_k$. Then, we can derive the following expression for the difference between adjacent iterates of (8)-(10):*

$$\mathbf{u}_k - \mathbf{u}_{k-1} = \frac{1}{k}\left(\mathbf{q} - \mathbf{u}_{k-1}\right)$$

Lemma 5 *Because the objective $\ell(\cdot)$, defined over the space $\mathcal{M}_N$, is both quadratic and convex, the following expressions hold for any $\mathbf{s} \in \mathcal{M}_N$:*

$$\begin{aligned}
\ell(\mathbf{s}) &\geq \ell(\mathbf{u}_{k-1}) + \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1} \rangle \\
\ell(\mathbf{s}) &\leq \ell(\mathbf{u}_{k-1}) + \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1} \rangle + \|\mathbf{s} - \mathbf{u}_{k-1}\|_2^2
\end{aligned}$$

Observation 1 *From Lemma 5, we can derive the following inequality.*

$$\begin{aligned}
\mathcal{L}_N \geq \mathcal{L}_N^\star &= \min_{\mathbf{s} \in \mathcal{M}_N} \ell(\mathbf{s}) \\
&\geq \min_{\mathbf{s} \in \mathcal{M}_N} \left\{ \ell(\mathbf{u}_{k-1}) + \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1} \rangle \right\} \\
&= \ell(\mathbf{u}_{k-1}) + \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s}_k - \mathbf{u}_{k-1} \rangle .
\end{aligned}$$

Lemma 6 *Assume there exists a sequence of values $\{z_k\}_{k \geq 0}$ such that $z_0 = 0$ and*

$$|z_{k+1}|^2 \leq |z_k|^2 - 2\beta|z_k| + C, \quad \forall k \geq 0$$

where C and β are positive constants. Then, it must be the case that $|z_k| \leq \max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta}\right)$ for $k > 0$.

Proof We use an inductive argument to prove the above claim. For the base case of $z_0 = 0$, the claim is trivially true because $|z_0| = 0 \leq \max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta}\right)$. For the inductive case, we define $f(x) = x^2 - 2\beta x + C$. It should be noted that $f(\cdot)$ is a 1-dimensional convex function. Therefore, given some closed interval $[a, b]$ within the domain of f, the maximum value of f over this interval must be achieved on one of the end points. This fact simplifies to the following expression, where a and b are two values within the domain of f such that $a \leq b$.

$$\max_{x \in [a,b]} f(x) = \max\{f(a), f(b)\}$$

From here, we begin the inductive step, for which we consider two cases.

Case 1: Assume that $|z_k| \leq \frac{C}{2\beta}$. Then, the following expression for $|z_{k+1}|$ can be derived.

$$|z_{k+1}|^2 \leq f(|z_k|) \leq \max_z \left\{ f(z) : z \in \left[0, \tfrac{C}{2\beta}\right] \right\} = \max \left\{ f(0), f\left(\tfrac{C}{2\beta}\right) \right\}$$
$$= \max \left\{ C, \frac{C^2}{4\beta^2} \right\} \leq \left[\max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta} \right) \right]^2$$

Case 2: Assume that $|z_k| \geq \frac{C}{2\beta}$. Then, the following expression can be derived.

$$|z_{k+1}|^2 \leq |z_k|^2 - 2\beta|z_k| + C \overset{i)}{\leq} |z_k|^2 \leq \left[\max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta} \right) \right]^2$$

where $i)$ holds because $2\beta|z_k| \geq C$. In both cases, it is shown that $|z_{k+1}| \leq \max\left(\sqrt{C}, \frac{C}{2}, \frac{2}{2\beta}\right)$. Therefore, the lemma is shown to be true by induction. ■

Observation 2 *We commonly refer to the value of $\mathcal{L}_N$, representing the quadratic loss of the two-layer network over the full dataset. The value of $\mathcal{L}_N$ can be expressed as follows.*

$$\mathcal{L}_N = \ell\left(\frac{1}{N} \sum_{\mathbf{v} \in \mathtt{Vert}(\mathcal{M}_N)} \mathbf{v} \right) = \frac{1}{2m} \| f(\mathbf{X}, \boldsymbol{\Theta}) - \mathbf{Y} \|_2^2$$

The expression above is derived by simply applying the definitions associated with $\ell(\cdot)$ that are provided in Section 2.

A.1.1. PROOF OF LEMMA 1

We now present the proof of Lemma 1.

Proof We define $\mathcal{L}_N^\star = \min_{\mathbf{s} \in \mathcal{M}_N} \ell(\mathbf{s})$. Additionally, we define $\mathbf{s}_k$ as follows.

$$\mathbf{s}_k = \arg\min_{\mathbf{s} \in \mathcal{M}_N} \left\{ \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1} \rangle \right\} = \arg\min_{\mathbf{s} \in \mathtt{Vert}(\mathcal{M}_N)} \left\{ \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1} \rangle \right\}.$$

The second equality holds because a linear objective is being optimized on a convex polytope $\mathcal{M}_N$. Thus, the solution to this optimization is known to be achieved on some vertex $\mathbf{v} \in \mathtt{Vert}(\mathcal{M}_N)$. Recall, as stated in Section 2, that $\mathcal{D}_{\mathcal{M}_N}$ denotes the diameter of the marginal polytope $\mathcal{M}_N$.

We assume the existence of some global two-layer neural network with N hidden neurons from which the pruned network is derived. It should be noted that the N neurons of this global network are used to define the vertices $\mathbf{v} \in \mathtt{Vert}(\mathcal{M}_N)$ as described in Section 2. As a result, the loss of this global network, which we denote as $\mathcal{L}_N$, is the loss achieved by a uniform average over the N vertices of $\mathcal{M}_N$ (i.e., see (1)). In other words, the loss of the global network at the time of pruning is given by the expression below; see Observation 2.

$$\mathcal{L}_N = \ell\left(\frac{1}{N} \sum_{\mathbf{v} \in \mathtt{Vert}(\mathcal{M}_N)} \mathbf{v} \right)$$

It is trivially known that $\mathcal{L}_N \geq \mathcal{L}_N^\star$. Intuitively, the value of $\mathcal{L}_N$ has an implicit dependence on the amount of training underwent by the global network. However, we make no assumptions regarding the global network's training (i.e., $\mathcal{L}_N$ can be arbitrarily large for the purposes of this analysis). Using Observation 1, as well as Lemmas 4 and 5, we derive the following expression for the loss of iterates obtained with (10).

$$\begin{aligned}
\ell(\mathbf{u}_k) &\overset{i)}{=} \min_{\mathbf{q}\in\texttt{Vert}(\mathcal{M}_N)} \ell\left(\frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{q})\right)\\
&\overset{ii)}{\leq} \ell\left(\frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{s}_k)\right)\\
&\overset{iii)}{\leq} \ell(\mathbf{u}_{k-1}) + \left\langle \nabla\ell(\mathbf{u}_{k-1}), \frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{s}_k) - \mathbf{u}_{k-1}\right\rangle + \left\|\frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{s}_k) - \mathbf{u}_{k-1}\right\|_2^2\\
&\overset{iv)}{=} \ell(\mathbf{u}_{k-1}) + \left\langle \nabla\ell(\mathbf{u}_{k-1}), \frac{1}{k}(\mathbf{s}_k - \mathbf{u}_{k-1})\right\rangle + \left\|\frac{1}{k}(\mathbf{s}_k - \mathbf{u}_{k-1})\right\|_2^2\\
&\overset{v)}{\leq} \ell(\mathbf{u}_{k-1}) + \left\langle \nabla\ell(\mathbf{u}_{k-1}), \frac{1}{k}(\mathbf{s}_k - \mathbf{u}_{k-1})\right\rangle + \frac{1}{k^2}\cdot\mathcal{D}_{\mathcal{M}_N}^2\\
&= \ell(\mathbf{u}_{k-1}) + \frac{1}{k}\langle \nabla\ell(\mathbf{u}_{k-1}), \mathbf{s}_k - \mathbf{u}_{k-1}\rangle + \frac{1}{k^2}\cdot\mathcal{D}_{\mathcal{M}_N}^2\\
&\overset{vi)}{\leq} \left(1-\frac{1}{k}\right)\ell(\mathbf{u}_{k-1}) + \frac{1}{k}\cdot\mathcal{L}_N^\star + \frac{1}{k^2}\cdot\mathcal{D}_{\mathcal{M}_N}^2\\
&\overset{vii)}{\leq} \left(1-\frac{1}{k}\right)\ell(\mathbf{u}_{k-1}) + \frac{1}{k}\cdot\mathcal{L}_N + \frac{1}{k^2}\cdot\mathcal{D}_{\mathcal{M}_N}^2
\end{aligned}$$

where $i)$ is due to (8)-(10), $ii)$ is because $\mathbf{s}_k \in \texttt{Vert}(\mathcal{M}_N)$, $iii)$ is from Lemma 5, $iv)$ is from Lemma 4 since it holds $\frac{1}{k}z_{k-1} - u_{k-1} = -\frac{1}{k}u_{k-1} \Rightarrow u_{k-1} = \frac{1}{k-1}z_{k-1}$, $v)$ is from the definition of $\mathcal{D}_{\mathcal{M}_N}$, and $vi - vii)$ are due to Observation 1. This expression can then be rearranged to yield the following recursive expression:

$$\ell(\mathbf{u}_k) \leq \left(1-\frac{1}{k}\right)\ell(\mathbf{u}_{k-1}) + \frac{1}{k}\cdot\mathcal{L}_N + \frac{1}{k^2}\cdot\mathcal{D}_{\mathcal{M}_N}^2 \Rightarrow$$
$$\ell(\mathbf{u}_k) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}_{\mathcal{M}_N}^2 \leq \left(1-\frac{1}{k}\right)\cdot\left(\ell(\mathbf{u}_{k-1}) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}_{\mathcal{M}_N}^2\right)$$

By unrolling the recursion in this expression over k iterations, we get the following:

$$\ell(\mathbf{u}_k) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}_{\mathcal{M}_N}^2 \leq \prod_{i=2}^{k}\left(1-\frac{1}{i}\right)\cdot\left(\ell(\mathbf{u}_1) - \mathcal{L}_N - \frac{1}{2}\cdot\mathcal{D}_{\mathcal{M}_N}^2\right) \Rightarrow$$
$$\ell(\mathbf{u}_k) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}_{\mathcal{M}_N}^2 \leq \frac{1}{k}\cdot\left(\ell(\mathbf{u}_1) - \mathcal{L}_N - \frac{1}{2}\cdot\mathcal{D}_{\mathcal{M}_N}^2\right)$$

By rearranging terms, we arrive at the following expression

$$\ell(\mathbf{u}_k) \leq \frac{1}{k}\cdot\left(\ell(\mathbf{u}_1) - \mathcal{L}_N + \frac{1}{2}\cdot\mathcal{D}_{\mathcal{M}_N}^2\right) + \mathcal{L}_N \leq \mathcal{O}\left(\frac{1}{k}\right) + \mathcal{L}_N \tag{13}$$

From here, we expand this expression as follows, yielding the final expression from Lemma 1.

$$\begin{aligned}\ell(\mathbf{u}_k) &\leq \frac{1}{k}\left(\ell(\mathbf{u}_1) - \mathcal{L}_N + \frac{1}{2}\mathcal{D}^2_{\mathcal{M}_N}\right) + \mathcal{L}_N \\ &= \frac{1}{k}\left(\ell(\mathbf{u}_1) + \frac{1}{2}\mathcal{D}^2_{\mathcal{M}_N}\right) + \frac{k-1}{k}\mathcal{L}_N \\ &= \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{k}\mathcal{L}_N\end{aligned}$$

■

A.1.2. Proof of a Faster Rate

Similar to Ye et al. (2020), we can obtain a faster $\mathcal{O}(\frac{1}{k^2})$ rate under assumption 1 and the assumption that $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$.

Lemma 7 *Assume that Assumption 1 holds and $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$. Then, the following bound is achieved for two-layer neural networks of width N after k iterations of greedy forward selection:*

$$\ell(\mathbf{u}_k) = \mathcal{O}\left(\frac{1}{k^2\min(1, \gamma)}\right), \quad \textit{where } \gamma \textit{ is a positive constant.}$$

Proof *We define $\mathbf{w}_k = k(\mathbf{y} - \mathbf{u}_k)$. Also recall that $\ell(\mathbf{z}) = \frac{1}{2}\|\mathbf{z} - \mathbf{y}\|_2^2$. Furthermore, we define $\mathbf{s}_{k+1}$ as follows.*

$$\begin{aligned}\mathbf{s}_{k+1} &= \arg\min_{\mathbf{s}\in\mathcal{M}_N} \nabla\ell(\mathbf{u}_k)^\top(\mathbf{s} - \mathbf{u}_k) \\ &= \arg\min_{\mathbf{s}\in\mathcal{M}_N} \langle\nabla\ell(\mathbf{u}_k), \mathbf{s} - \mathbf{u}_k\rangle \\ &\overset{i)}{=} \arg\min_{\mathbf{s}\in\mathcal{M}_N} \langle\mathbf{u}_k - \mathbf{y}, \mathbf{s} - \mathbf{u_k}\rangle \\ &= \arg\min_{\mathbf{s}\in\mathcal{M}_N} \langle-\mathbf{w}_k, \mathbf{s} - \mathbf{u}_k\rangle \\ &= \arg\min_{\mathbf{s}\in\mathcal{M}_N} \langle\mathbf{w}_k, \mathbf{u}_k - \mathbf{s}\rangle \\ &= \arg\min_{\mathbf{s}\in\mathcal{M}_N} \{\langle\mathbf{w}_k, \mathbf{u}_k\rangle + \langle\mathbf{w_k}, -\mathbf{s}\rangle\} \\ &= \arg\min_{\mathbf{s}\in\mathcal{M}_N} \langle\mathbf{w_k}, -\mathbf{s}\rangle \\ &= \arg\min_{\mathbf{s}\in\mathcal{M}_N} \langle\mathbf{w}_k, \mathbf{y} - \mathbf{s}\rangle\end{aligned}$$

where $i)$ follows from (5). Notice that $\mathbf{s}_k$ minimizes a linear objective (i.e., the dot product with $\mathbf{w}_k$) over the domain of the marginal polytope $\mathcal{M}_N$. As a result, the optimum is achieved on a *vertex of the marginal polytope, implying that $\mathbf{s}_k \in \texttt{Vert}(\mathcal{M}_N)$ for all $k > 0$. We assume that $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$. Under this assumption, it is known that $\mathbf{s}^\star = \mathbf{y} + \gamma \frac{\mathbf{w}_k}{\|\mathbf{w}_k\|_2} \in \mathcal{M}_N$, which allows the following to be derived.*

$$\begin{aligned}\langle \mathbf{w}_k, \mathbf{y} - \mathbf{s}_{k+1} \rangle &= \min_{\mathbf{s} \in \mathcal{M}_N} \langle \mathbf{w}_k, \mathbf{y} - \mathbf{s} \rangle \\ &\leq \langle \mathbf{w}_k, \mathbf{y} - \mathbf{s}^\star \rangle \\ &= -\gamma \|\mathbf{w}_k\|_2 \end{aligned} \tag{14}$$

From (10), the following expressions for $\mathbf{u}_k$ and $\mathbf{q}_k$ can be derived.

$$\mathbf{u}_k = \frac{1}{k} \cdot \mathbf{z}_k = \frac{1}{k} \cdot [(k-1)\mathbf{u}_{k-1} + \mathbf{q}_k] \tag{15}$$

$$\begin{aligned}\mathbf{q}_k &= \operatorname*{arg\,min}_{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_N)} \ell\left(\frac{1}{k}[\mathbf{z}_{k-1} + \mathbf{q}]\right) \\ &= \operatorname*{arg\,min}_{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_N)} \left\| \frac{1}{k}[(k-1)\mathbf{u}_{k-1} + \mathbf{q}] - \mathbf{y} \right\|_2^2 \end{aligned} \tag{16}$$

Combining all of this together, the following expression can be derived for $\|\mathbf{w}_k\|_2^2$, where $\mathcal{D}_{\mathcal{M}_N}$ is the diameter of $\mathcal{M}_N$:

$$\begin{aligned}\|\mathbf{w}_k\|_2^2 &= \|k(\mathbf{y} - \mathbf{u}_k)\|_2^2 = \|k(\mathbf{u}_k - \mathbf{y})\|_2^2 \\ &\overset{i)}{=} \min_{\mathbf{q} \in \mathcal{M}_N} \|k(\tfrac{1}{k}[(k-1)\mathbf{u}_{k-1} + \mathbf{q}] - \mathbf{y})\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \|(k-1)\mathbf{u}_{k-1} + \mathbf{q} - k\mathbf{y}\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \|-(k-1)\mathbf{y} + (k-1)\mathbf{u}_{k-1} + \mathbf{q} - \mathbf{y}\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \|-\mathbf{w}_{k-1} - \mathbf{y} + \mathbf{q}\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \|\mathbf{w}_{k-1} + \mathbf{y} - \mathbf{q}\|_2^2 \\ &\overset{ii)}{\leq} \|\mathbf{w}_{k-1} + \mathbf{y} - \mathbf{s}_k\|_2^2 \\ &= \|\mathbf{w}_{k-1}\|_2^2 + 2\langle \mathbf{w}_{k-1}, \mathbf{y} - \mathbf{s}_k \rangle + \|\mathbf{y} - \mathbf{s}_k\|_2^2 \\ &\leq \|\mathbf{w}_{k-1}\|_2^2 + 2\langle \mathbf{w}_{k-1}, \mathbf{y} - \mathbf{s}_k \rangle + \mathcal{D}_{\mathcal{M}_N}^2 \\ &\overset{iii)}{\leq} \|\mathbf{w}_{k-1}\|_2^2 - 2\gamma \|\mathbf{w}_{k-1}\|_2 + \mathcal{D}_{\mathcal{M}_N}^2 \end{aligned}$$

where $i)$ follows from (15) and (16), $ii)$ follows from the fact that $\mathbf{s}_k \in \mathcal{M}_N$, and $iii)$ follows from (14). Therefore, from this analysis, the following recursive expression for the value of $\|\mathbf{w}_k\|^2$ is derived:

$$\|\mathbf{w}_k\|_2^2 \leq \|\mathbf{w}_{k-1}\|_2^2 - 2\gamma \|\mathbf{w}_{k-1}\|_2 + \mathcal{D}_{\mathcal{M}_N}^2 \tag{17}$$

Then, by invoking Lemma 6, we derive the following inequality.

$$\begin{aligned}\|\mathbf{w}_k\|_2^2 &\leq \max\left\{\mathcal{D}_{\mathcal{M}_n}, \frac{\mathcal{D}^2_{\mathcal{M}_N}}{2}, \frac{\mathcal{D}^2_{\mathcal{M}_N}}{2\gamma}\right\} \\ &= \mathcal{O}\left(\frac{1}{\min(1,\gamma)}\right)\end{aligned}$$

With this in mind, the following expression can then be derived for the loss $\ell(\mathbf{u}_k)$ achieved by (8)-(10) after k iterations.

$$\ell(\mathbf{u}_k) = \frac{1}{2}\|\mathbf{u}_k - \mathbf{y}\|_2^2 = \frac{\|\mathbf{w}_k\|_2^2}{2k^2} = \mathcal{O}\left(\frac{1}{k^2\min(1,\gamma)^2}\right)$$

This yields the desired expression, thus completing the proof. ■

A.2. Training Analysis

Prior to analyzing the amount of training needed for a good pruning loss, several supplemental theorems and lemmas exist that must be introduced. From (Oymak and Soltanolkotabi, 2019), we utilize theorems regarding the convergence rates of two-layer neural networks trained with GD and SGD. We begin with the theorem for the convergence of GD in Theorem 8, then provide the associated convergence rate for SGD within Theorem 9. Both Theorems 8 and 9 are simply restated from (Oymak and Soltanolkotabi, 2019) for convenience purposes.

Theorem 8 *(Oymak and Soltanolkotabi, 2019). Assume there exists a two-layer neural network and associated dataset as described in Section 2. Denote N as the number of hidden neurons in the two-layer neural network, m as the number of unique examples in the dataset, and d as the input dimension of examples in the dataset. Assume without loss of generality that the input data within the dataset is normalized so that $\|\mathbf{x}^{(i)}\|_2 = 1$ for all $i \in [m]$. A moderate amount of overparameterization within the two-layer network is assumed, given by $Nd > m^2$. Furthermore, it is assumed that $m > d$ and that the first and second derivatives of the network's activation function are bounded (i.e., $|\sigma'_+(\cdot)| \leq \delta$ and $|\sigma''_+(\cdot)| \leq \delta$ for some $\delta \in \mathbb{R}$). Given these assumptions, the following bound is achieved with a high probability by training the neural network with gradient descent.*

$$\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2 \leq \left(1 - c\frac{d}{m}\right)^t \cdot \|f(\mathbf{X}, \mathbf{\Theta}_0) - \mathbf{Y}\|_2 \tag{18}$$

In (18), $c \in \mathbb{R}$, $\mathbf{\Theta}_t = \{\boldsymbol{\theta}_{1,t}, \dots, \boldsymbol{\theta}_{N,t}\}$ represents the network weights at iteration t of gradient descent, $f(\cdot,\cdot) \in \mathbb{R}^m$ represents the network output over the full dataset $\mathbf{X} \in \mathbb{R}^{m\times d}$, and $Y = [y^{(1)}, \dots, y^{(m)}]^T$ represents a vector of all dataset labels. $\mathbf{\Theta}_0$ is assumed to be randomly sampled from a normal distribution (i.e., $\mathbf{\Theta}_0 \sim \mathcal{N}(0,1)$).

Theorem 9 *(Oymak and Soltanolkotabi, 2019). Here, all assumptions of Theorem 8 are adopted, but we assume the two-layer neural network is trained with SGD instead of GD. For SGD, parameter updates are performed over a sequence of randomly-sampled examples within the training dataset* *(i.e., the true gradient is not computed for each update). Given the assumptions, there exists some event E with probability $P[E] \geq \max\left(\kappa,\ 1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)$, where $c_1, c_2, \kappa \in \mathbb{R}$, $0 \leq c_1 \leq \frac{4}{3}$, and $0 < \kappa \leq 1$. Given the event E, with high probability the following bound is achieved for training a two-layer neural network with SGD.*

$$\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right] \leq \left(1 - c\frac{d}{m^2}\right)^t \cdot \|f(\mathbf{X}, \mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \tag{19}$$

In (19), $c \in \mathbb{R}$, $\mathbf{\Theta}_t$ represent the network weights at iteration t of SGD, $\mathbb{1}_E$ is the indicator function for event E, $f(\cdot,\cdot)$ represents the output of the two layer neural network over the entire dataset $\mathbf{X} \in \mathbb{R}^{m\times d}$, and $\mathbf{Y} \in \mathbb{R}^m$ represents a vector of all labels in the dataset.

It should be noted that the overparameterization assumptions within Theorems 8 and 9 are very mild, which leads us to adopt this analysis within our work. Namely, we only require that the number of examples in the dataset exceeds the input dimension and the number of parameters within the first neural network layer exceeds the squared size of the dataset. In comparison, previous work lower bounds the number of hidden neurons in the two-layer neural network (i.e., more restrictive than the number of parameters in the first layer) with higher-order polynomials of m to achieve similar convergence guarantees (Allen-Zhu et al., 2018; Du et al., 2019; Li and Liang, 2018).

In comparing the convergence rates of Theorems 8 and 9, one can notice that these linear convergence rates are very similar. The extra factor of m within the denominator of Theorem 9 is intuitively due to the fact that m updates are performed in a single pass through the dataset for SGD, while GD uses the full dataset at every parameter update. Such alignment between the convergence guarantees for GD and SGD allows our analysis to be similar for both algorithms.

We now state a relevant technical lemma based on Lemma 1. Beginning from (13), the following can be shown.

A.2.1. PROOF OF THEOREM 2

We now provide the proof for Theorem 2.

Proof From Theorem 9, we have a bound for $\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right]$, where $\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2$ represents the loss over the entire dataset after t iterations of SGD (i.e., without the factor of $\frac{1}{2}$). Two sources of stochasticity exist within the expectation expression $\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right]$: $i)$ randomness over the event E and $ii)$ randomness over the t-th iteration of SGD given the first $t-1$ iterations. The probability of event E is independent of the randomness over SGD iterations, which allows the following expression to be derived.

$$\begin{aligned}\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right] &\overset{i)}{=} \mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right] \cdot \mathbb{E}\left[\mathbb{1}_E\right] \\ &\overset{ii)}{\geq} \max\left(\kappa,\ 1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right) \mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right]\end{aligned}$$

where $i)$ holds from the independence of expectations and $ii)$ is derived from the probability expression for event E in Theorem 9. Notably, the expectation within the above expression now has only a single source of stochasticity—the randomness over SGD iterations. Combining the above expression with (19) from Theorem 9 yields the following, where two possible cases exist.

Case 1: $\max\left(\kappa,\ 1 - c_1(c_2\sqrt{\frac{m}{d}})\right) = 1 - c_1(c_2\sqrt{\frac{m}{d}})$

$$\left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)\mathbb{E}\left[\|f(\mathbf{X},\mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \Rightarrow$$

$$\mathbb{E}\left[\|f(\mathbf{X},\mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)^{-1}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \quad (20)$$

Case 2: $\max\left(\kappa,\ 1 - c_1(c_2\sqrt{\frac{m}{d}})\right) = \kappa$

$$\kappa \cdot \mathbb{E}\left[\|f(\mathbf{X},\mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \Rightarrow$$

$$\mathbb{E}\left[\|f(\mathbf{X},\mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \kappa^{-1}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \quad (21)$$

From here, we use Observation 2 to derive the following, where the expectation is with respect to randomness over SGD iterations (i.e., we assume the global two-layer network of width N is trained with SGD).

$$\begin{aligned}\mathbb{E}[\mathcal{L}_N] &= \mathbb{E}\left[\frac{1}{2m}\|f(\mathbf{X},\mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right] \\ &\overset{i)}{=} \frac{1}{2m}\mathbb{E}\left[\|f(\mathbf{X},\mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right]\end{aligned}$$

Here, the equality in $i)$ holds true because $\frac{1}{2m}$ is a constant value given a fixed dataset. Now, notice that this expectation expression $\mathbb{E}\left[\|f(\mathbf{X},\mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right]$ is identical to the expectation within (20)-(21) (i.e., both expectations are with respect to randomness over SGD iterations). Thus, the above expression can be combined with (20) and (21) to yield the following.

$$\mathbb{E}[\mathcal{L}_N] \leq \frac{1}{2m}\left[\max\left(\kappa, \left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)\right)\right]^{-1}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \quad (22)$$

Then, we can substitute (22) into Lemma 1 to derive the final result, where expectations are with respect to randomness over SGD iterations. We also define $\mathcal{L}_0 = \|f(\mathbf{X},\Theta_0) - \mathbf{Y}\|_2^2$ and $\zeta = \left[\max\left(\kappa, \left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)\right)\right]^{-1} > 0$.

$$\begin{aligned}\mathbb{E}[\ell(\mathbf{u}_k)] &\leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{k-1}{k}\mathbb{E}[\mathcal{L}_N] \\ &\leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \\ &= \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \mathcal{L}_0 \quad (23)\end{aligned}$$

$\blacksquare$

A.2.2. PROOF OF THEOREM 3

We now provide the proof for Theorem 3.
Proof We begin with (23) from the proof of Theorem 2.

$$\mathbb{E}[\ell(\mathbf{u}_k)] \leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \mathcal{L}_0$$

It can be seen that all terms on the right-hand-side of the equation above will decay to zero as k increases aside from the rightmost term. The rightmost term of (23) will remain fixed as k increases due to its factor of $k-1$ in the numerator.

Within (23), there are two parameters that can be modified by the practitioner: N (i.e., ζ depends on N) and t. All other factors within the expressions are constants based on the dataset that cannot be modified. N only appears in the $1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}$ factor of ζ, thus revealing that the value of N cannot be naively increased within (23) to remove the factor of $k-1$.

To determine how t can be modified to achieve a better pruning rate, we notice that a setting of $\left(1 - c\frac{d}{m^2}\right)^t = \mathcal{O}\left(\frac{1}{k}\right)$ would cancel the factor of $k-1$ in (23). With this in mind, we observe the following.

$$\begin{aligned}
\left(1 - c\frac{d}{m^2}\right)^t &= O\left(\frac{1}{k}\right) \Rightarrow \\
t \cdot \log\left(1 - c\frac{d}{m^2}\right) &= O(-\log(k)) \Rightarrow \\
t &= \frac{O(-\log(k))}{\log(1 - c\frac{d}{m^2})} \Rightarrow \\
t &\approx O\left(\frac{-\log(k)}{\log(1 - c\frac{d}{m^2})}\right) \qquad (24)
\end{aligned}$$

If the amount of training in (24) is satisfied, the factor of $k-1$ in the rightmost term in Eq. (23) will be canceled, causing the expected pruning loss to decay to zero as k increases. Based on Theorem 9, it must be the case that $(1 - c\frac{d}{m^2}) \in [0,1]$ in order for SGD to converge. As a result, $\lim_{t\to\infty}(1 - c\frac{d}{m^2})^t = 0$, causing the rightmost term in (23) to approach zero as $t \to \infty$. In other words, the value of t can increase beyond the bound in (24) without damaging the loss of the pruned network, as the rightmost term in (23) will simply become zero. This observation that t can increase beyond the value in (24) yields the final expression for the number of SGD iterations required to achieve the desired $\mathcal{O}(\frac{1}{k})$ pruning rate.

$$t \gtrapprox O\left(\frac{-\log(k)}{\log(1 - c\frac{d}{m^2})}\right)$$

■

A.2.3. SUPPLEMENTAL RESULT AND PROOF WITH GD

In addition to the proof of Theorem 3, we provide a similar result for neural networks trained with GD to further support our analysis of the amount of training required to achieve good loss via pruning.

Theorem 10 *Assume a two-layer neural network of width N was trained for t iterations with gradient descent over a dataset of size m. Furthermore, assume that $Nd > m^2$ and $m > d$, where d represents the input dimension of data in D. When the network is pruned via (8)-(10), it will achieve a loss $\mathcal{L}' \propto O(\frac{1}{k})$ if the number of gradient descent iterations t satisfies the following condition.*

$$t \gtrapprox O\left(\frac{-\log(k)}{\log\left(1 - c\frac{d}{m}\right)}\right) \tag{25}$$

Otherwise, the loss of the pruned network will not improve during successive iterations of (8)-(10).

Proof Beginning with Lemma 1, we can use Theorem 8 to arrive at the following expression. Here, we define $\mathcal{L}_0 = \|f(\mathbf{X}, \mathbf{\Theta}_0) - \mathbf{Y}\|_2^2$ and define $\mathcal{L}_N$ as described in Observation 2.

$$\begin{aligned} l(\mathbf{u}_k) &\leq \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{k}\mathcal{L}_N \\ &\overset{i)}{=} \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{2mk}\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \\ &\overset{ii)}{\leq} \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{2k}(1 - c\frac{d}{m})^{2t} \cdot \mathcal{L}_0 \end{aligned} \tag{26}$$

where $i)$ is due to Observation 2 and $ii)$ is from Theorem 8. From this expression, one can realize that an $O(\frac{1}{k})$ rate is only achieved if the factor of $k-1$ within the rightmost term of (26) is removed. This factor, if not counteracted, will cause the upper bound for $\ell(\mathbf{u^k})$ to remain constant, as the right hand expression of (26) would be dominated by the value of $\mathcal{L}_0$. However, this poor upper bound can be avoided by manipulating the $(1 - c\frac{d}{m})^{2t}$ term to eliminate the factor of $k-1$. For this to happen, it must be true that $(1 - c\frac{d}{m})^{2t} \approx O(\frac{1}{k})$, which allows the following asymptotic expression for t to be derived.

$$\begin{aligned} \left(1 - c\frac{d}{m}\right)^{2t} &= O(\frac{1}{k}) \Rightarrow \\ 2t \cdot \log\left(1 - c\frac{d}{m}\right) &= O(-\log(k)) \Rightarrow \\ t &= \frac{O(-\log(k))}{2 \cdot \log\left(1 - c\frac{d}{m}\right)} \Rightarrow \\ t &\approx O\left(\frac{-\log k}{\log\left(1 - c\frac{d}{m}\right)}\right) \end{aligned} \tag{27}$$

If the amount of training in (27) is satisfied, it allows the factor of $k-1$ in the rightmost term of 26 to be canceled. It is trivially true that $\lim_{t\to\infty}(1 - c\frac{d}{m})^{2t} = 0$ because $(1 - c\frac{d}{m}) \in [0, 1]$ if gradient descent converges; see Theorem 8. As a result, the rightmost term in (26) also approaches zero as t

m	MNIST	CIFAR10
1000	1	4
2000	1	7
3000	1	6
4000	2	9
5000	2	10

Table 3: Results of empirically analyzing whether $\mathbf{y} \in \mathcal{M}_N$. For each of the possible datasets and sizes, we report the number of training epochs required before the assumption was satisfied for the best possible learning rate setting.

increases, allowing the $O(\frac{1}{k})$ pruning rate to be achieved. This observation that t can be increased beyond the value in (27) without issues leads to the final expression from Theorem 10.

$$t \gtrapprox O\left(\frac{-\log(k)}{\log\left(1 - c\frac{d}{m}\right)}\right)$$

■

Appendix B. Empirical Analysis of $\mathbf{y} \in \mathcal{M}_N$

To achieve the faster rate provided in Lemma 7, we make the assumption that $B(\mathbf{y}, \gamma) \in \mathcal{M}_N$. If it is assumed that $\gamma > 0$, this assumption is slightly stronger than $\mathbf{y} \in \mathcal{M}_N$, as it implies $\mathbf{y}$ cannot lie on the perimeter of $\mathcal{M}_N$. However, this assumption roughly requires that $\mathbf{y} \in \mathcal{M}_N$. Although previous work has attempted to analyze this assumption theoretically (Ye et al., 2020), it is not immediately clear whether this assumption is reasonable in practice.

We perform experiments using two-layer neural networks trained on different image classification datasets to determine if this assumption is satisfied in practice. Namely, we train a two-layer neural network on uniformly downsampled (i.e., equal number of examples taken from each class) versions of the MNIST and CIFAR10 datasets and test the $\mathbf{y} \in \mathcal{M}_N$ condition following every epoch (i.e., see Section B.2 for details). We record the first epoch of training after which $\mathbf{y} \in \mathcal{M}_N$ is true and report the results in Table 3. As can be seen, *the assumption is never satisfied at initialization and tends to require more training for larger datasets*.

B.1. Models and Datasets

All tests were performed with a two-layer neural network as defined in Section 2. This model contains a single output neuron, N hidden neurons, and uses a smooth activation function (i.e., we use the sigmoid activation). For each test, we ensure the overparameterization requirements presented in Theorem 3 (i.e., $Nd > m^2$ and $m > d$, where N is the number of hidden neurons, d is the input dimension, and m is the size of the dataset) are satisfied. For simplicity, the network is trained without any data augmentation, batch normalization, or dropout.

Experiments are performed using the MNIST and CIFAR10 datasets. These datasets are reduced in size in order to make overparameterization assumptions within the two-layer neural network easier

m	N
1000	1300
2000	6000
3000	12000
4000	21000
5000	32000

Table 4: Displays the hidden dimension of the two-layer neural network used to test the $\mathbf{y} \in \mathcal{M}_N$ assumption for different dataset sizes. Hidden dimensions are the same between tests with MNIST and CIFAR10.

to satisfy. In particular, we perform tests with 1000, 2000, 3000, 4000, and 5000 dataset examples for each of the separate datasets, where each dataset size is constructed by randomly sampling an equal number of data examples form each of the ten possible classes. Additionally, CIFAR10 images are downsampled from a spatial resolution of 32×32 to a spatial resolution of 18×18 using bilinear interpolation to further ease overparameterization requirements. Dataset examples were flattened to produce a single input vector for each image, which can be passed as input to the two-layer neural network.

The number of hidden neurons utilized for tests with different dataset sizes m are shown in Table 4, where the selected value of N is (roughly) the smallest round value to satisfy overparameterization assumptions. The hidden dimension of the two-layer network was kept constant between datasets because the CIFAR10 images were downsampled such that the input dimension is relatively similar to MNIST. To improve training stability, we add a sigmoid activation function to the output neuron of the two-layer network and train the network with binary cross entropy loss. The addition of this output activation function slightly complicates the empirical determination of whether $\mathbf{y} \in \mathcal{M}_N$, which is further described in Section B.2. Despite slightly deviating from the setup in Section 2, this modification greatly improves training stability and is more realistic (i.e., classification datasets are not regularly trained with ℓ_2 regression loss as described in Section 2). The network is trained using stochastic gradient descent.[8] We adopt a batch size of one to match the stochastic gradient descent setting exactly and do not use any weight decay.

B.2. Determining Convex Hull Membership

As mentioned in Section 2, we define $\mathbf{y} = \left[y^{(1)}, y^{(2)}, \dots, y^{(m)}\right] / \sqrt{m}$ as a vector containing all labels in our dataset (i.e., in this case, a vector of binary labels). Furthermore, we define $\phi_{i,j} = \sigma(\mathbf{x}^{(j)}, \boldsymbol{\theta}_i)$ as the output of neuron i for example j in the dataset. These values can be concatenated as $\mathbf{\Phi}_i = [\phi_{i,1}, \phi_{i,2}, \dots, \phi_{i,m}] / \sqrt{m}$, forming a vector of output activations for each neuron over the entire dataset. Then, we define $\mathcal{M}_N = \texttt{Conv}\left\{\mathbf{\Phi}_i : i \in [N]\right\}$. It should be noted that the activation vectors $\mathbf{\Phi}_i$ are obtained from the N neurons of a two-layer neural network that has been trained for any number of iterations T. For the experiments in this section, the vectors $\mathbf{\Phi}_i$ are computed after each epoch of training to determine whether $\mathbf{y} \in \mathcal{M}_N$, and we report the number for the first epoch in which this membership is satisfied.

8. Experiments are repeated with multiple learning rates and the best results are reported.

The convex hull membership problem can be formulated as a linear program, and we utilize this formulation to empirically determine when $\mathbf{y} \in \mathcal{M}_N$. In particular, we consider the following linear program.

$$\begin{aligned} \min \mathbf{c}^\top \boldsymbol{\alpha} \\ \text{s.t. } A\boldsymbol{\alpha} = \mathbf{y} \\ Z\boldsymbol{\alpha} = \mathbf{1} \\ \boldsymbol{\alpha} \geq \mathbf{0} \end{aligned} \tag{28}$$

where $\mathbf{c} \in \mathbb{R}^N$ is an arbitrary cost vector, $A = (\mathbf{\Phi}_1 \dots \mathbf{\Phi}_N) \in \mathbb{R}^{m \times N}$, and $Z = (1 \dots 1) \in \mathbb{R}^{1 \times N}$, and $\boldsymbol{\alpha}$ is the simplex being optimized within the linear program. If a viable solution to the linear program in (28) can be found, then it is known that $\mathbf{y} \in \mathcal{M}_N$. Therefore, we empirically solve for the vectors $\mathbf{y}$ and $(\mathbf{\Phi}_i)_{i=1}^N$ using the two-layer neural network after each epoch and use the `linprog` package in SciPy to determine whether (28) is solvable, thus allowing us to determine when $\mathbf{y} \in \mathcal{M}_N$

In our actual implementation, we slightly modify the linear program in (28) to account for the fact that our two-layer neural network is trained with a sigmoid activation on the output neuron. Namely, one can observe that for targets of one, the neural network output can be considered correct if the output value is greater than zero and vice versa. We formulate the following linear program to better reflect this correct classification behavior:

$$\begin{aligned} \min \mathbf{c}^\top \boldsymbol{\alpha} \\ \text{s.t. } A\boldsymbol{\alpha} < \mathbf{0} \\ B\boldsymbol{\alpha} > \mathbf{0} \\ Z\boldsymbol{\alpha} = \mathbf{1} \\ \boldsymbol{\alpha} \geq \mathbf{0} \end{aligned} \tag{29}$$

where $A \in \mathbb{R}^{\frac{m}{2} \times N}$ is a matrix of feature column vectors corresponding to a label of zero, $B \in \mathbb{R}^{\frac{m}{2} \times N}$ is a matrix of feature column vectors corresponding to a label of one, and other variables are defined as in (28). In words, (29) is solvable whenever a simplex can be found to re-weight neuron outputs such that the correct classification is produced for all examples in the dataset. The formulation in (29) better determines whether $\mathbf{y} \in \mathcal{M}_N$ given the modification of our two-layer neural network to solve a binary classification problem.

Appendix C. Experimental Details

C.1. Two-layer Network Experiments

We first provide a clear algorithmic description of the pruning algorithm used within all two-layer network experiments. This algorithm closely reflects the update rule provided in (7). However, instead of measuring loss over the full dataset to perform greedy forward selection, we perform selection with respect to the loss over a single mini-batch to improve the efficiency of the pruning algorithm. We use $f(\mathbf{\Theta}, \boldsymbol{X}')$ to denote the output of a two-layer network network with parameters $\mathbf{\Theta}$ over the mini-batch $\boldsymbol{X}'$ and $f_S(\mathbf{\Theta}, \boldsymbol{X}')$ to denote the same output only considering the neuron indices included within the set $\mathcal{S}$. *Notice that a single neuron can be selected more than once during pruning*.

Algorithm 1 Greedy Forward Selection for Two-Layer Network

```
$N$ := hidden size; $P$ := pruned size; $\mathcal{D}$ := training dataset
$\boldsymbol{\Theta}$ := weights; $\boldsymbol{\Theta}^\star = \varnothing$
$i = 0$
while $i < P$ do
    $j = 0$
    best_idx $= -1$
    $\mathcal{L}^\star = \infty$
    $\boldsymbol{X}', \mathbf{y}' \sim \mathcal{D}$
    while $j < N$ do
        $\mathcal{L}_j = \frac{1}{2}\left(f_{(\boldsymbol{\Theta}^\star \cup j)}(\mathbf{X}') - \mathbf{y}'\right)^2$
        if $\mathcal{L}_j < \mathcal{L}^\star$ then
            $\mathcal{L}^\star = \mathcal{L}_j$
            best_idx $= j$
        end
        $j = j + 1$
    end
    $\boldsymbol{\Theta}^\star = \boldsymbol{\Theta}^\star \cup$ best_idx
    $i = i + 1$
end
return $\boldsymbol{\Theta}^\star$
```

We now present in-depth details regarding the hyperparameters that were selected for the two-layer network experiments in the main text. To tune hyperparameters, we perform a random 80-20 split to generate a validation set. Experiments are repeated three times for each hyperparameter setting, and hyperparameters that yield the best average validation performance are selected.

Models are trained using SGD with momentum of 0.9 and no weight decay. Perturbing weight decay and momentum hyperparameters does not meaningfully impact performance, leading us to maintain this setting in all experiments with two-layer networks. We also use a batch size of 128, which was the largest size that could fit in the memory of our GPU. To determine the optimal learning rate and number of pre-training iterations, we adopt the same setup as described in the Two-layer Network Experiments section in the main text and train two-layer networks of various sizes with numerous different learning rates. These experiments are then replicated across several different sub-dataset sizes. The results of these experiments are shown in Fig. 3. From these results, it can be seen that the optimal learning rate does not change with the size of the dataset, but it does depend on the size of the network. Namely, for two-layer networks with 5K or 10K hidden neurons the optimal learning rate is 1e-5, while for two-layer networks with 20K hidden neurons the optimal learning rate is 1e-6. It can also be seen in Fig. 3 that models converge in roughly 8000 training iterations for all experimental settings, which leads us to adopt this amount of pre-training in the main experiments.

To determine the optimal pruned model size, we first fully pre-train two-layer networks of different hidden sizes (i.e., $N \in \{5K, 10K, 20K\}$) over the full, binarized MNIST dataset. Then, these models are pruned to different hidden neuron sizes between 1 and 500. At each possible hidden

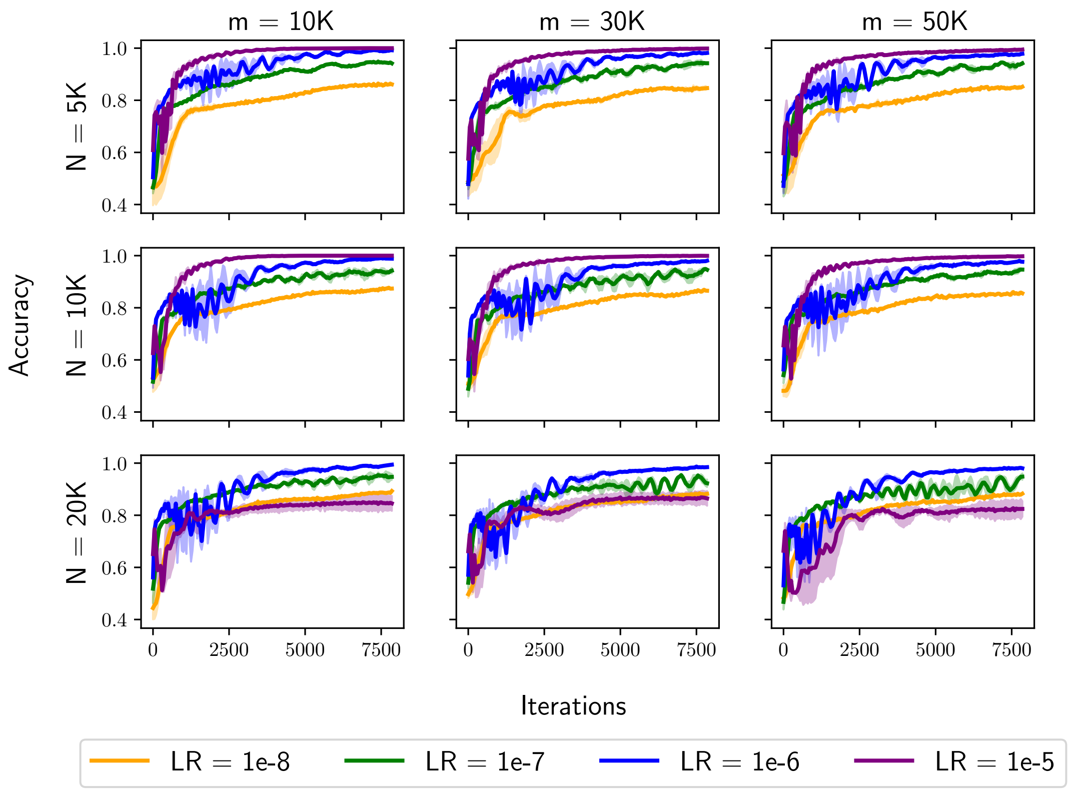

Figure 3: Validation accuracy of two-layer networks on binarized versions of MNIST as described in the Two-layer Network Experiments Section in the main text. Different subplots display results for several different learning rates for models trained with different settings of m and N (i.e., dataset size and number of hidden neurons, respectively). Shaded regions represent standard deviations of results, recorded across three separate trials.

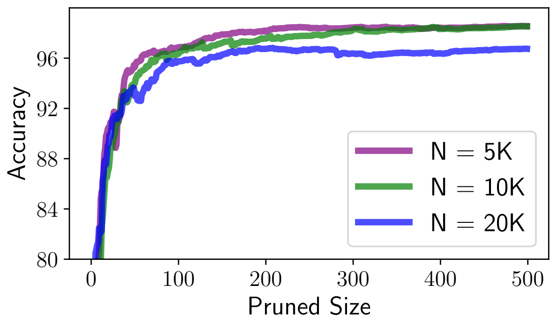

Figure 4: The performance of two-layer networks with different hidden dimensions over the entire, binarized MNIST dataset that have been pruned to various different hidden dimensions.

dimension for the pruned model, we measure the performance of the pruned model over the entire training dataset. The results of this experiment are depicted in Figure 4. As can be seen, the accuracy of the pruned models plateaus at a hidden dimension of 200 (roughly). As a result, we adopt a size of 200 neurons as our pruned model size within all two-layer network experiments.

C.2. CNN Experiments

We begin with an in-depth algorithmic description of the greedy forward selection algorithm that was used for structured, channel-based pruning of multi-layer CNN architectures. This algorithm is identical to the greedy forward selection algorithm adopted in Ye et al. (2020). In this algorithm, we denote the weights of the deep network as as $\mathbf{\Theta}$, and reference the weights within layer ℓ of the network as $\mathbf{\Theta}_{\ell}$. Similarly, we use $\mathbf{\Theta}_{\ell:}$ to denote the weights of all layers following layer ℓ and $\mathbf{\Theta}_{:\ell}$ to denote the weights of all layers up to and including layer ℓ. C denotes a list of hidden sizes within the network, where C_ℓ denotes the number of channels within layer ℓ of the CNN. Again, we use $f(\mathbf{\Theta}, \boldsymbol{X}')$ to denote the output of a two-layer network with parameters $\mathbf{\Theta}$ over the mini-batch $\boldsymbol{X}'$ and $f_{\mathcal{S}}(\mathbf{\Theta}, \boldsymbol{X}')$ to denote the same output only considering the channel indices included within the set $\mathcal{S}$.

Algorithm 2 Greedy Forward Selection for Deep CNN

```
$C$ := hidden sizes; $\epsilon$ := stopping criterion; $\mathcal{D}$ := training dataset
$\mathbf{\Theta}$ := weights; $L$ := # Layers; $\mathbf{\Theta}^\star_\ell = \varnothing \;\; \forall \ell \in [L]$
$\ell = 0$
while $\ell < L$ do
    while convergence criterion is not met do
        $j = 0$
        best_idx $= -1$
        $\mathcal{L}^\star = \infty$
        $\boldsymbol{X}', \mathbf{y}' \sim \mathcal{D}$
        while $j < C_\ell$ do
            $\mathbf{\Theta}' = \mathbf{\Theta}^\star \cup \mathbf{\Theta}_{\ell:} \cup (\mathbf{\Theta}^\star_\ell \cup j)$
            $\mathcal{L}_{\ell,j} = \frac{1}{2}\left(f_{\mathbf{\Theta}'}(\mathbf{X}') - \mathbf{y}'\right)^2$
            if $\mathcal{L}_{\ell,j} < \mathcal{L}^\star$ then
                $\mathcal{L}^\star = \mathcal{L}_{\ell,j}$
                best_idx $= j$
            end
            $j = j + 1$
        end
        $\mathbf{\Theta}^\star_\ell = \mathbf{\Theta}^\star_\ell \cup$ best_idx
    end
    $\ell = \ell + 1$
end
return $\mathbf{\Theta}^\star$
```

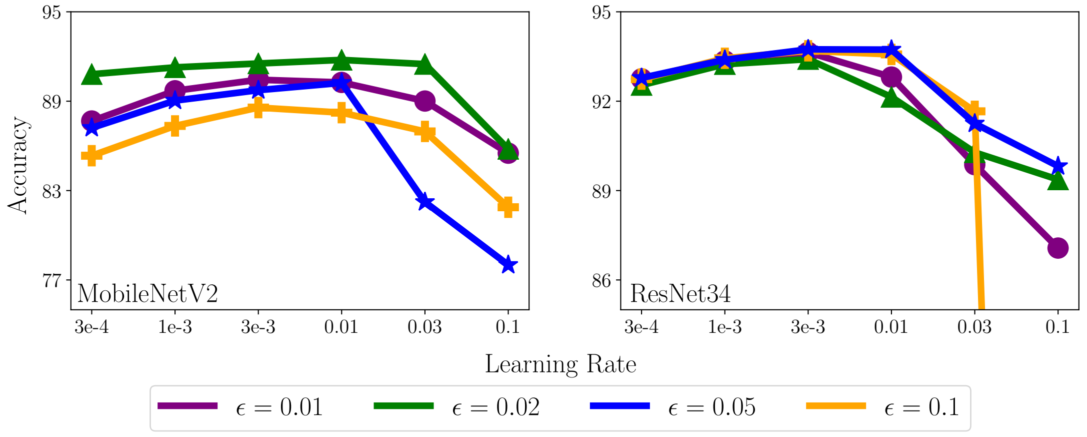

Figure 5: Subnetwork validation accuracy on the CIFAR10 dataset for different settings of ϵ and initial learning rate for fine-tuning. All models are pre-trained identically for 200 epochs. Fine-tuning is performed for 80 epochs, and we report validation accuracy for each subnetwork at the end of fine-tuning.

Now, we present more details regarding the hyperparameters that were utilized within large-scale experiments. For ImageNet experiments, we adopt the settings of Ye et al. (2020).[9] For CIFAR10, however, we tune both the setting of ϵ and the initial learning rate for fine-tuning using a grid search for both MobileNetV2 and ResNet34 architectures. This grid search is performed using a validation set on CIFAR10, constructed using a random 80-20 split on the training dataset. Optimal hyperparameters are selected based on their performance on the validation set. The results of this grid search are shown in Figure 5. As can be seen, for MobileNetV2, the best results are achieved using a setting of $\epsilon = 0.02$, which results in a subnetwork with 60% of the FLOPS of the dense model. Furthermore, an initial learning rate of 0.01 yields consistent subnetwork performance for MobileNetV2. For ResNet34, a setting of $\epsilon = 0.05$ yields the best results and yields a subnetwork with 60% of the FLOPS of the dense model. Again, an initial learning rate of 0.01 for fine-tuning yields the best results for ResNet34. For the rest of the hyperparameters used within CIFAR10 experiments (i.e., those used during pre-training), we adopt the settings of a widely-used, open-source repository that achieves good performance on CIFAR10 Liu (2017).

9. We adopt all of the same experimental settings, but decrease the number of fine-tuning epochs from 150 to 80 because we find that testing accuracy reaches a plateau well-before 150 epochs.