Proceedings of Machine Learning Research vol 144:1–27, 2022

i-SpaSP: Structured Neural Pruning via Sparse Signal Recovery

Cameron R. Wolfe CRW13@RICE.EDU
Department of Computer Science, Rice University, Houston, TX, USA.

Anastasios Kyrillidis ANASTASIOS@RICE.EDU
Department of Computer Science, Rice University, Houston, TX, USA.

Abstract

We propose a novel, structured pruning algorithm for neural networks—the **i**terative, **Spa**rse **S**tructured **P**runing algorithm, dubbed as i-SpaSP. Inspired by ideas from sparse signal recovery, i-SpaSP operates by iteratively identifying a larger set of important parameter groups (e.g., filters or neurons) within a network that contribute most to the residual between pruned and dense network output, then thresholding these groups based on a smaller, pre-defined pruning ratio. For both two-layer and multi-layer network architectures with ReLU activations, we show the error induced by pruning with i-SpaSP decays polynomially, where the degree of this polynomial becomes arbitrarily large based on the sparsity of the dense network's hidden representations. In our experiments, i-SpaSP is evaluated across a variety of datasets (i.e., MNIST and ImageNet) and architectures (i.e., feed forward networks, ResNet34, and MobileNetV2), where it is shown to discover high-performing sub-networks and improve upon the pruning efficiency of provable baseline methodologies by several orders of magnitude. Put simply, i-SpaSP is easy to implement with automatic differentiation, achieves strong empirical results, comes with theoretical convergence guarantees, and is efficient, thus distinguishing itself as one of the few computationally efficient, practical, and provable pruning algorithms.

Keywords: Lottery Ticket Hypothesis, Neural Network Pruning, Sparse Signal Recovery
Source Code: https://github.com/wolfecameron/i-SpaSP

1. Introduction

Background. Neural network pruning has garnered significant recent interest (Frankle and Carbin, 2018; Liu et al., 2018; Li et al., 2016), as obtaining high-performing sub-networks from larger, dense networks enables a reduction in the computational and memory overhead of neural network applications (Han et al., 2015a,b). Many popular pruning techniques are based upon empirical heuristics that work well in practice (Frankle et al., 2019; Luo et al., 2017; He et al., 2017). Generally, these methodologies introduce some notion of "importance" for network parameters (or groups of parameters) and eliminate parameters with negligible importance.

The empirical success of pruning methodologies inspired the development of pruning algorithms with theoretical guarantees (Baykal et al., 2019; Liebenwein et al., 2019; Mussay et al., 2019; Baykal et al., 2018; Ramanujan et al., 2019). Among such work, greedy forward selection (GFS) (Ye et al., 2020; Ye et al., 2020)—inpired by the Frank-Wolfe algorithm (Frank et al., 1956)—differentiated itself as a methodology that performs well in practice and provides theoretical guarantees. However, the inefficiency of GFS makes it less practical in comparison to popular pruning heurustics.

This Work. Numerous greedy selection algorithms exist beyond Frank-Wolfe that are efficient and can be applied to neural network pruning (Needell and Tropp, 2009; Khanna and Kyrillidis, 2018).

© 2022 C.R. Wolfe & A. Kyrillidis.

Within this work, we draw upon methods from sparse signal recovery (Needell and Tropp, 2009) to develop a structured pruning algorithm that is provable, practical, and efficient. This algorithm, which we call **i**terative, **Spa**rse **S**tructured **P**runing (i-SpaSP), iteratively estimates the most important parameter groups[1] within each layer based on the residual between pruned and dense network output, then thresholds this set of parameter groups based on a pre-defined pruning ratio.

i-SpaSP is simple to implementand significantly improves upon the runtime of GFS variants. From a theoretical perspective, we show for two-layer networks that i) the difference in output between the dense network and a pruned network obtained with i-SpaSP decays polynomially with respect to the size of the pruned network and ii) the order of this polynomial increases as the dense network's hidden representations become more sparse. We also extend this analysis to multi-layer networks. To the best of the authors' knowledge, we are the first to provide theoretical analysis showing that the quality of pruning depends upon the sparsity of representations within the dense network. In experiments, we show that i-SpaSP is capable of discovering high-performing sub-networks, which match or exceed the performance of both heuristic and provable baselines, across numerous different models (i.e., two-layer networks, ResNet34, and MobileNetV2) and datasets (i.e., MNIST and ImageNet).

2. Preliminaries

Notation. Vectors and scalars are denoted with lower-case letters and distinguished by context. Matrices and certain constants are denoted with upper-case letters and distinguished by context. Sets are denoted with upper-case, calligraphic letters (e.g., $\mathcal{G}$) with set complements $\mathcal{G}^c$. $[n]$ denotes the set $\{0, 1, \ldots, n\}$. Consider vector $x \in \mathbb{R}^N$. $\|x\|_p$ denotes an ℓ_p vector norm. x_s denotes the s largest-valued components of x. $\text{supp}(x)$ returns the support of x. For index set $\mathcal{G}$, we define $x|_{\mathcal{G}}$ as the vector with non-zeros at the indices in $\mathcal{G}$. Consider a matrix $X \in \mathbb{R}^{m \times n}$. $\|X\|_F$ and $X^\top$ represent the Frobenius norm and transpose of X. $\cdot$ denotes a matrix product. The i-th row and j-th column of X are given by $X_{i,:}$ and $X_{:,j}$, respectively. $\text{rsupp}(X)$ returns the row support of X (i.e., indices of non-zero rows). For index set $\mathcal{G}$, $X_{\mathcal{G},:}$ and $X_{:,\mathcal{G}}$ represent row and column sub-matrices, respectively, that contain rows or columns with indices in $\mathcal{G}$. $\mu(X) : \mathbb{R}^{m \times n} \longrightarrow \mathbb{R}^m$ sums over columns of a matrix (i.e., $\mu(X) = \sum_i X_{:,i}$), while $\texttt{vec}(X) : \mathbb{R}^{m \times n} \longrightarrow \mathbb{R}^{mn}$ generates a vector by stacking columns of a matrix.

Network Architecture. Our analysis primarily considers two-layer, feed forward networks:

$$f(X, \mathcal{W}) = W^{(1)} \cdot \sigma(W^{(0)} \cdot X) \tag{1}$$

The network's input, hidden, and output dimensions are given by d_{in}, d_{hid}, and d_{out}. $X \in \mathbb{R}^{d_{in} \times B}$ stores the full input dataset with B examples. $W^{(0)} \in \mathbb{R}^{d_{hid} \times d_{in}}$ and $W^{(1)} \in \mathbb{R}^{d_{out} \times d_{hid}}$ denote the network's weight matrices. $\sigma(\cdot)$ denotes the ReLU activation function and $H = \sigma(W^0 \cdot X)$ stores the network hidden representations across the dataset. We also extend our analysis to multi-layer networks with similar structure; see Appendix B.5 for more details.

Compressible Matrices. Given $X \in \mathbb{R}^{m \times n}$, a p-row-compressible matrix satisfies the following: $|\mu(X)|_{(i)} \leq \frac{R}{i^{\frac{1}{p}}}, \ \ \forall i \in [m]$, where $|\cdot|_{(i)}$ denotes the i-th sorted vector component (in magnitude). Lower p values indicate that a matrix is nearly row-sparse and vice versa. We consider

1. "Parameter groups" refers to the minimum structure used for pruning (e.g., neurons or filters).

row-compressibility within the dense network's hidden representation H and find that pruning performance improves as p decreases. Although network sparsity is not typically considered in theoretical pruning analysis, such a relationship is sensible, as pruning effectiveness is trivially dependent upon the existence of unimportant or redundant information within the network's hidden representation.

3. Related Work

Pruning. Pruning strategies for neural networks can be roughly separated into structured (Han et al., 2016; Li et al., 2016; Liu et al., 2017; Ye et al., 2020; Ye et al., 2020) and unstructured (Evci et al., 2019, 2020; Frankle and Carbin, 2018; Han et al., 2015) variants. Structured pruning, as considered in this work, prunes parameter groups instead of individual weights, allowing speedups to be achieved without sparse computation (Li et al., 2016). Many empirical heuristics for structured pruning exist; e.g., pruning parameter groups with low ℓ_1 norm (Li et al., 2016; Liu et al., 2017), measuring the gradient-based sensitivity of parameter groups (Baykal et al., 2019; Wang et al., 2020; Zhuang et al., 2018), preserving network output (He et al., 2017; Luo et al., 2017; Yu et al., 2017), and more (Suau et al., 2020; Chin et al., 2019; Huang and Wang, 2018; Molchanov et al., 2016). Pruning typically follows a three-step process of pre-training, pruning, and fine-tuning (Li et al., 2016; Liu et al., 2018). Pre-training is usually the most expensive component, but later work explores strategies of finding good pruned networks with minimal pre-training (You et al., 2019; Chen et al., 2020).

Provable Pruning. Empirical pruning research inspired the development of theoretical foundations for network pruning, including sensitivity-based analysis (Baykal et al., 2019; Liebenwein et al., 2019), coreset methodologies (Mussay et al., 2019; Baykal et al., 2018), and pruning analysis of random networks (Malach et al., 2020; Orseau et al., 2020; Pensia et al., 2020; Ramanujan et al., 2019). Later work analyzed pruned network generalization (Zhang et al., 2021) and the amount of dense network pre-training needed to obtain high-performing sub-networks (Wolfe et al., 2021). GFS—originally proposed for two-layer networks (Ye et al., 2020) and subsequently extended to multi-layer networks (Ye et al., 2020)—was one of the first theoretical pruning works that proposed an associated pruning methodology that outperforms heuristic pruning methods.

Greedy Selection. Greedy selection efficiently discovers approximate solutions to combinatorial optimization problems (Frank et al., 1956). Numerous algorithms and frameworks for greedy selection exist; e.g., Frank-Wolfe (Frank et al., 1956), sub-modular optimization (Nemhauser et al., 1978), CoSAMP (Needell and Tropp, 2009), and iterative hard thresholding (Khanna and Kyrillidis, 2018). GFS (Ye et al., 2020; Ye et al., 2020) uses a Frank-Wolfe-style algorithm to achieve good pruning performance in theory and practice. In a similar vein, Frank-Wolfe was shown capable of training deep neural networks (Bach, 2014; Pokutta et al., 2020), thus solidifying the connection between greedy selection and deep learning. Extending this connection, we leverage the CoSAMP (Needell and Tropp, 2009) algorithm in formulating our proposed methodology.

4. Methodology

i-SpaSP is formulated for two-layer networks in Algorithm 1, where the pruned model size s and number of iterations T are fixed. $\mathcal{S}$ stores the set of active neurons within the pruned model. This set is iteratively selected and thresholded based on importance values Y and hidden activation values H, respectively, to ultimately generate the sub-network that best approximates dense network output.

Algorithm 1 i-SpaSP for Two-Layer Networks

Parameters: T, s; $\mathcal{S} := \emptyset$; $t := 0$

compute hidden representation
$H = \sigma(W^{(0)} \cdot X)$
$h = \mu(H)$

compute dense network output
$U = W^{(1)} \cdot H$
$V = U$

while $t < T$ **do**
  $t = t + 1$

  # **Step I:** Estimating Importance
  $Y = (W^{(1)})^\top \cdot V$
  $y = \mu(Y)$
  $\Omega = \text{supp}(y_{2s})$

  # **Step II:** Merging and Pruning
  $\Omega^\star = \Omega \cup \mathcal{S}$
  $b = h|_{\Omega^\star}$
  $\mathcal{S} = \text{supp}(b_s)$

  # **Step III:** Computing New Residual
  $V = U - W^{(1)}_{:,\mathcal{S}} \cdot H_{\mathcal{S},:}$

end

return pruned model with neurons in $\mathcal{S}$
`return` $\{W^{(0)}_{\mathcal{S},:}, W^{(1)}_{:,\mathcal{S}}\}$

Why does this work? Each iteration of i-SpaSP follows a three-step procedure in Algorithm 1:

Step I: Compute neuron "importance" Y given the current residual matrix V.

Step II: Identify s neurons within the combined set of important and active neurons (i.e., $\Omega \cup \mathcal{S}$) with the largest-valued hidden representations.

Step III: Update V with respect to the new pruned model estimate.

We now provide intuition regarding the purpose of each individual step within i-SpaSP.

Estimating Importance. A single entry of the importance matrix Y_{ij} represents the importance of hidden neuron i with respect to dataset example j. Denoting pruned and dense network output as U and U', respectively, the following objective characterizes the discrepancy between U and U':

$$\mathcal{L}(U, U') = \frac{1}{2}\|W^{(1)} \cdot H - U'\|_F^2. \quad (2)$$

Considering U' to be fixed, $\nabla_H \mathcal{L}(U, U') = (W^{(1)})^\top \cdot V$. As such, if Y_{ij} is a large, positive (negative) value, decreasing (increasing) H_{ij} will decrease the value of $\mathcal{L}$ locally and, in turn, reduce the disparity between pruned and dense network output. Going further, because values within H are non-negative and cannot be modified by the pruning algorithm, one can realize that the best methodology of minimizing (2) is including neurons with large, positive importance values within the active set, as in Algorithm 1.

Merging and Pruning. The $2s$ most-important neurons—determined by component values in $\mu(Y)$—are selected, allowing a larger set of neurons (i.e., more than s) to be explored during the pruning process. From here, s neurons with the largest-valued components in $\mu(H)$ are sub-selected from this combined set, forming the next pruned model estimate. Hidden representation values are not considered within importance estimation, meaning that important neurons may have all zero hidden activations. Thus, performing this two-step merging and pruning process based upon both $\mu(Y)$ and $\mu(H)$ ensures neurons within the active set are important and have large hidden activations, properties that together indicate a meaningful impact on network output.

Computing the New Residual. Once the next pruned model estimate is derived, it is used to recompute V, which can be intuitively seen as updating U' in (2). As such, each iteration of Algorithm 1 computes importance based on the current disparity between pruned and dense network output, thus successively minimizing the value of (2).

4.1. Implementation

```
H.requires_grad := True
with torch.no_grad():
    prune_out := prune_layer(H_{S,:})
dense_out := dense_layer(H)
obj := sum((1/2)(dense_out − prune_out)^2)
obj.backward()
importance := sum(H.grad, dim = 0)
return importance
```

Algorithm 2: i-SpaSP importance computation via automatic differentiation.

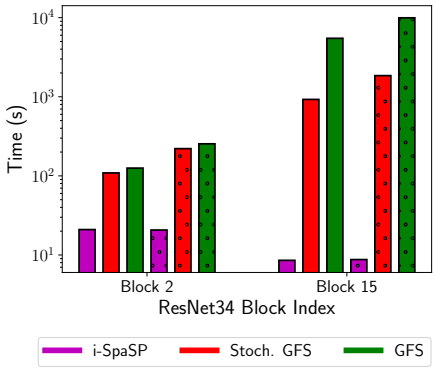

Figure 1: Runtime of pruning ResNet34 blocks with i-SpaSP and GFS variants to 20% (i.e., plain) or 40% (i.e., dotted) of filters.

Implementation with Automatic Differentiation. Because $Y = \nabla_H \mathcal{L}(U, U')$, importance within i-SpaSP can be computed efficiently using automatic differentiation (Paszke et al., 2017; Abadi et al., 2015); see Algorithm 2 for a PyTorch-style example. Automatic differentiation simplifies importance estimation and allows it to be run on a GPU, making the implementation efficient and parallelizable. Because the remainder of the pruning process only leverages basic sorting and set operations, the overall implementation of i-SpaSP is both simple and efficient.

Other Architectures. Algorithm 2 can be easily generalized to network modules beyond feed forward layers (i.e., automatic differentiation is agnostic to the underlying network module). Notably, convolutional filters can be pruned using the importance estimation from Algorithm 2 if $\texttt{sum}(\cdot)$ is performed over both batch and spatial dimensions. Furthermore, i-SpaSP can be used to prune multi-layer networks by greedily pruning each layer of the network from beginning to end.

Large-Scale Datasets. Algorithm 1 assumes the entire dataset is stored within X. For large-scale experiments, such an approach is not tractable. As such, we redefine X within experiements to contain a subset or mini-batch of data from the full dataset, allowing the pruning process to be performing in an approximate—but computationally tractable—manner. To improve the quality of this approximation, a new mini-batch is sampled during each i-SpaSP iteration, but the size of such mini-batches becomes a hyperparameter of the pruning process.[2]

4.2. Computational Complexity

Denote the complexity of matrix-matrix multiplication as ξ. The complexity of pruning a network layer with T iterations of i-SpaSP is $\mathcal{O}(T\xi + Td_{hid}\log(d_{hid}))$. In comparison, GFS (Ye et al., 2020; Ye et al., 2020) has a complexity of $\mathcal{O}(s\xi d_{hid})$, as it adds a single neuron to the (initially empty) network layer at each iteration by exhaustively searching for the neuron that minimizes training loss. Later GFS variants improve this asymptotic complexity to $\mathcal{O}(s\xi)$ (Ye et al., 2020). Yet, i-SpaSP is more efficient in practice because the forward pass (i.e. $\mathcal{O}(\xi)$) dominates the pruning procedure and $T \ll s$ (e.g., $T = 20$ in large-scale experiments; see Section 6).

2. Both GFS (Ye et al., 2020) and multi-layer GFS (Ye et al., 2020) adopt a similar mini-batch approach during pruning.

Towards a more practical comparison, we adopt a ResNet34 model and measure the wall-clock time of pruning select blocks[3] using both i-SpaSP and GFS. We use the public implementation of GFS (Ye, 2021) and test both stochastic and vanilla variants[4] using a batch size of 256 (i.e., this matches the setting of (Ye et al., 2020)). i-SpaSP uses a batch size of 1280 to match settings in Section 6, and blocks are pruned to ratios of 20% or 40% of original filters; see Figure 1. i-SpaSP significantly improves upon the runtime of GFS variants; e.g., i-SpaSP prunes Block 15 in roughly 10 seconds, while GFS takes over 1000 seconds in the best case. Furthermore, unlike GFS, the runtime of i-SpaSP is not sensitive to the size of the pruned network (i.e., wall-clock time is similar for ratios of 20% and 40%), though i-SpaSP does prune later network layers faster than earlier layers.

5. Theoretical Results

Proofs are deferred to Appendix B. The dense network contains d_{hid} hidden neurons and is pruned to size s via i-SpaSP. For all results, we assume that $W^{(1)}$ satisfies the restricted isometry property (RIP) (Candes and Tao, 2006), defined as follows:

Assumption 1 (Restricted Isometry Property (RIP) (Candes and Tao, 2006)) *Denote the r-th restricted isometry constant as δ_r and assume $\delta_r \leq 0.1$ for $r = 4s$.[5] Then, $W^{(1)}$ satisfies the RIP with constant δ_r, when the following holds for all $\|x\|_0 \leq r$:*

$$(1 - \delta_r)\|x\|_2^2 \leq \|W^{(1)} \cdot x\|_2^2 \leq (1 + \delta_r)\|x\|_2^2.$$

No assumption is made upon $W^{(0)}$. We define the two-layer network hidden representation as $H = \sigma(W^{(0)} \cdot X) \in \mathbb{R}^{d_{hid} \times B}$, which can be reformulated as $H = Z + E$ for s-row-sparse Z and arbitrary E. From here, we can show the following about the residual between pruned and dense network hidden representations after t iterations of Algorithm 1.

Lemma 1 *If Assumption 1 holds, the pruned approximation to H after t iterations of Algorithm 1, $H_{\mathcal{S}_t,:}$, is s-row-sparse and satisfies the following inequality:*

$$\|\mu(H - H_{\mathcal{S}_t,:})\|_2 \leq (0.444)^t \|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(E)\|_1$$

Going further, Lemma 1 can be invoked to bound the residual between pruned and dense network output after t iterations of Algorithm 1.

Theorem 2 *Let $U = W^{(1)} \cdot H$ and $U' = W^{(1)}_{:,\mathcal{S}_t} \cdot H_{\mathcal{S}_t,:}$ denote pruned and dense network output, respectively. $V_t = U - U'$ stores the residual between pruned and dense network output over the entire dataset. If Assumption 1 holds, we have the following at iteration t of Algorithm 1:*

$$\|V_t\|_F \leq \|W^{(1)}\|_F \cdot \left((0.444)^t \|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(E)\|_1\right)$$

3. We prune the 2nd and 15th convolutional blocks, which have channel dimensions to 64 and 512, respectively, and are not strided. We choose blocks within drastically different regions of the network to view the impact of channel and spatial dimension on pruning efficiency.
4. Vanilla GFS exhaustively searches over all neurons within the hidden layer during each iteration of GFS, while the stochastic variant randomly chooses a smaller set of 50 neurons over which to search during each pruning iteration.
5. This is a numerical assumption adopted from Needell and Tropp (2009), which holds for Gaussian matrices of size $\mathbb{R}^{m \times n}$ when $m \geq \mathcal{O}\left(r \log(\frac{n}{r})\right)$.

Because the $\|\mu(H)\|_2$ decays linearly in Theorem 2, $\|\mu(E)\|_1$ will dominate the residual between pruned and dense network output after sufficient iterations of Algorithm 1. By assuming H is row-compressible (i.e., see Section 2), we can derive a bound on $\|\mu(E)\|_1$.

Lemma 3 *Assume H is p-row-compressible with factor R, where $H = Z + E$ for s-row-sparse Z and arbitrary E. Then, $\|\mu(E)\|_1$ obeys the following: $\|\mu(E)\|_1 \leq R \cdot \frac{s^{1-\frac{1}{p}}}{\frac{1}{p}-1}$.*

From here, Lemma 3 can be combined with Theorem 2 to yield a final bound on the residual between pruned and dense network output.

Theorem 4 *Assume Algorithm 1 is run for a sufficiently large number of iterations t. If Assumption 1 holds and H is p-row-compressible with factor R, the output residual between the dense network and the pruned network discovered via i-SpaSP can be bounded as follows:*

$$\|V_t\|_F \leq \mathcal{O}\left(\frac{s^{\frac{1}{2}-\frac{1}{p}} p(2\sqrt{s}+1)}{1-p}\right).$$

Proof *This follows directly from substituting Lemma 3 into Theorem 2, assuming t is large enough such that $(0.444)^t \|W^{(1)}\|_F \|\mu(H)\|_2 \approx 0$, and factoring out constants in the resulting expression.* ■

Theorem 4 indicates that the quality of a pruned network obtained via i-SpaSP is dependent upon i) its size s and ii) the row-compressibility factor p of the dense network's hidden representation H. Intuitively, one would expect that lower values of p would make pruning easier, as neurons corresponding to zero rows in H could be eliminated without consequence. This trend is observed exactly within Theorem 4; e.g., for $p = \{\frac{3}{4}, \frac{1}{2}, \frac{1}{4}\}$ we have $\|V_t\|_F \leq \{\mathcal{O}(s^{-\frac{1}{3}}), \mathcal{O}(s^{-1}), \mathcal{O}(s^{-3})\}$, respectively. *To the best of the authors' knowledge, our work is the first to theoretically characterize pruning error with respect to sparsity properties of network hidden representations.*

The bound in Theorem 4 can also be extended to similarly-structured, multi-layer networks; see Appendix B.5 for more details.

Theorem 5 *Consider an L-hidden-layer network with weight matrices $\{W^{(0)}, \ldots, W^{(L)}\}$ and hidden representations $\{H^{(1)}, \ldots H^{(L)}\}$. We define $H^{(\ell)} = \sigma(W^{(\ell)} \cdot H^{(\ell-1)})$, where $H^{(1)} = \sigma(W^{(0)} \cdot X)$. No ReLU activation is applied to network output (i.e., $H^{(L)} = W^{(L)} \cdot H^{(L-1)}$). We assume all weight matrices other than $W^{(0)}$ obey Assumption 1 and all hidden representations other than $H^{(L)}$ are p-row-compressible. i-SpaSP is applied greedily to prune each layer within the network, in layer order, from d to s hidden neurons. Given a sufficient number of iterations t, the residual between pruned and dense multi-layer network output can be bounded as follows:*

$$\|V_t^{(L)}\|_F \leq \mathcal{O}\left(\sum_{i=1}^{L}\left(14+\frac{7}{\sqrt{s}}\right)^{L-i+1}\left(\|W^{(L)}\|_F \prod_{j=1}^{L-i}\|\texttt{vec}(W^{(j)})\|_1\right)\left(\frac{d^{\frac{L-i}{2}} s^{1-\frac{1}{p}}}{\frac{1}{p}-1}\right)\right) \quad (3)$$

Theorem 5 differs from Theorem 4 because pruning error must be summed over and propagated through network layers. Pruning error in (3) is summed over each network layer. Considering a single layer i within the sum, the green factor is inherited from Theorem 2 with an added exponent

due to a recursion over network layers after i. Similarly, the blue factor accounts for propagation of error through weight matrices after layer i, *revealing that green and blue factors account for propagation of error through network layers*. The red portion of (3), which captures the convergence properties of multi-layer pruning, comes from Lemma 3, where an extra factor $d^{\frac{L-i}{2}}$ arises as an artifact of the proof.[6] Though not desirable, this extra factor disappears—leading the overall expression to converge—for sufficiently small p (i.e., note that d is a constant multiple of s based on the pruning ratio). For example, if $p = \frac{1}{4}$, the red expression in (3) behaves asymptotically as $\{\mathcal{O}(s^{-2.5}), \mathcal{O}(s^{-2}), \mathcal{O}(s^{-1.5}), \dots\}$ for layers at the end of the network moving backwards.

6. Experiments

Within this section, we provide empirical analysis of i-SpaSP. We first present synthetic results using two-layer neural networks to numerically verify Theorem 4. Then, we perform experiments with two-layer networks on MNIST (Deng, 2012) and with convolutional neural networks (CNNs) on ImageNet (ILSVRC2012). For all experiments, we adopt best practices from previous work (Li et al., 2016) to determine pruning ratios within the dense network, often performing less pruning on sensitive layers; see Appendix A for details. As baselines, we adopt both greedy selection methodologies (Ye et al., 2020; Ye et al., 2020) and several common, heuristic methods. We find that, in addition to improving upon the pruning efficiency of GFS, i-SpaSP achieves comparable performance to baselines in all cases, demonstrating that it is both performant and efficient.

6.1. Synthetic Experiments

To numerically verify Theorem 4, we construct synthetic H matrices with different row-compressibility ratios p, denoted as $\mathcal{H} = \{H^{(i)}\}_{i=1}^{K}$. For each ratio $p \in \{0.3, 0.5, 0.7, 0.9\}$, we randomly generate three unique entries within $\mathcal{H}$ (i.e., $K = 12$) and present average performance across them all. We prune a randomly-initialized output weight matrix $W^{(1)}$ from size $d_{hid} \times d_{out}$ to sizes $s \times d_{out}$ for $S \in [d_{hid}]$. (i.e., each setting of s constitutes a separate pruning experiment) using $T = 20$.[7]. Two different hidden dimensions are tested (i.e., $d_{hid} \in \{100, 200\}$), and the same $W^{(1)}$ matrix is used for experiments with equal hidden dimension; see Appendix A.1 for more details.

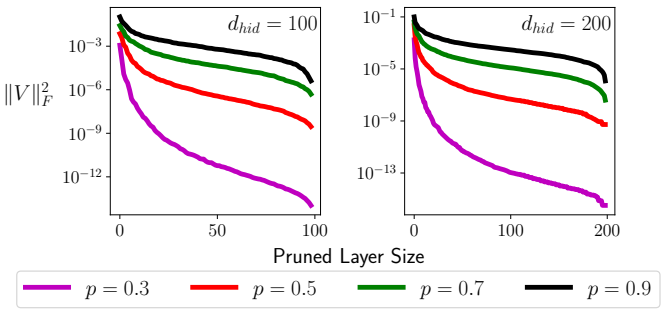

Figure 2: i-SpaSP pruning experiments for two-layer networks of different sizes and p ratios. *The error decay rate increases with decreasing p, confirming that more compressible hidden representations aid in the pruning process.*

Results are displayed in Figure 2, where $\|V\|_F^2$ is shown to decay polynomially with respect to the number of neurons within the pruned network s. Furthermore, as predicted by Theorem 4, the

6. This artifact arises because ℓ_1 norms must be replaced with ℓ_2 norms in certain areas to form a recursion over network layers. This artifact can likely be removed by leveraging sparse matrix analysis for expander graphs (e.g., see Section 5.2.1 in Bah and Tanner (2018)), but we leave this as future work as to avoid over-complicating the analysis.
7. We find that the active set of neurons selected by i-SpaSP becomes stable (i.e., none or very few neurons are modified within the active set) after 20 iterations or less in all synthetic experiments.

decay rate of $\|V\|_F^2$ increases as p decreases, revealing that higher levels of sparsity within the dense network's hidden representations improves pruning performance and speed with respect to s. This trend holds for all hidden dimensions that were considered, revealing that the behavior of i-SpaSP is indeed dependent upon the row-compressibility factor p.

6.2. Two-Layer Networks

Here, we perform pruning experiments with two-layer networks on MNIST (Deng, 2012). All MNIST images are flattened and no data augmentation is used. The dense network has 10×10^3 hidden neurons and is fully pre-trained prior to pruning. We prune the dense network using i-SpaSP, GFS, and Top-K, which we use as a naive greedy selection baseline.[8] After pruning, the network is fine-tuned for 50 epochs using stochastic gradient descent (SGD) with momentum. Performance is reported as an average across three separate trails; see Appendix A.2 for more details.

Results are provided in Figure 3. As shown in Figure 3-right, i-SpaSP outperforms other pruning methodologies after fine-tuning in all experimental settings. Networks obtained with GFS perform well without fine-tuning (i.e., Figure 3-left) because neurons are selected to minimize loss during the pruning process. Because i-SpaSP selects neurons based upon the importance criteria described in Section 4, the pruning process does not directly minimize training loss, thus leading to poorer performance prior to fine-tuning (i.e., Top-K exhibits similar behavior). Nonetheless, i-SpaSP, in addition to improving upon the pruning efficiency of GFS, discovers a set of neurons that more closely recovers dense network output, as revealed by its superior performance after fine-tuning.

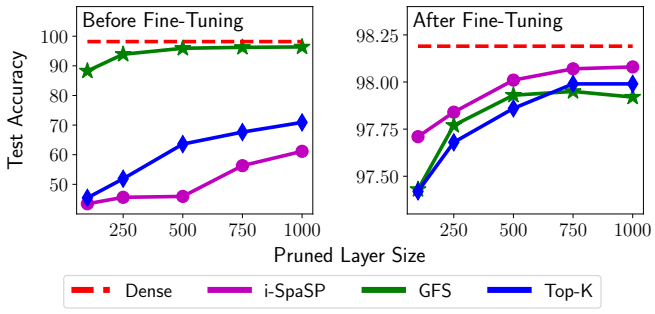

Figure 3: Performance of networks pruned with different greedy algorithms on MNIST before (left) and after (right) fine-tuning. Although GFS performs well prior to fine-tuning, *i-SpaSP always yields the top-performing network after fine-tuning.*

6.3. Deep Networks

We perform structured filter pruning of ResNet34 (He et al., 2015b) and MobileNetV2 (Sandler et al., 2018) with i-SpaSP on ImageNet (ILSVRC 2012). Beginning with public, pre-trained models (Paszke et al., 2017), we use i-SpaSP to prune chosen convolutional blocks within each network, then fine-tune the model for 90 epochs using SGD with momentum. After pruning each block, we run fine-tuning for one epoch; see Appendix A.3 for more details. Numerous heuristic and greedy selection-based algorithms are adopted as baselines; see Table 1.

ResNet34. We prune ResNet34 to 2.69 and 2.13 GFlops with i-SpaSP. As shown in Table 1, i-SpaSP yields comparable performance to GFS variants at similar FLOP levels; e.g., i-SpaSP matches 75.5% test accuracy of GFS with the 2.69 GFlop model and performs within 1% of GFS variants for the 2.13

8. Top-k selects k neurons with the largest-magnitude hidden representations in a mini-batch of data. It is a naive baseline for greedy selection that is not used within previous work to the best of our knowledge.

GFlop model. However, the multi-layer variant of GFS (Ye et al., 2020) does discover sub-networks with fewer FLOPS and similar performance in both cases. In comparison to heuristic methods, i-SpaSP improves upon or matches the performance of all baselines at similar FLOP levels.

MobileNetV2. We prune MobileNetV2 to 260, 242, and 220 MFlops with i-SpaSP. In all cases, sub-networks discovered with i-SpaSP achieve performance within 1% of those obtained via both GFS variants and heuristic methodologies. Although performance on MobileNetV2 is relatively lower in comparison to ResNet34, i-SpaSP is still capable of pruning the more difficult network to various different FLOP levels and performs comparably to baselines in all cases.

Discussion. Within Table 1, the performance of i-SpaSP is never more than 1% below that of a similar-FLOP model obtained with a baseline pruning methodology, including both greedy selection and heuristic-based methods. As such, similar to GFS, i-SpaSP can be seen as a theoretically-grounded pruning methodology that is practically useful, even in large-scale experiments. Such pruning methodologies that are both theoretically and practically relevant are few. In comparison to GFS variants, i-SpaSP significantly improves pruning efficiency; see Section 4.2. Thus, i-SpaSP can be seen as a viable alternative to GFS—both theoretically and practically—that may be preferable when runtime is a major concern.

	Pruning Method	Top-1 Acc.	FLOPS
ResNet34	Full Model	73.4	3.68G
	Filter PruningLi et al. (2016)	72.1	2.79G
	Rethinking Pruning Liu et al. (2018)	72.0	2.79G
	More is Less Dong et al. (2017)	73.0	2.75G
	i-SpaSP	73.5	2.69G
	GFS Ye et al. (2020)	73.5	2.64G
	Multi-Layer GFS Ye et al. (2020)	73.5	2.20G
	SFP He et al. (2018a)	71.8	2.17G
	FPGM He et al. (2019)	72.5	2.16G
	i-SpaSP	72.5	2.13G
	GFS Ye et al. (2020)	72.9	2.07G
	Multi-Layer GFS Ye et al. (2020)	73.3	1.90G
MobileNetV2	Full Model	72.0	314M
	i-SpaSP	71.6	260M
	GFS Ye et al. (2020)	71.9	258M
	Multi-Layer GFS Ye et al. (2020)	72.2	245M
	i-SpaSP	71.3	242M
	LEGR Chin et al. (2019)	71.4	224M
	Uniform	70.0	220M
	AMC He et al. (2018b)	70.8	220M
	i-SpaSP	70.7	220M
	GFS Ye et al. (2020)	71.6	220M
	Multi-Layer GFS Ye et al. (2020)	71.7	218M
	Meta Pruning Liu et al. (2019)	71.2	217M

Table 1: Test accuracy of ResNet34 and MobileNetV2 models pruned to different FLOP levels with various pruning algorithms on ImageNet.

7. Conclusion

We propose i-SpaSP, a pruning methodology for neural networks inspired by sparse signal recovery. Our methodology comes with theoretical guarantees that indicate, for both two and multi-layer networks, the quality of a pruned network decays polynomially with respect to its size. Going further, we connect this theoretical analysis with the sparsity within the dense network's hidden layer, showing that pruning performance improves as the dense network becomes more sparse. Practically, i-SpaSP performs comparably to numerous baseline pruning methodologies in large-scale experiments and drastically improves the computationally efficiency of the most common provable pruning methodologies. As such, i-SpaSP is a practical, provable, and efficient algorithm that we hope will enable a better understanding of neural network pruning both in theory and practice.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

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

Bubacarr Bah and Jared Tanner. On the construction of sparse matrices from expander graphs. *Frontiers in Applied Mathematics and Statistics*, 4:39, 2018.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint arXiv:1804.05345*, 2018.

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.

Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

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 2020.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Legr: Filter pruning via learned global ranking. 2019.

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

Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5840–5848, 2017.

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.

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, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

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

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.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv e-prints*, art. arXiv:1502.01852, February 2015a.

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

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018a.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

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

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018b.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

Rajiv Khanna and Anastasios Kyrillidis. Iht dies hard: Provable accelerated iterative hard thresholding. In *International Conference on Artificial Intelligence and Statistics*, pages 188–198. PMLR, 2018.

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.

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*, 2019.

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019.

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.

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.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Ben Mussay, Margarita Osadchy, Vladimir Braverman, Samson Zhou, and Dan Feldman. Data-independent neural pruning via coresets. *arXiv preprint arXiv:1907.04018*, 2019.

Deanna Needell and Joel A Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and computational harmonic analysis*, 26(3):301–321, 2009.

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.

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

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

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.

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.

Xavier Suau, Nicholas Apostoloff, et al. Filter distillation for network compression. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 3129–3138. IEEE, 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.

Cameron R Wolfe, Qihan Wang, Junhyung Lyle Kim, and Anastasios Kyrillidis. Provably efficient lottery ticket discovery. *arXiv preprint arXiv:2108.00259*, 2021.

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.

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.

Shuai Zhang, Meng Wang, Sijia Liu, Pin-Yu Chen, and Jinjun Xiong. Why lottery ticket wins? a theoretical perspective of sample complexity on pruned neural networks. *arXiv preprint arXiv:2110.05667*, 2021.

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. Experimental Details

A.1. Synthetic Experiments

Here, we present the details for generating the synthetic data used for pruning experiments with i-SpaSP on two-layer neural networks presented in Section 6.1. It should be noted that the synthetic data generated within this set of experiments does not correspond to the input data matrix $X \in \mathbb{R}^{d_{in} \times B}$ described in Section 2. Rather, the synthetic data that is generated $\{H^{(i)}\}_{i=1}^{K}$ corresponds to the hidden representation of a two-layer neural network, constructed as $H^{(i)} = \sigma(W^{(0)} \cdot X)$ for some input X. Within these experiments, we generate hidden representations instead of raw input because the row-compressibility ratio p considered within Theorem 4 is with respect to the neural network's hidden representations (i.e., not with respect to the input).

Consider the i-th synthetic hidden representation matrix $H^{(i)} \in \mathbb{R}^{d_{hid} \times B}$, and assume this matrix has a desired row-compressibility ratio p. For all experiments, we set B (i.e., the size of the dataset) to be 100. Recall that all entries within $H^{(i)}$ must be non-negative due to the ReLU activation present within (1). For each row $j \in [d_{hid}]$ of the matrix, the synthetic hidden representation is generated by $i)$ computing the upper bound on the sum of the values within this j-th row as $\frac{R}{i^{\frac{1}{p}}}$ and $ii)$ randomly sampling B non-negative values that, when summed together, do not exceed the upper bound.[9] Here, R is a scalar constant as described in Section 2, which we set to $R = 1$. After this process has been completed for each row, the rows of $H^{(i)}$ are randomly shuffled so as to avoid rows being in sorted, magnitude order.

Within the experiments presented in Section 6.1, we generate three random matrices, following the process described above, for each combination of d_{hid} and p. Then, the average pruning results across each of these three matrices is reported for each experimental setting. Experiments were also replicated with different settings of R, but the results observed were quite similar. Further, the $W^{(1)}$ matrix used within the synthetic experiments of Section 6.1 was generated using standard Kaiming initialization (He et al., 2015a), which is supported in deep learning packages like PyTorch (Paszke et al., 2017).

A.2. Two-Layer Networks

Within this section, we provide all relevant experimental details for pruning experiments with two-layer networks presented in Section 6.2.

Baseline Network. We begin by describing the baseline two-layer network that was used within pruning experiments on MNIST, as well as relevant details for pre-training the network prior to pruning. The network used within experiments in Section A.2 exactly matches the formulation in (1), and MNIST images are flattened—forming a 784-dimensional input vector—prior to being passed as input to the network. The network is first pre-trained on the MNIST dataset such that it has fully converged before being used in pruning experiments. During pre-training, we optimize the network with stochastic gradient descent (SGD) using a batch size of 128 and employ a learning rate step schedule that decays the learning rate $10\times$ after completing 50% and 75% of total epochs. For all network and pre-training hyperparameters, we identify optimal settings by dividing the MNIST training set randomly (i.e., a random, 80-20 split) into training and validation sets. Then,

9. In practice, this is implemented by keeping a running sum and continually sampling numbers randomly within a range that does not exceed the upper bound.

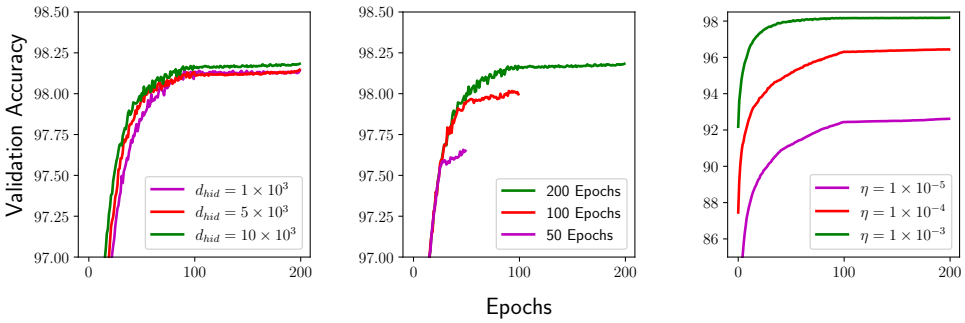

Figure 4: Performance of two-layer networks with different hyperparameter settings on the MNIST validation set. From left to right, subplots depict different hidden dimensions, pre-training epochs, and learning rates. For each of the plots, the best setting with respect to other hyperparameters is displayed.

performance is measured over the validation set using three separate trials and averaged to identify proper hyperparameter settings.

We find that weight decay and momentum settings do not significantly impact network performance. Thus, we use no weight decay during pre-training and set momentum to 0.9. The results of other hyperparameter tuning experiments are provided in Figure 4. We test different hidden dimensions of the two-layer network $d_{hid} \in \{1 \times 10^3,\ 5 \times 10^3,\ 10 \times 10^3\}$, finding that validation performance reaches a plateau when $d_{hid} = 10 \times 10^3$. We also test numerous possible learning rates $\eta \in \{1 \times 10^{-5},\ 1 \times 10^{-4},\ 1 \times 10^{-3}\}$ and pre-training epochs. Within these experiments, it is determined that a learning rate of $\eta = 1 \times 10^{-3}$ with 200 epochs of pre-training acheives the best performance on the validation set; as shown in the right and middle subplots of Figure 4. Thus, the dense networks used for experiments within Section 6.2 has a hidden dimension of 10×10^3 and is pre-trained for 200 epochs using a learning rate of 1×10^{-3} prior to pruning being performed.

Network Pruning. The two-layer network described above is pruned using several greedy selection strategies, including i-SpaSP, GFS, and Top-K. Each of these pruning strategies selects a subset of neurons within the dense network's hidden layer based on a mini-batch of data from the training set. Within all experiments, we adopt a batch size of 512 for pruning.

For i-SpaSP, the pruning procedure follows the exact steps outlined in Algorithm 1. Within each iteration of Algorithm 1, a new mini-batch of data is sampled to compute neuron importance. We use a fixed number of iterations as the stopping criterion for i-SpaSP. In particular, we terminate the algorithm and return the pruned model after 20 iterations, as the active set of neurons is consistently stable at this point.

An in-depth description of GFS is provided in (Ye et al., 2020). For two-layer networks, GFS is implemented by, at each iteration, sampling a new mini-batch of data and finding the neuron that, when included in the (initially empty) active set, yields the largest decrease in loss over the mini-batch. This process is repeated until the desired size of the pruned network has been reached—i.e., each iteration adds a single neuron to the pruned network and the size of the pruned network is used as a stopping criterion. Top-K is a naive, baseline method for greedy selection, which operates by computing hidden representations across the mini-batch and selecting the k neurons with the largest-

magnitude hidden activations (i.e., based on summing hidden representation magnitudes across the mini-batch). Top-K performs the entire pruning process in one step using a single mini-batch.

A.3. Deep Networks

Within this section, we provide relevant details to the large-scale CNN experiments on ImageNet (ILSVRC2012) presented in Section 6.3.

Pruning Deep Networks. As described in Section 4.1, i-SpaSP is extended to deep networks by greedily pruning each layer of the network from beginning to end. We prune filters within each block of the network independently, where a block either consists of two 3×3 convolution operations separated by batch normalization and ReLU (i.e., `BasicBlock` for ResNet34) or a depth-wise, separable convolution with a linear bottleneck (i.e., `InvertedResidualBlock` for MobileNetV2). We maintain the input and output channel resolution of each block—choosing instead to reduce each block's intermediate channel dimension—by performing pruning as follows:

- `BasicBlock`: We obtain the output of the first convolution (i.e., after batch normalization and ReLU) and perform pruning based on the forward pass of the second convolution, thus eliminating filters from the output of the first convolution and, in turn, the input of the second convolution.
- `InvertedResidualBlock`: We obtain the output of the first 1×1 point-wise and 3×3 depth-wise convolutions (i.e., after normalization and ReLU) and prune the filters of this representation based on the forward pass of the last 1×1 point-wise linear convolution, thus removing filters from the output of the first point-wise convolution and, in turn, within the depth-wise convolution operation.

Thus, each convolutional block is separated into two components. Input data is passed through the first component to generate a hidden representation, then i-SpaSP performs pruning using forward and backward passes of the second component as in Algorithm 1. Intuitively, one can view the two components of each block as the two neural network layers in (1). Due to implementation with automatic differentiation, pruning convolutional layers is nearly identical to pruning two-layer network as described in Section 4.1, the only difference being the need to sum over both batch and spatial dimensions in computing importance.

Pruning Ratios. As mentioned within Section 6.3, some blocks within each network are more sensitive to pruning than others. Thus, pruning all layers to a uniform ratio typically does not perform well. Within ResNet34, four groups of convolutional blocks exist, where each group contains a sequence of convolutional blocks with the same channel dimension. Following the recommendations of Li et al. (2016), we avoid pruning the fourth group of convolutional blocks, any strided blocks, or the last block in any group. The 2.69 GFlop model in Table 1 uses a uniform ratio of 40% for all three groups (i.e., 60% of filters are eliminated from each block). For the 2.13 GFlop model, the first group is pruned to a ratio of 20%, while the second and third groups are pruned to a ratio of 30%.

Within MobileNetV2, we simply avoid pruning network blocks that are either strided or have different input and output channel dimensions, leaving the following blocks to be pruned: 3, 5, 6, 8, 9, 10, 12, 13, 15, 16. We find that blocks 3, 5, 8, 15, and 16 are especially sensitive to pruning (i.e., pruning these layers causes noticeable performance degradation), so we avoid aggressive pruning upon these blocks. The 260 MFlop model is generated by pruning only blocks 6, 9, 10, 12, and 13 to

a ratio of 40%. For the 242 MFlop and 220 MFlop models, we prune sensitive (i.e., blocks 3, 4, 8, 15, and 16) and not sensitive (i.e., blocks 6, 9, 10, 12, and 13) blocks with ratios of 40%/80% and 30%/70%, respectively. For the 220 MFlop model, we also prune blocks 2, 11, and 17 to a ratio of 90% to further decrease the FLOP count.

Pruning Hyperparameters. Pruning of each layer is performed with a batch of 1280 data examples for both ResNet34 and MobileNetV2 architectures. This batch of data can be separated into multiple mini-batches (e.g., of size 256) during pruning. Such an approach allows the computation of i-SpaSP to be performed on a GPU (i.e., all of the data examaples cannot simultaneously fit within memory of a typical GPU), but this modification does not change the runtime of behavior of i-SpaSP. Furthermore, 20 total i-SpaSP iterations are performed in pruning each layer. We find that adding more data or pruning iterations beyond these settings does not provide any benefit to i-SpaSP performance.

Fine-tuning Details. For fine-tuning, we perform 90 epochs of fine-tuning using SGD with momentum and employ a cosine learning rate decay schedule over the entire fine-tuning process beginning with a learning rate of 0.01. For fine-tuning between the pruning of layers (i.e., a small amount of fine-tuning is performed after each layer is pruned), we simply adopt the same fine-tuning setup with a fixed learning rate of 0.01 and fine-tuning continues for one epoch. Our learning rate setup for fine-tuning matches that of Ye et al. (2020); Ye et al. (2020), but we choose to perform fewer epochs of fine-tuning, as we find network performance does not improve much after 90 epochs. In all cases, we adopt standard augmentation procedures for the ImageNet dataset during fine-tuning (i.e., normalizing the data and performing random crops and flips).

Appendix B. Main Proofs

B.1. Properties of $\mu(\cdot)$

Prior to providing any proofs of our theoretical results, we show a few properties of the function $\mu(\cdot)$ defined over matrices that will become useful in deriving later results. Given $A \in \mathbb{R}^{m\times n}$, $\mu(A) : \mathbb{R}^{m\times n} \longrightarrow \mathbb{R}^m$ and is defined as follows:

$$\mu(A) = \sum_{i=1}^{n} A_{:,i}$$

In words, $\mu(A)$ sums all columns within an arbitraty matrix A to produce a single column vector. We now provide a few useful properties of $\mu(\cdot)$.

Lemma 6 *Consider two arbitrary matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$. The following properties of $\mu(\cdot)$ must be true.*

$$\mu(A \cdot B) = A \cdot \mu(B)$$
$$\mu(A + B) = \mu(A) + \mu(B)$$

Proof Given the matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$, we can arrive at the first property as follows:

$$\mu(A \cdot B) = \sum_{i=1}^{p} (A \cdot B)_{:,i} = \sum_{i=1}^{p} A \cdot B_{:,i} = A \cdot \left(\sum_{i=1}^{p} B_{:,i} \right) = A \cdot \mu(B)$$

Furthermore, the second property can be shown as follows:

$$\mu(A+B) = \sum_{i=1}^{p}(A+B)_{:,i} = \sum_{i=1}^{p} A_{:,i} + B_{:,i} = \sum_{i=1}^{p} A_{:,i} + \sum_{j=1}^{p} B_{:,j} = \mu(A) + \mu(B)$$

■

B.2. Proof of Lemma 1

Here, we provide a proof for Lemma 1 from Section 5.

Proof Consider an arbitrary iteration of Algorithm 1. We define the active neurons within the pruned network entering this iteration as $\mathcal{S}$. The notation $\mathcal{S}'$ will be used to refer to the active neurons in the pruned model after the completion of the iteration. Similarly, we will use the $\cdot'$ notation to refer to other constructions after the iteration completes (e.g., V becomes V', R becomes R', etc.)

We begin by defining and expanding the definitions of the matrices U, V, R, and Y considering the substitution of $H = Z + E$;

$$\begin{aligned}
&U = W^{(1)}H = W^{(1)}(Z+E),\\
&V = U - W^{(1)}H_{\mathcal{S},:} = W^{(1)}(Z+E) - W^{(1)}(Z+E)_{\mathcal{S},:},\\
&R = Z - Z_{\mathcal{S},:},\\
&Y = (W^{(1)})^T V.
\end{aligned}$$

Now, observe the following about Y.

$$\begin{aligned}
Y &= (W^{(1)})^\top \cdot V\\
&= (W^{(1)})^\top \left(W^{(1)}(Z+E) - W^{(1)}(Z+E)_{\mathcal{S},:}\right)\\
&= (W^{(1)})^\top \left(W^{(1)}Z + W^{(1)}E - W^{(1)}(Z_{\mathcal{S},:}) - W^{(1)}(E_{\mathcal{S},:})\right)\\
&= (W^{(1)})^\top \left(W^{(1)}(Z - Z_{\mathcal{S},:}) + W^{(1)}(E - E_{\mathcal{S},:})\right)\\
&= (W^{(1)})^\top \left(W^{(1)}(Z - Z_{\mathcal{S},:}) + W^{(1)}\hat{E}\right)\\
&= (W^{(1)})^\top W^{(1)}(Z - Z_{\mathcal{S},:}) + (W^{(1)})^\top W^{(1)}\hat{E},
\end{aligned}$$

where we denote $\hat{E} = E - E_{\mathcal{S},:}$. It should be noted that $\hat{E}$ is simply the matrix E with all the i-th rows set to zero where $i \in \mathcal{S}$. Invoking Lemma 6 in combination with the above expression, we can show the following.

$$\begin{aligned}
\mu(Y) &= (W^{(1)})^\top W^{(1)}\mu(Z - Z_{\mathcal{S},:}) + (W^{(1)})^\top W^{(1)}\mu(\hat{E})\\
&= (W^{(1)})^\top W^{(1)}\left(\mu(R) + \mu(\hat{E})\right).
\end{aligned}$$

Based on the definition of $H = Z + E$, R must be s-row-sparse. This is because Z is s-row-sparse and $R = Z - Z_{\mathcal{S},:}$. Thus, the number of non-zero rows within R must be less than or equal to

s (i.e., the rows that are not subtracted out of Z by $Z_{\mathcal{S},:}$). Denote $\Delta = \text{rsupp}(R)$, where $|\Delta| \leq s$. From the definition of Ω in Algorithm 1, we have the following properties of Y:

$$\|\mu(Y)|_{\Delta}\|_2 \leq \|\mu(Y)|_{\Omega}\|_2 \quad \overset{i)}{\Longrightarrow} \quad \|\mu(Y)|_{\Delta\setminus\Omega}\|_2 \leq \|\mu(Y)|_{\Omega\setminus\Delta}\|_2$$

where $i)$ follows by removing indices within $\Omega \cap \Delta$. From here, we can upper bound the value of $\|\mu(Y)|_{\Omega\setminus\Delta}\|_2$ as follows.

$$\begin{aligned}
\|\mu(Y)|_{\Omega\setminus\Delta}\|_2 &= \left\| \left((W^{(1)})^\top W^{(1)} \mu(R) \right) |_{\Omega\setminus\Delta} + \left((W^{(1)})^\top W^{(1)} \mu(\hat{E}) \right) |_{\Omega\setminus\Delta} \right\|_2 \\
&= \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(R) + (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{i)}{\leq} \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(R) \right\|_2 + \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{ii)}{\leq} \delta_{3s} \|\mu(R)\|_2 + \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{iii)}{\leq} \delta_{3s} \|\mu(R)\|_2 + \sqrt{1+\delta_{2s}} \|W^{(1)} \mu(\hat{E})\|_2 \\
&\overset{iv)}{\leq} \delta_{3s} \|\mu(R)\|_2 + (1+\delta_{2s}) \|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}} \|\mu(\hat{E})\|_1
\end{aligned}$$

where $i)$ follows from the triangle inequality, $ii)$ holds from Corrolary 3.3 on RIP properties of $W^{(1)}$ in (Needell and Tropp, 2009) because $|\Omega \cup \Delta| \leq 3s$, $iii)$ holds from Proposition 3.1 on RIP properties of $W^{(1)}$ in (Needell and Tropp, 2009) because $|\Omega\setminus\Delta| \leq 2s$, and $iv)$ holds from Proposition 3.5 on RIP properties of $W^{(1)}$ in (Needell and Tropp, 2009). Now, we can also lower bound the value of $\|\mu(Y)|_{\Delta\setminus\Omega}\|_2$ as follows:

$$\begin{aligned}
\|\mu(Y)|_{\Delta\setminus\Omega}\|_2 &= \left\| \left((W^{(1)})^\top W^{(1)} \mu(R) \right) |_{\Delta\setminus\Omega} + \left((W^{(1)})^\top W^{(1)} \mu(\hat{E}) \right) |_{\Delta\setminus\Omega} \right\|_2 \\
&= \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R) + (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&= \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Delta\setminus\Omega} + (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Omega} + (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{i)}{\geq} \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Delta\setminus\Omega} \right\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Omega} \right\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{ii)}{\geq} (1-\delta_s) \|\mu(R)|_{\Delta\setminus\Omega}\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Omega} \right\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{iii}{\geq} (1-\delta_s) \|\mu(R)|_{\Delta\setminus\Omega}\|_2 - \delta_s \|\mu(R)|_{\Omega}\|_2 - \sqrt{1+\delta_s} \|W^{(1)} \mu(\hat{E})\|_2 \\
&\overset{iv)}{\geq} (1-\delta_s) \|\mu(R)|_{\Delta\setminus\Omega}\|_2 - \delta_s \|\mu(R)\|_2 - (1+\delta_{2s}) \|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}} \|\mu(\hat{E})\|_1
\end{aligned}$$

where $i)$ follows from the triangle inequality. $ii)$ follows from Proposition 3.1 in (Needell and Tropp, 2009) and $iii)$ follows from Corollary 3.3 and Proposition 3.1 in (Needell and Tropp, 2009) because $|\Delta\setminus\Omega| \leq s$. Finally, $iv)$ follows from Proposition 3.5 within (Needell and Tropp, 2009). By noting that $\mu(R)|_{\Delta\setminus\Omega} = \mu(R)|_{\Omega^c}$ and combining the lower and upper bounds above, we derive the

following:

$$\|\mu(R)|_{\Omega^c}\|_2 \leq \frac{(\delta_s + \delta_{3s})\|\mu(R)\|_2 + 2\left((1+\delta_{2s})\|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}}\|\mu(\hat{E})\|_1\right)}{1-\delta_s} \tag{4}$$

From here, we can show the following:

$$\begin{aligned}
\|\mu(R)|_{\Omega^c}\|_2 &\overset{i)}{\geq} \|\mu(R)|_{\Omega^{\star c}}\|_2 \\
&= \|\mu(Z - Z_{\mathcal{S},:})|_{\Omega^{\star c}}\|_2 \\
&= \|(\mu(Z) - \mu(Z)|_{\mathcal{S}})|_{\Omega^{\star c}}\|_2 \\
&\overset{ii)}{=} \|\mu(Z)|_{\Omega^{\star c}}\|_2 \\
&= \|\mu(H - E)|_{\Omega^{\star c}}\|_2 \\
&= \|\mu(H)|_{\Omega^{\star c}} - \mu(E)|_{\Omega^{\star c}}\|_2 \\
&\overset{iii)}{\geq} \|\mu(H)|_{\Omega^{\star c}}\|_2 - \|\mu(E)|_{\Omega^{\star c}}\|_2
\end{aligned} \tag{5}$$

where $i)$ follows from the fact that $\Omega \subseteq \Omega^\star$, $ii)$ follows from the fact that $\mathcal{S} \subset \Omega^\star$, and $iii)$ follows from the triangle inequality. Then, as a final step, we make the following observation, where we leverage notation from Algorithm 1:

$$\begin{aligned}
\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 &= \|\mu(H) - b_s\|_2 \\
&= \|(\mu(H) - b) + (b - b_s)\|_2 \\
&\overset{i)}{\leq} \|\mu(H) - b\|_2 + \|b - b_s\|_2 \\
&\overset{ii)}{\leq} 2\|\mu(H) - b\|_2 \\
&= 2\|\mu(H) - \mu(H)|_{\Omega^\star}\|_2
\end{aligned} \tag{6}$$

where $i)$ follow from the triangle inequality and $ii)$ follows from the fact that b_s is the best s-sparse approximation to b (i.e., $\mu(H)$ is a worse s-sparse approximation with respect to the ℓ_2 norm). Now, noticing that $\|\mu(H) - \mu(H)|_{\Omega^\star}\|_2 = \|\mu(H)|_{\Omega^{\star c}}\|_2$, we can aggregate all of this information as follows:

$$\begin{aligned}
\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 - 2\|\mu(\hat{E})\|_2 &\overset{i)}{\leq} \|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 - 2\|\mu(E)|_{\Omega^{\star c}}\|_2 \\
&\overset{ii)}{\leq} 2\left(\|\mu(H) - \mu(H)|_{\Omega^\star}\|_2 - \|\mu(E)|_{\Omega^{\star c}}\|_2\right) \\
&= 2\left(\|\mu(H)|_{\Omega^{\star c}}\|_2 - \|\mu(E)|_{\Omega^{\star c}}\|_2\right) \\
&\overset{iii)}{\leq} 2\|\mu(R)|_{\Omega^c}\|_2 \\
&\overset{iv)}{\leq} \frac{2(\delta_s + \delta_{3s})\|\mu(R)\|_2 + 4\left((1+\delta_{2s})\|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}}\|\mu(\hat{E})\|_1\right)}{1-\delta_s}
\end{aligned}$$

where $i)$ holds because $\mathcal{S} \subset \Omega^\star$, $ii)$ holds from (6), $iii)$ holds from (5), and $iv)$ hold from (4). By simply moving the $\|\mu(\hat{E})\|_2$ term to the other side of the inequality, we get the following:

$$\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 \leq \frac{2(\delta_s + \delta_{3s})\|\mu(R)\|_2 + (6 + 4\delta_{2s} - 2\delta_s)\|\mu(\hat{E})\|_2 + \frac{4(1+\delta_{2s})}{\sqrt{2s}}\|\mu(\hat{E})\|_1}{1-\delta_s}$$

From here, we recover $\mu(H) - \mu(H)|_{\mathcal{S}}$ from $\mu(R)$ as follows:

$$\begin{aligned}\|\mu(R)\|_2 &= \|\mu(Z - Z|_{\mathcal{S}})\|_2 \\ &= \|\mu(H - E - (H - E)|_{\mathcal{S}})\|_2 \\ &= \|\mu(H - H|_{\mathcal{S}}) - \mu(E - E|_{\mathcal{S}})\|_2 \\ &\overset{i)}{\leq} \|\mu(H - H|_{\mathcal{S}})\|_2 + \|\mu(E - E|_{\mathcal{S}})\|_2 \\ &\leq \|\mu(H) - \mu(H)|_{\mathcal{S}}\|_2 + \|\mu(\hat{E})\|_2\end{aligned}$$

where $i)$ holds from the triangle inequality. Combining this with the final inequality derived above, we arrive at the following recursion for error between pruned and dense model hidden layers over iterations of Algorithm 1:

$$\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 \leq \frac{2(\delta_s + \delta_{3s})\|\mu(H) - \mu(H)|_{\mathcal{S}}\|_2 + (6 + 2\delta_{3s} + 4\delta_{2s})\|\mu(\hat{E})\|_2 + \frac{4(1+\delta_{2s})}{\sqrt{2s}}\|\mu(\hat{E})\|_1}{1 - \delta_s}$$

We now adopt an identical numerical assumption as (Needell and Tropp, 2009) and assume that $W^{(1)}$ has restricted isometry constant $\delta_{4s} \leq 0.1$. Then, noting that $\delta_s, \delta_{2s}, \delta_{3s} \leq \delta_{4s} \leq 0.1$ per Corollary 3.4 in (Needell and Tropp, 2009), we substitute the restricted isometry constants into the above recursion to yield the following expression:

$$\begin{aligned}\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 &\leq 0.444\|\mu(H) - \mu(H)|_{\mathcal{S}}\|_2 + 7.333\|\mu(\hat{E})\|_2 + \frac{4.888}{\sqrt{2s}}\|\mu(\hat{E})\|_1 \\ &= 0.444\|\mu(H) - \mu(H)|_{\mathcal{S}}\|_2 + 7.333\|\mu(\hat{E})\|_2 + \frac{3.456}{\sqrt{s}}\|\mu(\hat{E})\|_1\end{aligned}$$

If we invoke Lemma 6, unroll the above recursion over t iterations of Algorithm 1, notice it is always true that $\|\hat{E}\|_2 \leq \|E\|_2$ and $\|\hat{E}\|_1 \leq \|E\|_1$, and draw upon the property of ℓ_1 and ℓ_2 vector norms that $\|\cdot\|_2 \leq \|\cdot\|_1$ we arrive at the desired result:

$$\begin{aligned}\|\mu(H - H_{\mathcal{S}_t,:})\|_2 &= \|\mu(H) - \mu(H)|_{\mathcal{S}_t}\|_2 \\ &\leq (0.444)^t\|\mu(H) - \mu(H)|_{\mathcal{S}_0}\|_2 + 14\|\mu(E)\|_2 + \frac{7}{\sqrt{s}}\|\mu(E)\|_1 \\ &\leq (0.444)^t\|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right)\|\mu(E)\|_1\end{aligned}$$

■

B.3. Proof of Theorem 2

Here, we provide a proof of Theorem 2 from Section 5.

Proof Consider an arbitrary iteration t of Algorithm 1 with a set of active neurons within the pruned model denoted by $\mathcal{S}_t$. The matrix V_t contains the residual between the dense and pruned model output over the entire dataset, as formalized below:

$$\begin{aligned}V_t &= U - (W^{(1)} \cdot H_{\mathcal{S}_t,:}) \\ &= (W^{(1)} \cdot H) - (W^{(1)} \cdot H_{\mathcal{S}_t,:}) \\ &= W^{(1)} \cdot (H - H_{\mathcal{S}_t,:})\end{aligned}$$

Consider the squared Frobenius norm of the matrix V_t. We can show the following:

$$\begin{aligned}
\|V_t\|_F^2 &= \|W^{(1)} \cdot (H - H_{\mathcal{S}_t,:})\|_F^2 \\
&= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(\sum_{z=1}^{d_{hid}} W_{iz}^{(1)} (H - H_{\mathcal{S}_t,:})_{zj} \right)^2 \\
&= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(W_{i,:}^{(1)} \cdot (H - H_{\mathcal{S}_t,:})_{:,j} \right)^2 \\
&= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(|W_{i,:}^{(1)} \cdot (H - H_{\mathcal{S}_t,:})_{:,j}| \right)^2 \\
&\overset{i)}{\leq} \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(\|W_{i,:}^{(1)}\|_2 \cdot \|(H - H_{\mathcal{S}_t,:})_{:,j}\|_2 \right)^2 \\
&= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \|W_{i,:}^{(1)}\|_2^2 \cdot \|(H - H_{\mathcal{S}_t,:})_{:,j}\|_2^2 \\
&= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(\sum_{k=1}^{d_{hid}} (W_{ik}^{(1)})^2 \right) \left(\sum_{z=1}^{d_{hid}} (H - H_{\mathcal{S}_t,:})_{zj}^2 \right) \\
&= \left(\sum_{i=1}^{d_{out}} \sum_{k=1}^{d_{hid}} (W_{ik}^{(1)})^2 \right) \left(\sum_{j=1}^{B} \sum_{z=1}^{d_{hid}} (H - H_{\mathcal{S}_t,:})_{zj}^2 \right) \\
&= \|W^{(1)}\|_F^2 \left(\sum_{z=1}^{d_{hid}} \sum_{j=1}^{B} (H - H_{\mathcal{S}_t}, :)_{zj}^2 \right) \\
&\overset{ii)}{\leq} \|W^{(1)}\|_F^2 \left(\sum_{z=1}^{d_{hid}} \mu(H - H_{\mathcal{S}_t,:})_z^2 \right) \\
&= \|W^{(1)}\|_F^2 \cdot \|\mu(H - H_{\mathcal{S}_t,:})\|_2^2
\end{aligned}$$

where $i)$ follows from the Cauchy-Schwarz inequality and $ii)$ follows from the fact that all entries of $H - H_{\mathcal{S}_t,:}$ are non-negative. From here, we simply invoke Lemma 1 to arrive at the desired result.

$$\begin{aligned}
\|V_t\|_F &\leq \|W^{(1)}\|_F \cdot \|\mu(H - H_{\mathcal{S}_t,:})\|_2 \\
&\leq \|W^{(1)}\|_F \cdot \left((0.444)^t \|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}} \right) \|\mu(E)\|_1 \right)
\end{aligned}$$

■

B.4. Proof of Lemma 3

Here, we provide the proof for Lemma 3 from Section 5.

Proof Assume that we choose the s-row-sparse Z matrix within $H = Z + E$ as follows

$$Z = H_{\mathcal{D},:}, \text{ where } \mathcal{D} = \underset{\mathcal{D} \subset [d_{hid}]}{\arg\min} \|\mu(H) - \mu(H)|_{\mathcal{D}}\|$$

The norm used within the above expression is arbitrary, as we simply care to select indices $\mathcal{D}$ such that $\mu(H)|_{\mathcal{D}}$ contains the largest-magnitude components of $\mu(H)$. Such a property will be true for any ℓ_p norm selection within the above equation, as they all ensure $\mu(H)|_{\mathcal{D}}$ is the best possible s-sparse approximation of $\mu(H)$. From here, we have that $E = H - H_{\mathcal{D}}$, which yields the following:

$$\begin{aligned} \|\mu(E)\|_1 &= \|\mu(H - H_{\mathcal{D}})\|_1 \\ &\overset{i)}{=} \|\mu(H) - \mu(H)|_{\mathcal{D}}\|_1 \\ &\overset{ii)}{\leq} R \cdot \frac{s^{1-\frac{1}{p}}}{\frac{1}{p} - 1} \end{aligned}$$

where $i)$ holds from Lemma 6 and $ii)$ is a bound on the ℓ_1 norm of the best-possible s-sparse approximation of a p-compressible vector; see Section 2.6 in (Needell and Tropp, 2009). ■

B.5. Proof of Theorem 5

Here, we generalize our theoretical results beyond single-hidden-layer networks to arbitrary depth networks. For this section, we define the forward pass in a L-hidden-layer network via the following recursion:

$$H^{(\ell)} = \sigma(W^{(\ell-1)} \cdot H^{(\ell-1)})$$

where σ denotes the nonlinear ReLU activation applied at each layer and we have weight matrices $\mathcal{W} = \{W^{(0)}, W^{(1)}, \ldots, W^{(L)}\}$ and hidden representations $\mathcal{H} = \{H^{(0)}, H^{(1)}, \ldots, H^{(L)}\}$ for each of the L layers within the network. Here, $H^{(L)}$ denotes the network output and $H^{(0)} = (W^{(0)} \cdot X)$. It should be noted that no ReLU activation is applied at the final network output (i.e., $H^{(L)} = W^{(L)} \cdot H^{(L-1)}$). We denote the pre-activation values each hidden representation as $U^{(\ell)} = W^{(\ell)} \cdot H^{(\ell-1)}$.

Proof We denote the active set of neurons after t iterations at each layer ℓ as $\mathcal{S}_t^{(\ell)}$. The dense network output is given by $U^{(L)} = H^{(L)}$, and we compute the residual between pruned and dense networks at layer ℓ as $V_t^{(\ell)} = U^{(\ell)} - W^{(\ell)}_{:,\mathcal{S}_t^{(\ell)}} \cdot H^{(\ell-1)}_{\mathcal{S}_t^{(\ell)},:}$. We further denote $U_t^{(\ell)\prime} = W^{(\ell)}_{:,\mathcal{S}_t^{(\ell)}} \cdot H^{(\ell-1)}_{\mathcal{S}_t^{(\ell)},:}$ as shorthand for intermediate, pre-activation outputs of the pruned network. For simplicity, we assume all hidden layers of the multi-layer network have the same number of hidden neurons d and are pruned to a size s, such that $s \ll d$. Consider the final output representation of an L-hidden-layer network $H^{(L)}$. We will now bound $\|V_t^{(L)}\|_F$, revealing that a recursion can be derived over the layers of the network to upper bound the norm of the final layer's residual between pruned and dense networks.

Following previous proofs, we decompose the hidden representation at each layer prior to the output layer as $H^{(\ell)} = Z^{(\ell)} + E^{(\ell)} + \Delta^{(\ell)}$, where $Z^{(\ell)}$ is an s-row-sparse matrix and $E^{(\ell)}$ and $\Delta^{(\ell)}$ are both arbitrary-valued matrices. $E^{(\ell)}$ denotes the same arbitrarily-valued matrix from the previous $H = Z + E$ formulation and $\Delta^{(\ell)}$ captures all error introduced by pruning layers prior to ℓ within

the network. $E^{(\ell)}$ and $\Delta^{(L)}$ can also be combined into a single matrix as $\Xi^{(\ell)} = E^{(\ell)} + \Delta^{(\ell)}$. Now, we consider the value of $\|V_t^{(L)}\|_F$:

$$\begin{aligned}
\|V_t^{(L)}\|_F &\overset{i}{=} \|W^{(L)}\|_F \cdot \left((0.444)^t \|\mu(H^{(L-1)})\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(\Xi^{(L-1)})\|_1 \right) \\
&= \|W^{(L)}\|_F \cdot \left((0.444)^t \|\mu(H^{(L-1)})\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(E^{(L-1)}) + \mu(\Delta^{(L-1)})\|_1 \right) \\
&\overset{ii}{\leq} \|W^{(L)}\|_F \cdot \left((0.444)^t \|\mu(H^{(L-1)})\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \left(\|\mu(E^{(L-1)})\|_1 + \|\mu(\Delta^{(L-1)})\|_1 \right) \right) \\
&\overset{iii}{\approx} \|W^{(L)}\|_F \cdot \left(14 + \frac{7}{\sqrt{s}}\right) \left(\|\mu(E^{(L-1)})\|_1 + \|\mu(\Delta^{(L-1)})\|_1 \right) \qquad (7)
\end{aligned}$$

where i holds from Theorem 2, ii holds from the triangle inequality, and iii holds by eliminating all terms that approach zero with enough pruning iterations t. Now, notice that $\|\mu(\Delta^{(L-1)})\|_1$ characterizes the error induced by pruning within the previous layer of the network, following the ReLU activation. Thus, our expression for the error induced by pruning on network output is now expressed with respect to the pruning error of the previous layer, revealing that a recursion can be derived for pruning error over the entire multi-layer network. We now expand the expression for $\|\mu(\Delta^{(\ell)})\|_1$ (i.e., we use arbitrary layer ℓ to enable a recursion over layers to be derived).

$$
\begin{aligned}
\|\mu(\Delta^{(\ell)})\|_1 &\overset{i)}{=} \|\mu(\sigma(U^{(\ell)}) - \sigma(U_t^{(\ell)\prime}))\|_1 \\
&\overset{ii)}{\leq} \|\mu(\sigma(U^{(\ell)} - U_t^{(\ell)\prime}))\|_1 \\
&= \|\mu(\sigma(V_t^{(\ell)}))\|_1 \\
&= \left\|\mu\left(\sigma\left(W^{(\ell)} \cdot \left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)\right)\right)\right\|_1 \\
&\overset{iii)}{\leq} \sum_{i=1}^{d}\sum_{j=1}^{B}\left|\left(\sum_{z=1}^{d} W_{iz}^{(\ell)}\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{zj}\right)\right| \\
&= \sum_{i=1}^{d}\sum_{j=1}^{B}\left|W_{i,:}^{(\ell)} \cdot \left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{:,j}\right| \\
&\overset{iv)}{\leq} \sum_{i=1}^{d}\sum_{j=1}^{B}\|W_{i,:}^{(\ell)}\|_1\left\|\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{:,j}\right\|_1 \\
&= \sum_{i=1}^{d}\sum_{j=1}^{B}\left(\sum_{k=1}^{d}\left|W_{ik}^{(\ell)}\right|\right)\left(\sum_{z=1}^{d}\left|\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{zj}\right|\right) \\
&= \left(\sum_{i=1}^{d}\sum_{k=1}^{d}\left|W_{ik}^{(\ell)}\right|\right)\left(\sum_{z=1}^{d}\sum_{j=1}^{B}\left|\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{zj}\right|\right) \\
&= \left\|\mathtt{vec}(W^{(\ell)})\right\|_1 \cdot \left\|\mu\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)\right\|_1 \\
&\overset{v)}{\leq} \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1 \cdot \left\|\mu\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)\right\|_2 \\
&\overset{vi)}{\leq} \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1 \left((0.444)^t\|\mu(H^{(\ell-1)})\|_2 + (14 + \frac{7}{\sqrt{s}})\|\mu(\Xi^{(\ell-1)})\|_1\right) \\
&\overset{vii)}{\leq} \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1 \left((0.444)^t\|\mu(H^{(\ell-1)})\|_2 + (14 + \frac{7}{\sqrt{s}})\left(\|\mu(E^{(\ell-1)}\|_1 + \|\mu(\Delta^{(\ell-1)}\|_1\right)\right) \\
&\approx \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1 (14 + \frac{7}{\sqrt{s}})\left(\|\mu(E^{(\ell-1)}\|_1 + \|\mu(\Delta^{(\ell-1)}\|_1\right)
\end{aligned}
$$

where $i)$ holds because $\Delta^{(\ell)}$ simply characterizes the difference between pruned and dense network hidden representations, $ii)$ holds due to properties of the ReLU function, $iii)$ hold by expanding the expression as a sum and replacing the ReLU operation with absolute value, $iv)$ holds due to the Cauchy Schwarz inequality, $v)$ holds due to properties of ℓ_1 norms, $vi)$ holds due to Lemma 1, and $vii)$ holds due to the triangle inequality. As can be seen within the above expression, the value of $\|\mu(\Delta^{(\ell)})\|_1$ can be expressed with respect to $\|\mu(\Delta^{(\ell-1)})\|_1$. Now, by beginning with (7)

and unrolling the equation above over all L layers of the network, we obtain the following

$$\|V_t^{(L)}\|_F \leq \mathcal{O}\left(\sum_{i=1}^{L} d^{\frac{L-i}{2}} \left(14 + \frac{7}{\sqrt{s}}\right)^{L-i+1} \left\|\mu(E^{(i)})\right\|_1 \left(\|W^{(L)}\|_F \prod_{j=1}^{L-i} \|\texttt{vec}(W^{(j)})\|_1\right)\right)$$

Now, assume that $H^{(\ell)}$ is $p^{(\ell)}$-row-compressible with factor $R^{(\ell)}$ for $\ell \in [L-1]$. Then, defining $p = \max_i p^{(i)}$ and $R = \max_i R^{(i)}$, we invoke Lemma 3 to arrive at the desired result

$$\|V_t^{(L)}\|_F \leq \mathcal{O}\left(\sum_{i=1}^{L} \left(14 + \frac{7}{\sqrt{s}}\right)^{L-i+1} \left(\|W^{(L)}\|_F \prod_{j=1}^{L-i} \|\texttt{vec}(W^{(j)})\|_1\right) \left(\frac{d^{\frac{L-i}{2}} s^{1-\frac{1}{p}}}{\frac{1}{p}-1}\right)\right)$$

where R is factored out because it has no asymptotic impact on the expression.

■