

---

# SiPPing Neural Networks: Sensitivity-informed Provable Pruning of Neural Networks

---

Cenk Baykal\*  
CSAIL, MIT  
baykal@mit.edu

Lucas Liebenwein\*  
CSAIL, MIT  
lucasl@mit.edu

Igor Gilitschenski  
CSAIL, MIT  
igilitschenski@mit.edu

Dan Feldman  
University of Haifa  
dannyf.post@gmail.com

Daniela Rus  
CSAIL, MIT  
rus@csail.mit.edu

## Abstract

We introduce a pruning algorithm that provably sparsifies the parameters of a trained model in a way that approximately preserves the model’s predictive accuracy. Our algorithm uses a small batch of input points to construct a data-informed importance sampling distribution over the network’s parameters, and adaptively mixes a sampling-based and deterministic pruning procedure to discard redundant weights. Our pruning method is simultaneously computationally efficient, provably accurate, and broadly applicable to various network architectures and data distributions. Our empirical comparisons show that our algorithm reliably generates highly compressed networks that incur minimal loss in performance relative to that of the original network. We present experimental results that demonstrate our algorithm’s potential to unearth essential network connections that can be trained successfully in isolation, which may be of independent interest.

## 1 Introduction

The deployment of large state-of-the-art neural networks to resource-constrained platforms, such as mobile phones and embedded devices, is often prohibitive in terms of both time and space. *Network pruning* algorithms have the potential to reduce the memory

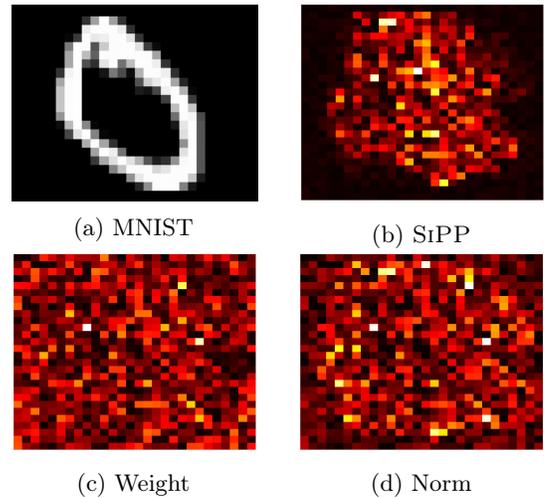


Figure 1: (a): A sample input from the MNIST data-set and (b)-(d): importances assigned to the weights in the first layer by various pruning methods, *SiPP*, *Weight* (Han et al., 2015), and *Norm* (Kundu and Drineas, 2014), for a fully-connected network. The goal is to assign high importances (yellow or red) to the weights corresponding to parts of the image where digits tend to appear. Our data-informed method, SiPP, leverages the structure of the data distribution and assigns lower importances (black) to the weights associated with the corners of the image, where digits do not appear.

footprint and inference time complexity of large neural network models in low-resource settings. The goal of network pruning is to discard redundant weights of an overparameterized network and generate a compressed model whose performance is competitive with that of the original network. In addition, network pruning can be used to reduce the burden of manually designing a small network by automatically inferring efficient architectures from larger networks.

---

\*These authors contributed equally to this work

Existing network pruning algorithms are predominantly based on data-oblivious heuristics (Han et al., 2015) or low-rank decompositions and sparsifications (Denton et al., 2014, Kundu and Drineas, 2014) that at most implicitly take the data distribution into account. Thus, these approaches do not fully capture the importance of each network parameter (see Fig. 1). Even when considering the data-distribution, existing approaches do not provide any provable guarantees (Lee et al., 2018) or are only applicable to a narrow class of architectures (Baykal et al., 2018).

We close this research gap by introducing SiPP, a network pruning algorithm that provably compresses the network’s parameters in a data-informed manner. Building and improving on state-of-the-art pruning methods, our algorithm is simultaneously provably accurate, data-informed, and broadly applicable to various architectures including Fully-connected (FNNs) and Convolutional Neural Networks (CNNs).

This paper contributes the following:

1. A robust and versatile pruning algorithm, SiPP, that combines novel sample size allocation and adaptive sparsification procedures to prune network parameters
2. An analysis of the algorithms, including the size and accuracy of the compressed network generated by our pruning scheme
3. Empirical evaluations for prune-only and iterative prune + retrain scenarios on fully-connected, convolutional, and residual networks, and comparisons to baseline pruning approaches

## 2 Related Work

**Traditional Approaches** Among others, contemporary interest in network pruning was sparked by Denil et al. (2013), which demonstrates the significant extent of overparameterization in modern neural networks. To alleviate computational costs and memory footprint of large models, traditional techniques such as Singular Value Decomposition (SVD) and regularized training of neural networks (Alvarez and Salzmann, 2017, Denton et al., 2014, Ioannou et al., 2015, Jaderberg et al., 2014, Kim et al., 2015, Tai et al., 2015, Yu et al., 2017) were applied to approximate the weight tensors  $W$  with their low-rank counterparts  $\hat{W}$  and to induce sparsity of the weights during training.

Other approaches in this realm exploit the structure of weight tensors to induce sparsity (Cheng et al., 2015, Choromanska et al., 2016, Sindhwani et al., 2015, Wen et al., 2016, Zhao et al., 2017). Another line of work is that of sampling-based matrix and tensor sparsification algorithms (Drineas and Zouzias, 2011, Fung

et al., 2011, Kundu and Drineas, 2014). These methods generate an importance sampling distribution over the entries of tensor  $W$  and obtain a sparse tensor  $\hat{W}$  by sampling and reweighing the entries of  $W$ . These approaches either impose constraints on the structure of the weights, lack theoretical guarantees, or only provide norm-based error bounds on the tensor approximation, i.e., on  $\|W - \hat{W}\|$ . Our work, in contrast, is a data-informed approach with guarantees on the size, relative error incurred at each output, and accuracy of the compressed network.

**Network Pruning** Weight pruning (LeCun et al., 1990) hinges on the idea that only a few dominant weights within a layer are required to approximately preserve the output. Approaches of this flavor were investigated by Dong et al. (2017), Lebedev and Lepitsky (2016), e.g., by embedding sparsity as a constraint (Aghasi et al., 2017, Iandola et al., 2016, Lin et al., 2017). A popular weight-based pruning method is that of Han et al. (2015), where weights with absolute values below a threshold are removed. A recent approach of Lee et al. (2018) prunes the parameters of the network by using a mini-batch of data points to approximate the influence of each parameter on the loss function of a randomly initialized network. Unlike our approach, modern pruning algorithms lack rigorous theoretical analysis of the effect that the discarded weights can have on the model’s performance.

**Theoretical Foundations** Recently, Arora et al. (2018) introduced a compression method based on the Johnson-Lindenstrauss (JL) Lemma and proved norm-based bounds on the performance of the compressed network for points in the *training set only* under the assumption of  $p$ -wise filter independence. In contrast, our work provides more general and stronger entry-wise guarantees on the network’s performance that hold even for points outside the training set. A coresets-based (Braverman et al., 2016, Feldman and Langberg, 2011) approach for provably compressing fully-connected networks was introduced by Baykal et al. (2018) but does not extend to other type of networks. Our approach, on the other hand, is widely applicable to various network architectures, including CNNs, exhibits stronger theoretical properties by mixing deterministic and sampling-based pruning strategies, and leverages our error bounds to optimally allocate sample sizes across the network’s filters and layers.

## 3 Problem Definition

The set of parameters  $\theta$  of a CNN with  $L$  convolutional layers is a tuple of 4-dimensional weight matrices corresponding to each layer, i.e.,  $\theta = (W^1, \dots, W^L)$ . The set of parameters  $\theta$  defines the mapping  $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$

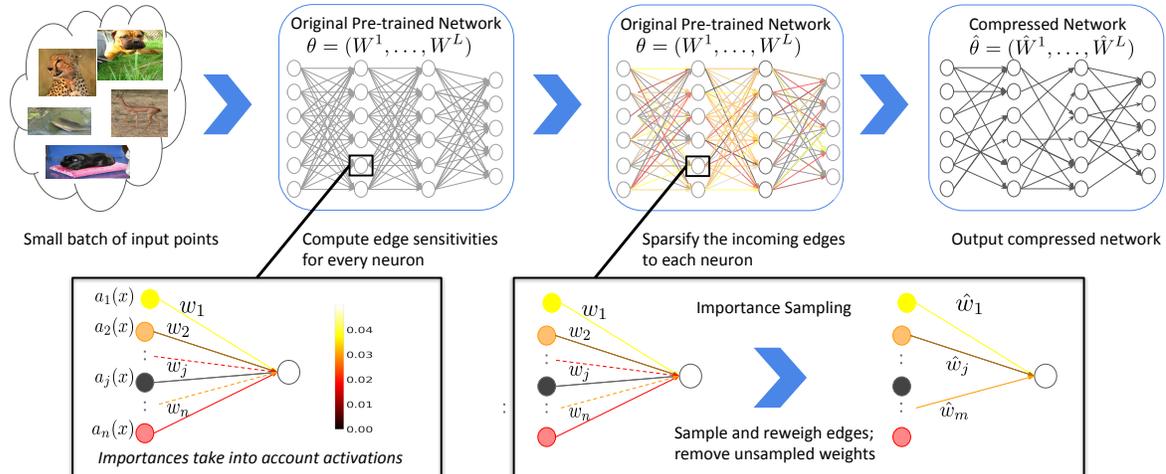


Figure 2: The overview of our method consisting of 4 parts. We use a small batch of input points to quantify the relative contribution (importance) of each edge to the output of each neuron. We then construct an importance sampling distribution over the incoming edges and sample a small set of weights for each neuron. The unsampled parameters are then discarded to obtain the resulting compressed network.

from the input space  $\mathcal{X}$  to the output space  $\mathcal{Y}$ . We consider the setting where a neural network  $f_\theta(\cdot)$  has been trained on a training set of independent and identically distributed (i.i.d.) samples from a joint distribution defined on  $\mathcal{X} \times \mathcal{Y}$ , yielding parameters  $\theta$ . We let  $\mathcal{D}$  denote the marginal distribution over the input space  $\mathcal{X}$  and define the size of the parameter tuple  $\theta$ ,  $\text{nnz}(\theta)$ , to be the number of all non-zero entries in the weight tensors  $W^1, \dots, W^L$ .

For given  $\varepsilon, \delta \in (0, 1)$ , our overarching goal is to use a randomized algorithm to generate a sparse reparameterization  $\hat{\theta}$  of  $\theta$  such that  $\text{nnz}(\hat{\theta}) \ll \text{nnz}(\theta)$  and for  $x \sim \mathcal{D}$  the reference network output  $f_\theta(x)$  can be approximated by  $f_{\hat{\theta}}(x)$  up to  $1 \pm \varepsilon$  entry-wise<sup>1</sup> multiplicative error with probability greater than  $1 - \delta$ , i.e.,  $\mathbb{P}_{\hat{\theta}, x}(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon)f_\theta(x)) \geq 1 - \delta$ .

## 4 Method

In this section, we present the core sparsification procedures used by our algorithm SiPP: Sensitivity-informed Provable Pruning (see Fig. 2). We introduce randomized (Alg. 1) and deterministic (Alg. 2) algorithms for pruning the parameters of a given filter and show how they can be adaptively mixed (Alg. 3) to generate sparse filters that incur smaller error (Sec. 4.2). The algorithm in full that utilizes the procedure to optimally allocate a given sampling budget across filters in all layers (as outlined in Sec. 5.5) can be found in the supplementary (Sec. A).

<sup>1</sup>For two tensors  $T_1, T_2$  of same dimensions,  $T_1 \in (1 \pm \varepsilon)T_2$  denotes that for each scalar entry  $t_1$  in tensor  $T_1$ ,  $t_1 \in (1 \pm \varepsilon)t_2$ , where  $t_2$  is the corresponding entry in  $T_2$ .

### 4.1 Parameter Importance via Sensitivity

Our pruning pipeline is shown in Fig. 2. Our pruning algorithm SiPP is based on a data-informed definition of *sensitivity*, i.e., parameter importance, that is computed by utilizing a small batch of validation points. To compute the sensitivity of each filter parameter, we obtain the set  $\mathcal{S} \subset \mathcal{X}$  by sampling a small batch of points from the set of validation points  $\mathcal{P} \subset \mathcal{X}$  (first step, Fig. 2). Using the points in  $\mathcal{S}$ , we define the sensitivity of each filter parameter as the maximum relative contribution of the parameter to the output of the next layer (second step, Fig. 2; Line 1, Alg. 3). In effect, important parameters that have a relatively large impact on the output have sensitivities close to 1, whereas those with a negligible influence on the output have sensitivities close to 0. The value of each patch  $a(x)$  can be computed from the output  $A^{\ell-1}(x)$  for each  $x \in \mathcal{S}$ .

---

#### Algorithm 1 RANDOMIZED

---

**Input:**  $\mathcal{I}, w, m$ : as in Alg. 3;  $(s_j)_{j \in \mathcal{I}}$ : sensitivities

- 1:  $q_j \leftarrow s_j / \sum_{k \in \mathcal{I}} s_k$ ;  $\forall j \in \mathcal{I}$
  - 2:  $\mathbf{K} \sim \text{MULTINOMIAL}(q, m)$ ;
  - 3: **for**  $j \in \mathcal{I}$  **do**
  - 4:  $\hat{w}_j \leftarrow \frac{\mathbf{K}_j w_j}{m q_j}$ ; {Entries are reweighted to ensure unbiasedness of our estimator}
  - 5: **end for**
  - 6: **return**  $\hat{w}$ ;
- 

Our core sampling scheme is denoted by RANDOMIZED (Alg. 1) and is illustrated in the third step in Fig. 2. RANDOMIZED uses the sensitivities to construct an importance sampling distribution over the filter pa-

rameters (Line 1, Alg. 1) and samples  $m$  parameters from the filter accordingly (Line 2). Intuitively, if a parameter has a high impact on the output, then it should be kept with high probability, and vice-versa. The sampled weights are then reweighed to ensure that the resulting sparse filter  $\hat{w}$  is an unbiased estimator for the output value. As we will see in Sec. 5, our formulation of sensitivity enables us to analytically bound the variance of our estimator.

---

**Algorithm 2** DETERMINISTIC
 

---

**Input:**  $\mathcal{I}, w, m$ : as in Alg. 3;  $(s_j)_{j \in \mathcal{I}}$ : sensitivities

- 1:  $s_m \leftarrow m^{\text{th}}$  largest value of  $\{s_j : j \in \mathcal{I}\}$ ;
  - 2:  $\mathcal{I}_m \leftarrow$  indices of the largest  $m$  values of  $(s_j)_{j \in \mathcal{I}}$  where ties are broken arbitrarily
  - 3: **for**  $j \in \mathcal{I}_m$  **do**
  - 4:    $\hat{w}_j \leftarrow w_j$ ;
  - 5: **end for**
  - 6: **return**  $\hat{w}$ ;
- 

---

**Algorithm 3** SPARSIFY( $\mathcal{I}, w, m, \delta, \mathcal{S}, \mathcal{A}, K$ )
 

---

**Input:**  $\mathcal{I}$ : set of filter parameter indices;  $w$ : filter to sparsify;  $m$ : number of parameters to keep,  $\delta \in (0, 1)$ : failure probability;  $\mathcal{S} \subseteq \mathcal{P}$ : batch of validation points;  $\mathcal{A}$ : set of patch maps;  $K$ : constant from Asm. 1 **Output:**  $\hat{w}$ : sparse weight tensor

- 1:  $s_j \leftarrow \max_{x \in \mathcal{S}} \max_{a(\cdot) \in \mathcal{A}} \frac{w_j a_j(x)}{\sum_{k \in \mathcal{I}} w_k a_k(x)}$ ;    $\forall j \in \mathcal{I}$   
    {Compute parameter importances}
  - 2:  $S \leftarrow \sum_{j \in \mathcal{I}} s_j$ ;    $\tilde{S} \leftarrow SK \log(8\eta/\delta)$ ;
  - 3:  $\varepsilon_{\text{randomized}} \leftarrow \frac{\tilde{S} + \sqrt{\tilde{S}(\tilde{S} + 6m)}}{m}$ ;
  - 4:  $\varepsilon_{\text{deterministic}} \leftarrow 3K \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_m)} s_j$ ; {where  $\mathcal{I}_m$  is the set of indices of the largest  $m$  values in  $(s_j)_{j \in \mathcal{I}}$ }
  - 5: **if**  $\varepsilon_{\text{randomized}} > \varepsilon_{\text{deterministic}}$  **then**
  - 6:   **return** DETERMINISTIC( $\mathcal{I}, w, m, (s_j)_{j \in \mathcal{I}}$ );
  - 7: **end if**
  - 8: **return** RANDOMIZED( $\mathcal{I}, w, m, (s_j)_{j \in \mathcal{I}}$ );
- 

## 4.2 Randomized and Deterministic Pruning

DETERMINISTIC (Alg. 2) is a variant of RANDOMIZED that, for a given sample size  $m$ , deterministically selects the parameters with the top  $m$  sensitivities and discards the rest. In Sec. 5 we show that in order to achieve a bound of  $\varepsilon \in (0, 1)$  on the relative error incurred by the sparsified filter  $\hat{w}$ , the number of samples  $m$  to our randomized algorithm must be of order  $\varepsilon^{-2}$ . This requirement on  $m$  cannot be avoided by sampling-based methods in general (Tropp et al., 2015). Thus, if the sample size is too small, our deterministic variant may perform better. To get the best of both worlds, we introduce SPARSIFY (Alg. 3), an algorithm that adaptively invokes the randomized or deterministic

procedure (Lines 3-8) by comparing the corresponding error bounds established in Sec. 5.

## 5 Analysis

In this section, we establish the theoretical guarantees of our core pruning procedure (Alg. 3) and state our main compression theorem. The full proofs can be found in the supplementary material.

### 5.1 Notation

We let the 4-dimensional tensor  $W^\ell \in \mathbb{R}^{\kappa_1^\ell \times \kappa_2^\ell \times c^{\ell-1} \times c^\ell}$  denote the weights of each convolutional layer  $\ell \in [L] = \{1, \dots, L\}$ , where  $\kappa_1^\ell$  and  $\kappa_2^\ell$  are the spatial dimensions of the kernel in layer  $\ell$  and  $c^{\ell-1}$  and  $c^\ell$  are the number of channels in layers  $\ell - 1$  and  $\ell$ , respectively. For a given input  $x = A^0(x) \in \mathcal{X}$  to the network, we let  $A^\ell(x) \in \mathbb{R}^{s_1^\ell \times s_2^\ell \times c^\ell}$  denote the input feature map of height  $s_1^\ell$  and width  $s_2^\ell$  to convolutional layer  $\ell + 1$ . We let  $\eta^\ell \in \mathbb{N}$  denote the number of scalar values of  $A^\ell(\cdot)$ , i.e.,  $\eta^\ell = s_1^\ell \cdot s_2^\ell \cdot c^\ell$ , and let  $\eta = \sum_{\ell=1}^L \eta^\ell$ . Finally, we let  $\rho^\ell = \text{nnz}(W^\ell)$  denote the number of parameters in the tensor  $W^\ell$  and let  $\rho^* = \max_{\ell \in [L]} \rho^\ell$ .

For each layer  $\ell \in [L]$ , let  $\mathcal{A}^{\ell-1}$  denote the set of *patch maps*  $a : \mathcal{X} \rightarrow \mathbb{R}^{\kappa_1^\ell \times \kappa_2^\ell \times c^{\ell-1}}$ , where  $a(x)$  maps  $x \in \mathcal{X}$  to the corresponding kernel patch in the input image  $A^{\ell-1}(x)$ . Note that  $|\mathcal{A}^{\ell-1}| = s_1^\ell \cdot s_2^\ell$  by definition. The pre-activation tensor of layer  $\ell$  is denoted by  $Z^\ell(x) \in \mathbb{R}^{s_1^\ell \times s_2^\ell \times c^\ell}$ , where for each entry  $(h, g, r) \in [s_1^\ell] \times [s_2^\ell] \times [c^\ell]$  and corresponding patch map  $a(\cdot) = \mathcal{A}_{h,g}^{\ell-1}$ , each entry is defined by the tensor dot product  $Z_{h,g,r}^\ell(x) = \langle W_{:, :, :, r}^\ell, a(x) \rangle$ . The input feature map is defined by the general non-linearity<sup>2</sup> function  $\phi$  between two convolutional layers, i.e.,  $A^\ell(x) = \phi(Z^\ell)$ .

### 5.2 Preliminaries

We begin by considering the sparsification of an arbitrary filter indexed by  $r \in [c^\ell]$  in an arbitrary layer  $\ell \in [L]$ , denoted by  $w = W_{:, :, :, r}^\ell \in \mathbb{R}^{\kappa_1^\ell \times \kappa_2^\ell \times c^{\ell-1}}$  and let  $\mathcal{I} = [\kappa_1^\ell] \times [\kappa_2^\ell] \times [c^{\ell-1}]$  denote the parameter index set. For ease of exposition, we henceforth omit explicit references to the layer  $\ell$ , and assume that the filter  $w$  consists of non-negative entries. The generalization to all weights can be found in the supplementary (Sec. B).

Our overarching goal is to keep weights of high influence and discard redundant ones by defining an appropriate

<sup>2</sup>For sake of simplicity we assume that  $\phi$  is the element-wise ReLU function, which implies that  $A^\ell(x) \geq 0$  entry-wise. Our analysis can be generalized – by adapting the input  $\varepsilon$  – to account for more general non-linearities, e.g., ones that include batch-normalization.

importance sampling distribution that captures the relative contribution of each parameter to the output of the convolutional layer. To gauge the relative importance of each weight of filter indexed by  $r \in [c^\ell]$  on the output  $Z(x)$ , we quantify the relative importance of each filter parameter  $j \in \mathcal{I}$  with respect to the corresponding patch maps  $\mathcal{A}$  and an input  $x \in \mathcal{X}$  as  $g_j(x) = \max_{a(\cdot) \in \mathcal{A}} w_j a_j(x) / \sum_{k \in \mathcal{I}} w_k a_k(x)$ .

To obtain sufficiently accurate approximations of parameter importance, rather than using the value of  $g_j(x)$  for a single point  $x \sim \mathcal{D}$ , we consider the maximum value of  $g_j(\cdot)$  over a batch of randomly drawn points  $\mathcal{S} \subset \mathcal{X}$  and formally define the *empirical sensitivity* of each parameter  $j \in \mathcal{I}$  as  $s_j = \max_{x \in \mathcal{S}} g_j(x) = \max_{x \in \mathcal{S}} \max_{a(\cdot) \in \mathcal{A}} w_j a_j(x) / \sum_{k \in \mathcal{I}} w_k a_k(x)$ .

To ensure that a *small* batch of points  $\mathcal{S}$  suffices for an accurate approximation of parameter importance, we impose the following mild assumption on the Cumulative Distribution Function (CDF) of  $g_j(x)$ . Traditional distributions such as the Gaussian, Uniform, and Exponential, among others, supported on the interval  $[0, 1]$  satisfy Assumption 1 with sufficiently small values of  $K$  and  $K'$ .

**Assumption 1** (Baykal et al. (2018)). *There exist universal constants  $K, K' > 0$  such that for all  $j \in \mathcal{I}$ , the CDF of the random variable  $g_j(x)$  for  $x \sim \mathcal{D}$ , denoted by  $F_j(\cdot)$ , satisfies  $F_j(M_j/K) \leq \exp(-1/K')$ , where  $M_j = \min\{y \in [0, 1] : F_j(y) = 1\}$ .*

### 5.3 Importance Sampling Bounds for Positive Weights

Lemma 1 establishes a core result that relates the sum of weighted activations with respect to the sparse filter  $\hat{w}$ ,  $\hat{z}(x) = \sum_{k \in \mathcal{I}} \hat{w}_k a_k(x)$ , to the ground-truth sum  $z(x) = \sum_{k \in \mathcal{I}} w_k a_k(x)$ .

**Lemma 1.** *For  $\delta \in (0, 1)$ , invoking RANDOMIZED as in Alg. 3 with  $m \in \mathbb{N}$  satisfying  $m > 8\tilde{S}$ , and a set  $\mathcal{S} \subset \mathcal{X}$  composed of  $\lceil K' \log(2\rho^*/\delta) \rceil$  i.i.d. points drawn from  $\mathcal{D}$  generates a filter  $\hat{w}$  such that  $\text{nnz}(\hat{w}) \leq m$  and for an arbitrary patch map  $a(\cdot) \in \mathcal{A}$  and  $x \sim \mathcal{D}$ ,*

$$\mathbb{P}(|\hat{z}(x) - z(x)| \geq \varepsilon_m z(x)) \leq \delta,$$

where  $\hat{z}(x)$  and  $z(x)$  are with respect to patch map  $a(\cdot)$ ,

$$\varepsilon_m = \left( \sqrt{\frac{\tilde{S}}{m} \left( \frac{\tilde{S}}{m} + 6 \right)} + \frac{\tilde{S}}{m} \right) \in (0, 1),$$

$\tilde{S} = SK \log(4/\delta)$ , and  $S = \sum_{j \in \mathcal{I}} s_j$ .

As an immediate corollary to Lemma 1, we obtain a bound on the relative error in terms of the sample complexity, which we will subsequently use to optimize

the sample size across filters in different layers with respect to an allotted sample budget (see Sec. 5.5).

Likewise, we establish a guarantee on the output of a sparse filter  $\hat{w}$  generated in a deterministic fashion according to Alg. 2.

**Lemma 2.** *In the context of Lemma 1, invoking DETERMINISTIC( $\mathcal{I}, w, m, (s_j)_{j \in \mathcal{I}}$ ) generates a filter  $\hat{w}$  such that for an arbitrary patch map  $a(\cdot) \in \mathcal{A}$  and  $x \sim \mathcal{D}$ ,  $\mathbb{P}(|\hat{z}(x) - z(x)| \geq \varepsilon_m z(x)) \leq \frac{\delta}{2}$ , where  $\varepsilon_m = 3K \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_m)} s_j$  and  $\mathcal{I}_m$  is the set of indices of the largest  $m$  values in  $(s_j)_{j \in \mathcal{I}}$ .*

The following theorem follows immediately from Lemmas 1 and 2 and establishes the error guarantee of our hybrid approach that adaptively mixes deterministic and sampling-based constructions of  $\hat{w}$ .

**Theorem 3.** *For any  $\delta \in (0, 1)$  and an integral sample size  $m$  satisfying  $m > 8\tilde{S}$ , SPARSIFY( $\mathcal{I}, w, m, \delta, \mathcal{S}, \mathcal{A}, K$ ) returns a filter  $\hat{w}$  such that  $\text{nnz}(\hat{w}) \leq m$  and  $\mathbb{P}(|\hat{z}(x) - z(x)| \geq \varepsilon_m z(x)) \leq \delta$ , where  $\varepsilon_m = \min \left\{ \sqrt{\frac{\tilde{S}}{m} \left( \frac{\tilde{S}}{m} + 6 \right)} + \frac{\tilde{S}}{m}, 3K \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_m)} s_j \right\}$ .*

### 5.4 Main Theorem

In this section we outline the details of generalizing Theorem 3 which pertains to preserving a single output of a filter to obtain an error guarantee for the output of entire compressed network. Our analysis of preserving a single output  $z(x)$  from Sec. 5.3 can be generalized to establish bounds for all weights by decomposing each weight into its positive and negative components, i.e.,  $w = w^+ - w^-$ ,  $w^+, w^- \geq 0$ , and applying our analytical results established in the previous subsection to each component. We then apply this result to each output of every filter in a layer to obtain layer-wise guarantees, i.e.,  $\hat{Z}^\ell(x) \in (1 \pm \varepsilon)Z(x)^\ell$ , and subsequently conduct a careful error propagation analysis through the layers to obtain our main theorem. Our main result is stated below for completeness. We refer the reader to the supplementary (Sec. B) for the proofs and technical details in full.

**Theorem 4.** *For given  $\varepsilon, \delta \in (0, 1)$  and a set of parameters  $\theta = (W^1, \dots, W^L)$ , SiPP (Alg. 4 in supplementary) generates a set of compressed parameters  $\hat{\theta} = (\hat{W}^1, \dots, \hat{W}^L)$  such that  $\mathbb{P}_{\hat{\theta}, x \sim \mathcal{D}}(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon)f_\theta(x)) \geq 1 - \delta$ , and*

$$\text{nnz}(\hat{\theta}) = \mathcal{O} \left( \sum_{\ell=1}^L \sum_{r \in [c]^\ell} \frac{L^2 (\Delta^{\ell-1})^2 S_r^\ell K \log(\eta/\delta)}{\varepsilon^2} \right),$$

where  $S_r^\ell$  is the sum of sensitivities with respect to filter  $r \in [c]^\ell$  and layer  $\ell \in [L]$ .

## 5.5 Informed Sample Size Allocation

Suppose we are given a sampling budget  $\mathcal{B} \in \mathbb{N}$  as input that constrains our compression such that  $\text{nnz}(\hat{\theta}) \leq \mathcal{B}$ . For clarity of exposition, we will present the sample allocation procedure for a single 4 dimensional tensor  $W^\ell$  that contains non-negative entries, as the generalization of our procedure to all weights and layers is straightforward. Now let  $\delta \in (0, 1)$  and note that by application of Lemma 1 and the union bound, we know that for filter  $r \in [c^\ell]$ , if RANDOMIZED is invoked with an integral sample size  $m_r$  satisfying  $m_r > 8\tilde{S}_r$ , then the probability of the event that  $\exists a(\cdot) \in \mathcal{A} : |\hat{z}(x) - z(x)| \geq \varepsilon_r(m_r)z(x)$ , is bounded above by  $|A|\delta/2\eta$ , where  $\varepsilon_r(m_r)$  and  $\tilde{S}_r$  are as in Alg. 3.

Our objective is to pick the optimal sample size  $m_r \in \mathbb{N}_+$  for each filter  $r \in [c^\ell]$  subject to the sampling constraint  $\mathcal{B}$  such that the sum of relative errors over all  $r \in [c^\ell]$  is minimized. To perform this integral optimization efficiently, we consider the fractional relaxation of the integer program, i.e.,

$$\begin{aligned} \min_{(m_1, \dots, m_{c^\ell})} & \sum_{r \in [c^\ell]} \varepsilon_r(m_r) \\ \text{s.t.} & \sum_{r \in [c^\ell]} m_r \leq \mathcal{B} \quad \text{and} \quad \forall r \in [c^\ell] \quad m_r \geq 8\tilde{S}_r. \end{aligned}$$

This is a convex optimization problem since each summand  $\varepsilon_r(m_r)$  is convex. In particular, it is an instance of the classic water filling problem (Boyd and Vandenberghe, 2004) and can be solved efficiently in practice using standard convex optimization libraries (Diamond and Boyd, 2016). The optimal solution of the fractional relaxation can then be rounded to the nearest integer using a randomized rounding scheme (Srinivasan, 1999) to generate provably competitive integral solutions.

## 6 Experiments

In this section, we evaluate and compare the performance of our algorithm, SiPP, on pruning fully-connected, convolutional, and residual networks. For each network, we assessed the loss in accuracy with respect to both *prune-only* and *iterative prune + retrain* compression schemes. We qualitatively compared the algorithmic properties of our approach and evaluated its ability to accurately capture the importance of each parameter and wisely allocate the sampling budget across layers. All experiments were conducted on a NVIDIA GTX 1080Ti with 11GB RAM using PyTorch Paszke et al. (2017).

### 6.1 Experimental Setup

**Architectures and data sets** As our baseline comparisons, we considered the fully-connected network

LeNet300-100 (LeCun et al., 1998) and the convolutional network LeNet5 (LeCun et al., 1990), the residual networks Resnet20 (He et al., 2016) and WideResNet16 (WRN16) (Zagoruyko and Komodakis, 2016). All networks were trained with Stochastic Gradient Descent (SGD) with cross-entropy loss, where 10% of the training data set was used as the validation set and the remainder was used for training. Additional details of the experimental setup can be found in Sec. E of the supplementary.

**Compared Algorithms** We compared the performance of our algorithm to the following pruning approaches. We refer the reader to the supplementary (Sec. F) for more details on each method.

1. *Singular Value Decomposition (SVD)*: based on decomposing the weight tensors into their low-rank counterparts (Denton et al., 2014, Yu et al., 2017)
2.  $\ell_1 + \ell_2/2$  (also labeled "Norm") (Kundu and Drineas, 2014): a randomized matrix sparsification scheme that samples weights according to a convex combination of their  $\ell_1$ - and  $\ell_2$ -norms
3. *Weight Thresholding* (also labeled "Weight") (Han et al., 2015): based on discarding weights with absolute values below a threshold

### 6.2 Pruning without Retraining

We first consider the baseline effectiveness of each pruning algorithm in generating an accurate, compressed model *without retraining*. For each network architecture, we prune the network down to various specified sparsity ratio, and report the accuracy of the pruned network on the test data set. We evaluate the performance of each algorithm on 15 equally-spaced sparsity ratios for each network, and average the results over 3 repetitions and 3 trained networks (with differing random seeds) per sparsity ratio.

The first row of Fig. 3 depicts the results of our evaluations and comparisons for varying levels of network sparsity. Our algorithm, SiPP, consistently outperforms competing approaches in generating more accurate pruned networks for virtually all of the target sparsity ratios. The improvement over Weight Thresholding (WT) is modest, but consistent across all of our evaluations for the prune-only setting. We highlight that our pruning algorithm generates models with commensurate accuracy at comparatively high levels of sparsity – e.g., 5x compression for LeNet300-100 and LeNet5 – even without a fine-tune step. The shaded regions corresponding to values within one standard deviation also show that our algorithm’s performance exhibits lower variance across varying trials and trained networks.

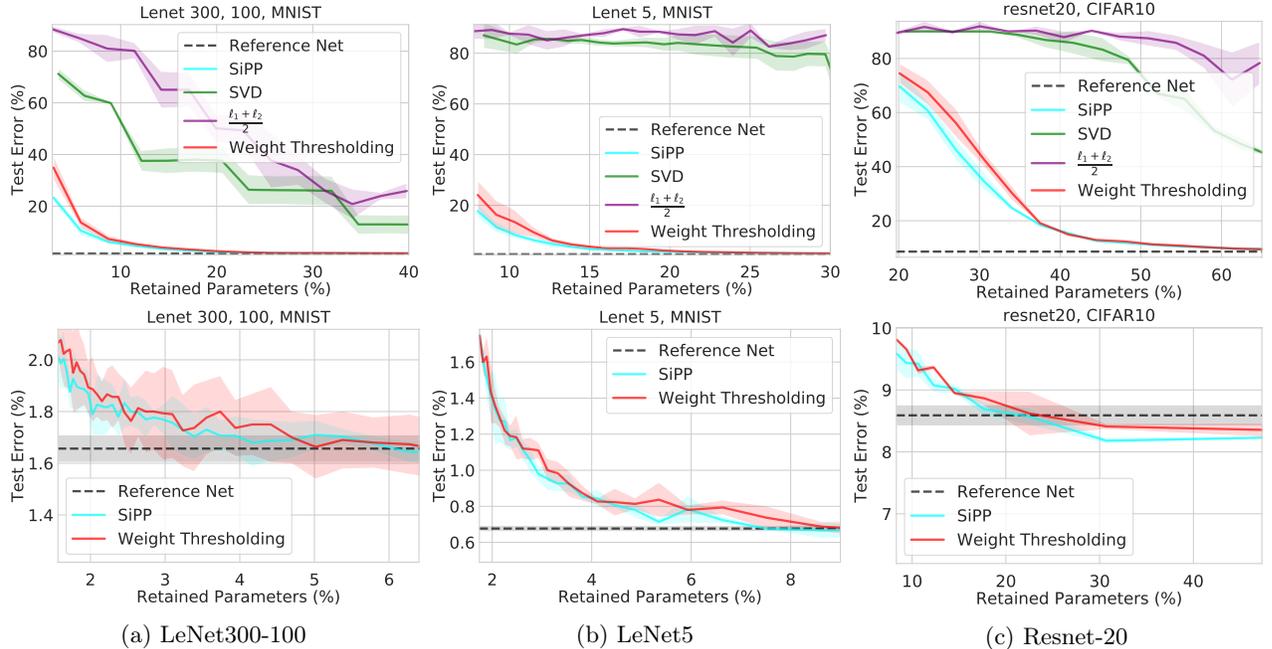


Figure 3: The performance of the evaluated pruning methods against various architectures and data sets, where the first and second row correspond to prune-only and iterative prune + retrain schemes. The x-axis denotes the percentage of the parameters that are *kept* after pruning, i.e., the values on the x axis correspond to  $1 - (\text{sparsity ratio})$ . The shaded region depicts the values within one standard deviation of the mean. Our algorithm, SiPP, reliably – and with lower variance – outperforms or matches the performance of the compared methods for various levels of target sparsity.

### 6.3 Iterative Pruning with Fine-tuning

In practice, it is common to combine the pruning step with a fine-tuning step in an iterative fashion in order to improve the pruned model’s accuracy. In this experiment, we consider the objective of compressing a given trained network as much as possible while retaining the predictive power of the original model. We use a harmonic progression of the sparsity ratios to iteratively prune and fine-tune the networks (see Sec. E in the supplementary for details).

The second row of Fig. 3 summarizes the results of the prune + retrain procedure that was iterated on until the pruned model was approximately 1-2% and 9% of the original model’s size for LeNet and ResNet20 architectures, respectively. The results were averaged across 3 repetitions and 3 trained networks. The figures show the comparisons with Weight Thresholding (WT) (Han et al., 2015) with the plots of the other approaches omitted due to the fact that WT’s pruning performance was consistently and by far the best among the baseline approaches. Our results are consistent with the fact that, to date, WT has been the prominent choice for network pruning applications (Frankle and Carbin, 2018) and has been shown to be the most practical and time-efficient way of reducing the number of network parameters among contemporary pruning algorithms (Pitas et al., 2019).

Our empirical evaluations show that our algorithm reliably outperforms or matches the performance of WT. For example, for LeNet5, a network compressed with SiPP achieves a test error that is within 0.2% of the original network with  $\approx 95\%$  sparsity, compared to only  $\approx 92\%$  sparsity for WT. A similar trend can be seen for LeNet300-100, where we attain commensurate accuracy for  $\approx 96\%$  sparsity, compared to 95% for WT. For all of the scenarios evaluated, our method also exhibits lower variance and tends to perform better at extremely high sparsities. For LeNet300-100, for example, we observe that our approach is strictly better for sparsities above  $\approx 97\%$ .

Our algorithm exhibits similar performance increases over WT as we consider deeper and wider networks. For Resnet20 (third column, Fig. 3), SiPP achieves a modest, but noticeable and consistent improvement over WT: we can prune over  $\approx 77\%$  of the parameters with commensurate accuracy, compared to 75% for WT. We achieve improved accuracy relative to the accuracy of the original network and WT for sparsities of up to 70%. Similarly, Fig. 4 depicts the performance of our algorithm on WideResnet-16-8, a wide residual network. We observe that in this scenario, SiPP significantly outperforms WT, especially at high sparsities: SiPP preserves the original model’s accuracy at around 40% sparsity, compared to 25% for WT (not shown in the figure).

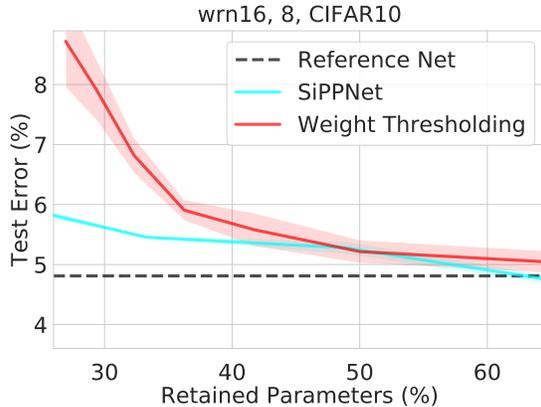


Figure 4: The performance of our algorithm compared to that of Weight Thresholding for iteratively pruning a Wide Residual Network (WRN16) trained on CIFAR-10. SiPP generates sparser, more accurate models than the best-performing baseline method.

#### 6.4 Retaining Important Connections

Recent work suggests that the value of network pruning may lie in identifying and retaining essential *network connections*, rather than in keeping the *values* of the weights in the original trained network (Frankle et al., 2019, Liu et al., 2018). Here, we investigate whether SiPP is merely keeping the weights with high values (akin to WT), or whether it can be used to identify and retain the essential network connections. To do so, we modify the iterative prune + retrain process from the previous subsection so that in each iteration, after pruning the network, the parameters that are kept are randomly initialized prior to the retraining procedure. We refer to this process as *iterative prune + random-init + retrain*. Since the retained weights are randomly initialized before the retraining procedure, the values of the weights prior to pruning have no influence on the resulting fine-tuned network – only the retained *connections*, i.e., edges, matter.

Fig. 5 summarizes the results of our evaluations – averaged over 3 networks and 3 repetitions – on the LeNet5 network trained on the MNIST data set. The plot shows that our algorithm’s performance is consistently better than that of WT, and this relative improvement in performance becomes even more pronounced at extreme sparsities. Namely, as the sparsity of the model exceeds  $\approx 96.5\%$ , the connections that are identified by WT can no longer be trained successfully in isolation to obtain an accurate model, as can be noted by the rapid spike in test error. On the other hand, SiPP’s performance is only slightly worse relative to its performance under the traditional *prune + retrain* scheme (compare to Fig. 3b), and its variance is significantly lower than that of WT.

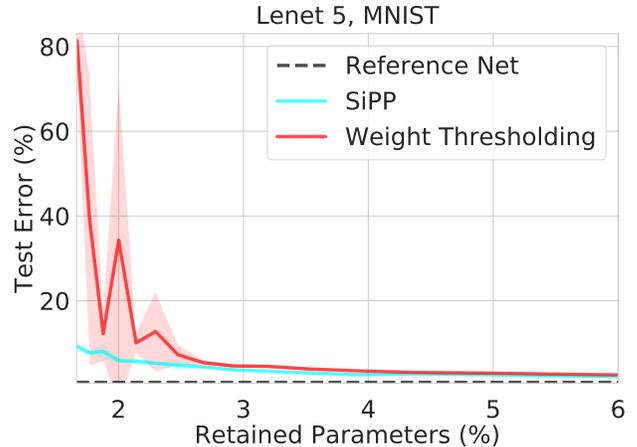


Figure 5: Results of the *iterative prune + random-init + retrain* scheme on LeNet5 trained on the MNIST data set. The performance of SiPP is strictly better than that of WT, and SiPP’s relative effectiveness over WT improves as we increase the sparsity ratio.

## 7 Conclusion

In this work, we presented a practical network pruning method, SiPP, that identifies important network connections and prunes redundant ones in a data-informed way. Our analysis establishes provable guarantees that quantify the trade-off between the desired model sparsity and resulting accuracy of the pruned model. SiPP is easy-to-implement, requires minimal hyper-parameter tuning, and is computationally efficient – requiring on the order of a minute to prune even large networks. Our empirical evaluations show that SiPP *reliably* sparsifies modern network architectures with minimal degradation in model accuracy, and tends to offer a modest, but noticeable improvement over competing approaches.

SiPP’s favorable performance in scenarios involving random re-initialization suggests that our method inherently considers the crucial pathways through the network, and does not merely operate by considering the properties, e.g., values, of the network parameters alone. Our empirical evaluations suggest that our approach may be more effective than contemporary magnitude-based approaches in unearthing essential connections that can be trained successfully in isolation, and more generally, they highlight the potential application of SiPP to discovering winning lottery tickets (Frankle et al., 2019) and efficient architecture search. As future work, we plan to rigorously investigate the effectiveness of SiPP on these applications, and on compressing even larger and deeper networks (e.g. ImageNet experiments).

## References

- Achlioptas, D., Karnin, Z., and Liberty, E. (2013). Matrix entry-wise sampling: Simple is best. *Submitted to KDD*, 2013(1.1):1–4. [18](#)
- Aghasi, A., Abdi, A., Nguyen, N., and Romberg, J. (2017). Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, pages 3180–3189. [2](#)
- Alvarez, J. M. and Salzmann, M. (2017). Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867. [2](#)
- Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. (2018). Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*. [2](#)
- Baykal, C., Liebenwein, L., Gilitschenski, I., Feldman, D., and Rus, D. (2018). Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint arXiv:1804.05345*. [2](#), [5](#), [12](#), [13](#), [15](#)
- Baykal, C., Liebenwein, L., and Schwarting, W. (2017). Training support vector machines using coresets. *arXiv preprint arXiv:1708.03835*. [16](#)
- Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press. [6](#)
- Braverman, V., Feldman, D., and Lang, H. (2016). New frameworks for offline and streaming coreset constructions. *arXiv preprint arXiv:1612.00889*. [2](#)
- Cheng, Y., Yu, F. X., Feris, R. S., Kumar, S., Choudhary, A., and Chang, S.-F. (2015). An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865. [2](#)
- Choromanska, A., Choromanski, K., Bojarski, M., Jabara, T., Kumar, S., and LeCun, Y. (2016). Binary embeddings with structured hashed projections. In *International Conference on Machine Learning*, pages 344–353. [2](#)
- Denil, M., Shakibi, B., Dinh, L., Ranzato, M. A., and de Freitas, N. (2013). Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156. [2](#)
- Denton, E., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR*, abs/1404.0736. [2](#), [6](#)
- Diamond, S. and Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5. [6](#)
- Dong, X., Chen, S., and Pan, S. (2017). Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874. [2](#)
- Drineas, P. and Zouzias, A. (2011). A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. *Information Processing Letters*, 111(8):385–389. [2](#), [18](#)
- Feldman, D. and Langberg, M. (2011). A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578. ACM. [2](#)
- Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Training pruned neural networks. *arXiv preprint arXiv:1803.03635*. [7](#)
- Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019). The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*. [8](#)
- Fung, W. S., Hariharan, R., Harvey, N. J., and Panigrahi, D. (2011). A general framework for graph sparsification. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 71–80. ACM. [2](#)
- Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149. [1](#), [2](#), [6](#), [7](#), [18](#)
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. [6](#)
- Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360*. [2](#)
- Ioannou, Y., Robertson, D., Shotton, J., Cipolla, R., and Criminisi, A. (2015). Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*. [2](#)
- Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*. [2](#)
- Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., and Shin, D. (2015). Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*. [2](#)
- Kundu, A. and Drineas, P. (2014). A note on randomized element-wise matrix sparsification. *arXiv preprint arXiv:1404.0320*. [1](#), [2](#), [6](#), [18](#)

- Lebedev, V. and Lempitsky, V. (2016). Fast convnets using group-wise brain damage. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2554–2564. IEEE. 2
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 6
- LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605. 2, 6
- Lee, N., Ajanthan, T., and Torr, P. H. (2018). Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*. 2
- Lin, J., Rao, Y., Lu, J., and Zhou, J. (2017). Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2178–2188. 2
- Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2018). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*. 8
- Munteanu, A., Schwiegelshohn, C., Sohler, C., and Woodruff, D. P. (2018). On coresets for logistic regression. *arXiv preprint arXiv:1805.08571*. 12
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*. 6
- Pitas, K., Davies, M., and Vandergheynst, P. (2019). Revisiting hard thresholding for dnn pruning. *arXiv preprint arXiv:1905.08793*. 7
- Sindhwani, V., Sainath, T., and Kumar, S. (2015). Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096. 2
- Srinivasan, A. (1999). Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29(2):648–670. 6
- Tai, C., Xiao, T., Zhang, Y., Wang, X., et al. (2015). Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*. 2
- Tropp, J. A. et al. (2015). An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230. 4
- Vershynin, R. (2016). High-dimensional probability. *An Introduction with Applications*. 12
- Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082. 2
- Yu, X., Liu, T., Wang, X., and Tao, D. (2017). On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379. 2, 6
- Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*. 6
- Zhao, L., Liao, S., Wang, Y., Tang, J., and Yuan, B. (2017). Theoretical properties for neural networks with weight matrices of low displacement rank. *CoRR*, abs/1703.00144. 2

## A SiPP: Sensitivity-informed Provable Pruning

Our algorithm SiPP in full is given below.

---

### Algorithm 4 SiPP( $\theta, \delta, \mathcal{B}, \mathcal{P}, K, K'$ )

---

**Input:**  $\theta = (W^1, \dots, W^L)$ : parameters of the original uncompressed neural network;  $\delta \in (0, 1)$ : failure probability;  $\mathcal{B} \in \mathbb{N}$ : sampling budget;  $\mathcal{P} \subseteq \mathcal{X}$ : a set of  $n$  i.i.d. validation points drawn from  $\mathcal{D}$ ;  $K, K' > 0$ : universal constants from Asm. 1

- 1:  $\mathcal{S} \leftarrow$  Uniform sample (without replacement) of  $\lceil K' \log(4\eta\rho^*/\delta) \rceil$  points from  $\mathcal{P}$ ;
  - 2: Populate the patches  $\mathcal{A}^\ell(x)$  for each  $x \in \mathcal{X}$  and layer  $\ell \in [L]$ ;
  - 3: Compute the sensitivities for all filter parameters in all layers as in Alg. 3, Line 3 using the computed  $\mathcal{A}^\ell(x)$  values;
  - 4: Allocate the sampling budget  $\mathcal{B}$  to obtain the sample size  $m_r^{\ell+}, m_r^{\ell-}$  for each  $r \in [c^\ell], \ell \in [L]$  as in Sec. 5.5;
  - 5: **for**  $\ell \in [L]$  **do**
  - 6:    $\mathcal{I} \leftarrow [\kappa_1^\ell] \times [\kappa_2^\ell] \times [c^{\ell-1}]$ ;
  - 7:    $\hat{W}^\ell \leftarrow \mathbf{0} \in \mathbb{R}^{\kappa_1^\ell \times \kappa_2^\ell \times c^{\ell-1} \times c^\ell}$ ; {Initialize a 4D null tensor}
  - 8:   **for**  $r \in [c^\ell]$  **do**
  - 9:      $w \leftarrow W_{:, :, :, r}^\ell$ ; {Get the  $r^{\text{th}}$  filter}
  - 10:      $\mathcal{I}^+ \leftarrow \{j \in \mathcal{I} : w_j > 0\}$ ;
  - 11:      $\mathcal{I}^- \leftarrow \{j \in \mathcal{I} : w_j < 0\}$ ;
  - 12:      $\hat{w}^+ \leftarrow \text{SPARSIFY}(\mathcal{I}^+, w, m_r^{\ell+}, \delta, \mathcal{S}, \mathcal{A}^{\ell-1}, K)$ ;
  - 13:      $\hat{w}^- \leftarrow \text{SPARSIFY}(\mathcal{I}^-, -w, m_r^{\ell-}, \delta, \mathcal{S}, \mathcal{A}^{\ell-1}, K)$ ;
  - 14:      $\hat{W}_{:, :, :, r}^\ell \leftarrow \hat{w}^+ - \hat{w}^-$ ; {Store the resulting sparse filter};
  - 15:   **end for**
  - 16: **end for**
  - 17: **return**  $\hat{\theta} = (\hat{W}^1, \dots, \hat{W}^L)$ ;
- 

For our experiments, we set a small failure probability for  $\delta$ , i.e.,  $\delta = 1.0 \times 10^{-16}$  and found that our hybrid approach tended to select the deterministic sub-procedure based on the error bounds most of the time.

## B Generalizing to all Weights

In this section we generalize our established results to all weights by resolving three main challenges: (i) relaxing the requirement that the weights are non-negative, (ii) extending our probabilistic bounds to all  $r \in [c^\ell]$  filters in each layer to establish a guarantee on all outputs of a layer, and (iii) propagating and bounding the layer-wise error across the layers of the network.

For ease of presentation, our analysis focuses on the *upper bound* of the error incurred by our sampling-based approach (Alg. 1), which also serves as an upper bound of error our hybrid SPARSIFY (Alg. 3) procedure since SPARSIFY only invokes DETERMINISTIC in case the upper bound of RANDOMIZED exceeds that of DETERMINISTIC. To this end, we present a useful corollary that provides a sufficient sampling complexity to obtain an  $(\varepsilon, \delta)$  approximation.

**Corollary 5.** *For any given  $\varepsilon, \delta \in (0, 1)$ , in the context of Lemma 1, invoking RANDOMIZED with sample size*

$$m = \lceil (6 + 2\varepsilon) S K \log(4/\delta) \varepsilon^{-2} \rceil$$

*generates a filter  $\hat{w}$  such that for any arbitrary patch map  $a(\cdot) \in \mathcal{A}$  and  $x \sim \mathcal{D}$ ,*

$$\mathbb{P} \left( \sum_{k \in \mathcal{I}} \hat{w}_k a_k(x) \notin (1 \pm \varepsilon) \sum_{k \in \mathcal{I}} w_k a_k(x) \right) \leq \delta.$$

Our analytical results from the previous subsection can be extended in a straightforward way by decomposing the filter  $w$  into its corresponding positive and negative components, i.e.,  $w = w^+ - w^-$ , where  $w^+, w^- \geq 0$  entry-wise with respective index sets  $\mathcal{I}_+, \mathcal{I}_- \subseteq \mathcal{I}$ . Using this decomposition, we can apply our previous analysis to obtain bounds on the approximation  $z^+(x) = \sum_{k \in \mathcal{I}_+} w_k a_k(x) \geq 0$  and  $z^-(x) = \sum_{k \in \mathcal{I}_-} (-w_k) a_k(x) \geq 0$  for both positive and negative weights by separately defining empirical sensitivity over and choosing (weighted) samples from  $w^+$  and  $w^-$  to obtain  $\hat{w}^+$  and  $\hat{w}^-$ , respectively. The end result is a sparsified tensor defined by  $\hat{w} = \hat{w}^+ - \hat{w}^-$ .

This decomposition, however, requires us to be careful in our error bounds when considering the approximation guarantee of  $\hat{w}$ . In particular, invoking Corollary 5 provides sparse filters  $\hat{w}^+$  and  $\hat{w}^-$  that  $(1 \pm \varepsilon)$  approximate  $z^+$  and  $z^-$  by  $\hat{z}^+$  and  $\hat{z}^-$ , respectively<sup>3</sup>. However, this does not provide a  $(1 \pm \varepsilon)$  approximation for  $z = z^+ - z^-$ . To see this, note that our approximation is defined as  $\hat{z} = \hat{z}^+ - \hat{z}^-$ . Now consider the case that  $z^+, z^- > 0$  and  $\hat{z}^+ = (1 + \varepsilon)z^+$  and  $z^- = (1 - \varepsilon)z^-$ . This yields an overall error of  $\varepsilon(z^+ + z^-) > \varepsilon|z|$  for  $z$ .

To tackle this complexity, we let  $\Delta(x, a(\cdot)) = (z^{+(x)+z^-(x)})/|z(x)| - z^+(x)$  and  $z^-(x)$  are implicitly defined with respect to patch  $a(\cdot)$  – denote the complexity measure<sup>4</sup> with respect to  $x \in \mathcal{X}$  and patch  $a(\cdot) \in \mathcal{A}$ . For a particular filter  $r \in [c^\ell]$ , we let  $\tilde{\Delta}_r(x) = \max_{a(\cdot) \in \mathcal{A}} \Delta(x, a(\cdot))$ .

**Lemma 6.** *Let  $x \sim \mathcal{D}$ , let  $\Delta_r \geq \tilde{\Delta}_r(x)$  be a constant, and let  $\mathcal{S} \subseteq \mathcal{X}$  be a set of i.i.d.  $[K' \log(4\eta\rho^*/\delta)]$  Consider invoking RANDOMIZED separately with  $(w, \mathcal{I}^+)$  and  $(-w, \mathcal{I}^+)$  each with sample size  $m = \left\lceil (6 + 2\varepsilon)\Delta_r^2 \left( \sum_{j \in \mathcal{I}} s_j \right) K \log(8\eta/\delta)\varepsilon^{-2} \right\rceil$  as inputs, where the sum of sensitivities is with respect to  $\mathcal{I}^+$  or  $\mathcal{I}^-$ , to obtain the outputs  $\hat{w}^+$  and  $\hat{w}^-$ . Then, for any arbitrary patch map  $a(\cdot) \in \mathcal{A}$ ,  $\hat{w} = \hat{w}^+ - \hat{w}^-$  satisfies*

$$\mathbb{P}\left(\exists a(\cdot) \in \mathcal{A} : \sum_{k \in \mathcal{I}} \hat{w}_k a_k(x) \notin (1 \pm \varepsilon) \sum_{k \in \mathcal{I}} w_k a_k(x)\right) \leq \delta'$$

where  $\delta' = |\mathcal{A}|\delta/\eta$ ,  $\text{nnz}(\hat{w}) \leq (7 + 2\varepsilon)\Delta_r^2 S_r K \log(8\eta/\delta)\varepsilon^{-2}$ , and  $S_r$  is the sum of sensitivities with respect to filter  $r$ .

Note that from a practical standpoint, the value of  $\Delta_r$  is not required for our algorithm since  $\Delta_r$  is only used to compute the number of samples necessary with respect to  $\varepsilon$  and  $\delta$ , but the number of samples is already provided as input<sup>5</sup>. For the next lemma, let  $\hat{W}^\ell = (\hat{w}_1, \dots, \hat{w}_{c^\ell})$  denote the 4-dimensional tensor generated by invoking RANDOMIZED with  $W_{:::,:::,r}^\ell$  as input as in Lemma 6 to obtain  $\hat{w}_r = \hat{w}_r^+ - \hat{w}_r^-$  for each filter  $r \in [c^\ell]$ , and let  $\hat{Z}^\ell(x)$  denote the approximation of  $Z^\ell(x) \in \mathbb{R}^{s_1 \times s_2 \times c^\ell}$  when  $W_{:::,:::,r}^\ell$  is replaced by  $\hat{W}^\ell$ . The following lemma provides a layer-wise approximation guarantee for each entry of the output.

**Lemma 7.** *For any  $\varepsilon, \delta \in (0, 1)$  and layer  $\ell \in [L]$ , the approximate output  $\hat{Z}^\ell(x) \in \mathbb{R}^{s_1 \times s_2 \times c^\ell}$  with respect to tensor  $\hat{W}^\ell$  satisfies  $\mathbb{P}(\hat{Z}^\ell(x) \notin (1 \pm \varepsilon)Z^\ell(x)) \leq \frac{\delta\eta^\ell}{\eta}$ , where  $\text{nnz}(\hat{W}^\ell) \leq \sum_{r \in [c^\ell]} (7 + 2\varepsilon)\Delta_r^2 S_r K \log(8\eta/\delta)\varepsilon^{-2}$ .*

Lemma 7 establishes that the output of any arbitrary layer  $\ell$  can be  $(1 \pm \varepsilon)$ -approximated (entry-wise) by replacing the tensor  $W^\ell$  with the corresponding sparsified tensor  $\hat{W}^\ell$ . Propagating the error through the  $L$  convolutional layers can be performed by repeatedly applying a variant of Corollary 5 that takes into account the fact that compressing a layer  $\ell$  affects the value of the patch maps  $a(\cdot) \in \mathcal{A}^\ell$  in the next layer, i.e., instead of  $a(\cdot)$  appearing in the sum of  $z(x)$ , we have an approximation  $\hat{a}(\cdot)$ .

To deal with this propagation, we define  $\Delta^{\ell \rightarrow} = \max_{r \in [c^\ell]} \prod_{k=\ell}^L \Delta_r^k$ . An appropriate error propagation analysis implies that for each layer  $\ell$  and filter  $r \in [c^\ell]$ , if the sampling complexity of Lemma 6 is adjusted to be

$$m = \left\lceil 32 L^2 (\Delta^{\ell \rightarrow})^2 K \log(8\eta/\delta) \varepsilon^{-2} \sum_{j \in \mathcal{I}} s_j \right\rceil \quad (1)$$

then the output of the network is  $(1 \pm \varepsilon)$  approximated with probability at least  $1 - \delta$ . This result is formalized in the theorem below and the details of the error propagation proof can be found in Baykal et al. (2018).

**Theorem 4.** *For given  $\varepsilon, \delta \in (0, 1)$  and a set of parameters  $\theta = (W^1, \dots, W^L)$ , SiPP (Alg. 4 in supplementary) generates a set of compressed parameters  $\hat{\theta} = (\hat{W}^1, \dots, \hat{W}^L)$  such that  $\mathbb{P}_{\hat{\theta}, x \sim \mathcal{D}}(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon)f_\theta(x)) \geq 1 - \delta$ , and*

$$\text{nnz}(\hat{\theta}) = \mathcal{O}\left(\sum_{\ell=1}^L \sum_{r \in [c^\ell]} \frac{L^2 (\Delta^{\ell \rightarrow})^2 S_r^\ell K \log(\eta/\delta)}{\varepsilon^2}\right),$$

where  $S_r^\ell$  is the sum of sensitivities with respect to filter  $r \in [c^\ell]$  and layer  $\ell \in [L]$ .

<sup>3</sup>where we omitted explicit references to  $x$  for clarity

<sup>4</sup>A similar complexity measure was introduced in Munteanu et al. (2018) for logistic regression using coresets.

<sup>5</sup>If desired, an appropriate value of  $\Delta_r$  in Lemma 6 can be approximated using the subset of points  $\mathcal{S}$  under a standard sub-exponentiality assumption on  $\Delta(x, a(\cdot))$  Vershynin (2016).

## C Proofs of the Analytical Results in Section 5.3

In the following, we state the proofs of Section 5.3, where we establish that Algorithm 3 (SPARSIFY) can be used to provably sparsify a (positive) filter  $w$  of a convolutional layer  $\ell \in [L]$ . In particular, recall that  $w = W_{\dots, r}^\ell \in \mathbb{R}^{\kappa_1^\ell \times \kappa_2^\ell \times c^{\ell-1}}$  denotes an arbitrary filter  $r \in [c^\ell]$ ,  $a(\cdot) \in \mathcal{A}$  denotes the map from a point  $x \sim \mathcal{D}$  to the corresponding kernel patch, and that  $\mathcal{I} = [\kappa_1^\ell] \times [\kappa_2^\ell] \times [c^{\ell-1}]$  denotes the index set of  $w$ . In this context, we show how  $\hat{z}(x) = \sum_{k \in \mathcal{I}} \hat{w}_k a_k(x)$ , where  $\hat{w}$  is the sparse filter produced by SPARSIFY, relates to the true filter output  $z(x) = \sum_{k \in \mathcal{I}} w_k a_k(x)$ .

### C.1 Proof of Lemma 1

**Lemma 1.** *For  $\delta \in (0, 1)$ , invoking RANDOMIZED as in Alg. 3 with  $m \in \mathbb{N}$  satisfying  $m > 8\tilde{S}$ , and a set  $\mathcal{S} \subseteq \mathcal{X}$  composed of  $\lceil K' \log(2\rho^*/\delta) \rceil$  i.i.d. points drawn from  $\mathcal{D}$  generates a filter  $\hat{w}$  such that  $\text{nnz}(\hat{w}) \leq m$  and for an arbitrary patch map  $a(\cdot) \in \mathcal{A}$  and  $x \sim \mathcal{D}$ ,*

$$\mathbb{P}(|\hat{z}(x) - z(x)| \geq \varepsilon_m z(x)) \leq \delta,$$

where  $\hat{z}(x)$  and  $z(x)$  are with respect to patch map  $a(\cdot)$ ,

$$\varepsilon_m = \left( \sqrt{\frac{\tilde{S}}{m} \left( \frac{\tilde{S}}{m} + 6 \right)} + \frac{\tilde{S}}{m} \right) \in (0, 1),$$

$\tilde{S} = SK \log(4/\delta)$ , and  $S = \sum_{j \in \mathcal{I}} s_j$ .

*Proof.* Our proof closely follows the proof of Lemma 1 of Baykal et al. (2018). The sampling procedure of Alg. 1 that samples from the multinomial distribution  $\text{MULTINOMIAL}(q, m)$  is equivalent to sequentially constructing the multiset  $\mathcal{C}$  consisting of  $m$  samples from  $\mathcal{I}$  where each  $j \in \mathcal{I}$  is sampled with probability  $q_j$ . Let  $\mathcal{C} = \{c_1, \dots, c_m\}$  be the subset of weight indices  $\mathcal{I}$  used to construct  $\hat{w}$  as in Alg. 1. Let  $a(\cdot) \in \mathcal{A}$  be arbitrary, let  $\mathbf{x}$  be a realization of  $x \sim \mathcal{D}$ , and let

$$\hat{z} = \sum_{k \in \mathcal{C}} \hat{w}_k a_k(\mathbf{x}) = \sum_{k \in \mathcal{C}} \frac{w_k a_k(\mathbf{x})}{m q_k}$$

be the approximate intermediate value corresponding to the sparsified tensor  $\hat{w}$  and let

$$z = \sum_{k \in \mathcal{I}} w_k a_k(\mathbf{x}).$$

Now define  $\mathcal{E}$  to be the (good) event that  $\hat{z}$   $\varepsilon$ -approximates  $z$ , i.e.,  $\hat{z} \in (1 \pm \varepsilon)z$ . We will now show that the complement of this event,  $\mathcal{E}^c$ , occurs with sufficiently small probability. Let  $\mathcal{Z} \subseteq \text{supp}(\mathcal{D})$  be the set of *well-behaved* points and defined as follows:

$$\mathcal{Z} = \{x' \in \text{supp}(\mathcal{D}) : g_j(x') \leq C s_j \quad \forall j \in \mathcal{I}\}, \quad (2)$$

where  $C = 3K$ . Let  $\mathcal{E}_{\mathcal{Z}}$  denote the event that  $\mathbf{x} \in \mathcal{Z}$

**Conditioned on  $\mathcal{E}_{\mathcal{Z}}$ , event  $\mathcal{E}^c$  occurs with probability  $\leq \frac{\delta}{2}$ :** Let  $\mathbf{x}$  be a realization of  $x \sim \mathcal{D}$  such that  $\mathbf{x} \in \mathcal{Z}$  and let  $\mathcal{C} = \{c_1, \dots, c_m\}$  be  $m$  samples from  $\mathcal{I}$  with respect to distribution  $q$  as before. Define  $m$  random variables  $T_{c_1}, \dots, T_{c_m}$  such that for all  $j \in \mathcal{C}$

$$T_j = \frac{w_j a_j(\mathbf{x})}{m q_j} = \frac{S w_j a_j(\mathbf{x})}{m s_j}. \quad (3)$$

For any  $j \in \mathcal{C}$ , we have for the conditional expectation of  $T_j$ :

$$\mathbb{E}[T_j \mid \mathbf{x}, \mathcal{E}_{\mathcal{Z}}] = \sum_{k \in \mathcal{I}} \frac{w_k a_k(\mathbf{x})}{m q_k} \cdot q_k = \frac{z}{m},$$

where we use the expectation notation  $\mathbb{E}[\cdot]$  with the understanding that it denotes the conditional expectation  $\mathbb{E}_{\mathcal{C}|\mathbf{x},\mathcal{E}_{\mathcal{Z}}}[\cdot]$  and note that conditioning on the event  $\mathcal{E}_{\mathcal{Z}}$  (i.e., the event that  $\mathbf{x} \in \mathcal{Z}$ ) does not affect the expectation of  $T_j$ . Let  $T = \sum_{j \in \mathcal{C}} T_j = \hat{z}$  denote our approximation and note that by linearity of expectation,

$$\mathbb{E}[T] = \sum_{j \in \mathcal{C}} \mathbb{E}[T_j | \mathbf{x}, \mathcal{E}_{\mathcal{Z}}] = z.$$

Thus,  $\hat{z} = T$  is an unbiased estimator of  $z$  for any realization  $\mathbf{a}(\cdot)$  and  $\mathbf{x}$ ; thus, we will henceforth refer to  $\mathbb{E}[T | \mathbf{a}(\cdot), \mathbf{x}]$  as simply  $z$  for brevity.

For the remainder of the proof we will assume that  $z > 0$ , since otherwise,  $z = 0$  if and only if  $T_j = 0$  for all  $j \in \mathcal{C}$  almost surely, in which case the lemma follows trivially.

We now proceed with the case where  $z > 0$  and leverage the fact that  $\mathbf{x} \in \mathcal{Z}$  (by the conditioning on  $\mathcal{E}_{\mathcal{Z}}$ ) to obtain that for all  $j \in \mathcal{I}$ :

$$\begin{aligned} C s_j &\geq g_j(\mathbf{x}) \\ &= \max_{a'(\cdot) \in \mathcal{A}} \frac{w_j a'_j(\mathbf{x})}{\sum_{k \in \mathcal{I}} w_k a'_k(\mathbf{x})} \\ &\geq \frac{w_j a_j(\mathbf{x})}{\sum_{k \in \mathcal{I}} w_k a_k(\mathbf{x})} \\ &= \frac{w_j a_j(\mathbf{x})}{z}. \end{aligned}$$

Rearranging the inequality above, we obtain

$$\frac{w_j a_j(\mathbf{x})}{s_j} \leq C z. \quad (4)$$

Applying the inequality (4), we bound the (conditional) variance of each  $T_j$ ,  $j \in \mathcal{C}$  as follows

$$\begin{aligned} \text{Var}(T_j | \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) &\leq \mathbb{E}[T_j^2 | \mathbf{x}, \mathcal{E}_{\mathcal{Z}}] \\ &= \sum_{k \in \mathcal{I}} \frac{(w_k a_k(\mathbf{x}))^2}{(m q_k)^2} \cdot q_k \\ &\leq \frac{S}{m^2} \left( \sum_{k \in \mathcal{I}} w_k a_k(\mathbf{x}) \right) C z \\ &= \frac{S C z^2}{m^2}. \end{aligned}$$

where  $\text{Var}(\cdot)$  is short-hand for  $\text{Var}_{\mathcal{C}|\mathbf{x},\mathcal{E}_{\mathcal{Z}}}(\cdot)$ . Since  $T$  is a sum of (conditionally) independent random variables, we obtain

$$\text{Var}(T | \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) = m \text{Var}(T_j | \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) \leq \frac{S C z^2}{m}. \quad (5)$$

Now, for each  $j \in \mathcal{C}$  let

$$\tilde{T}_j = T_j - \mathbb{E}[T_j | \mathbf{x}, \mathcal{E}_{\mathcal{Z}}] = T_j - z,$$

and let  $\tilde{T} = \sum_{j \in \mathcal{C}} \tilde{T}_j$ . Note that by the fact that we conditioned on the realization  $\mathbf{x}$  of  $x$  such that  $\mathbf{x} \in \mathcal{Z}$  (event  $\mathcal{E}_{\mathcal{Z}}$ ), we obtain by definition of  $T_j$  in (3) and the inequality (4):

$$T_j = \frac{S w_j a_j(\mathbf{x})}{m s_j} \leq \frac{S C z}{m}. \quad (6)$$

Thus, the inequality established in (6) with the fact that  $S \geq 1$  (by definition) we obtain an upper bound on the absolute value of the centered random variables:

$$|\tilde{T}_j| = \left| T_j - \frac{z}{m} \right| \leq \frac{S C z}{m} = M. \quad (7)$$

Applying Bernstein's inequality to both  $\tilde{T}$  and  $-\tilde{T}$  we have by symmetry and the union bound,

$$\begin{aligned}
 \mathbb{P}(\mathcal{E}^c \mid \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) &= \mathbb{P}(|T - z| \geq \varepsilon z \mid \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) \\
 &\leq 2 \exp\left(-\frac{\varepsilon^2 z^2}{2 \operatorname{Var}(T \mid \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) + \frac{2\varepsilon z M}{3}}\right) \\
 &\leq 2 \exp\left(-\frac{\varepsilon^2 z^2}{\frac{2SCz^2}{m} + \frac{2SC\varepsilon z^2}{3m}}\right) \\
 &= 2 \exp\left(-\frac{\varepsilon^2 m}{SK(6 + 2\varepsilon)}\right) \\
 &\leq \frac{\delta}{2},
 \end{aligned}$$

where the second inequality follows by our upper bounds on  $\operatorname{Var}_{\mathcal{C} \mid \mathbf{x}, \mathcal{E}_{\mathcal{Z}}}(T \mid \mathbf{a}(\cdot), \mathbf{x})$  and  $|\tilde{T}_j|$ , the second by our choice of  $m = \lceil (6 + 2\varepsilon)SK \log(4/\delta)\varepsilon^{-2} \rceil$ , and the last equality by definition of  $C = 3K$ . This establishes that for any realization  $\mathbf{x}$  of  $x$  satisfying  $\mathbf{x} \in \mathcal{Z}$ , the event  $\mathcal{E}^c$  occurs with probability at most  $\frac{\delta}{2}$ .

**Removing the conditioning on  $\mathcal{E}_{\mathcal{Z}}$ :** We have by law of total probability

$$\begin{aligned}
 \mathbb{P}(\mathcal{E}) &\geq \int_{\mathbf{x} \in \mathcal{Z}} \mathbb{P}(\mathcal{E} \mid \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) \mathbb{P}_{x \sim \mathcal{D}}(x = \mathbf{x}) d\mathbf{x} \\
 &\geq \left(1 - \frac{\delta}{2}\right) \int_{\mathbf{x} \in \mathcal{Z}} \mathbb{P}_{x \sim \mathcal{D}}(x = \mathbf{x}) d\mathbf{x} \\
 &= \left(1 - \frac{\delta}{2}\right) \mathbb{P}_{x \sim \mathcal{D}}(\mathcal{E}_{\mathcal{Z}}) \\
 &\geq \left(1 - \frac{\delta}{2}\right) \left(1 - \frac{\delta}{2}\right) \\
 &\geq 1 - \delta,
 \end{aligned}$$

where the second inequality follows from  $\mathbb{P}(\mathcal{E}^c \mid \mathbf{x}, \mathcal{E}_{\mathcal{Z}}) \leq \frac{\delta}{2}$  as was established above, and the third follows by an application of Lemma 9 with  $C = 3K$  of Baykal et al. (2018) and

$$|\mathcal{S}| = \lceil K' \log(2\rho^*/\delta) \rceil,$$

which implies that  $\mathbb{P}_{x \sim \mathcal{D}}(\mathcal{E}_{\mathcal{Z}}) \geq (1 - \frac{\delta}{2})$ . This concludes the proof.  $\square$

## C.2 Proof of Lemma 2

**Lemma 2.** *In the context of Lemma 1, invoking DETERMINISTIC( $\mathcal{I}, w, m, (s_j)_{j \in \mathcal{I}}$ ) generates a filter  $\hat{w}$  such that for an arbitrary patch map  $a(\cdot) \in \mathcal{A}$  and  $x \sim \mathcal{D}$ ,  $\mathbb{P}(|\hat{z}(x) - z(x)| \geq \varepsilon_m z(x)) \leq \frac{\delta}{2}$ , where  $\varepsilon_m = 3K \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_m)} s_j$  and  $\mathcal{I}_m$  is the set of indices of the largest  $m$  values in  $(s_j)_{j \in \mathcal{I}}$ .*

*Proof.* We proceed as in the proof of Lemma 1. Let  $\mathbf{x}$  be a realization of  $x \sim \mathcal{D}$ , let  $\mathcal{Z} \subseteq \operatorname{supp}(\mathcal{D})$  be the set of *well-behaved* points as defined in (2), and let  $\mathcal{E}_{\mathcal{Z}}$  denote the event that  $\mathbf{x} \in \mathcal{Z}$ . We know from the proof of Lemma 1 that if event  $\mathcal{E}_{\mathcal{Z}}$  occurs, then for all indices  $j \in \mathcal{I}$ , the sensitivity inequality of (4) holds, i.e.,

$$\frac{w_j a_j(\mathbf{x})}{s_j} \leq C z \quad \forall j \in \mathcal{I},$$

where  $z = \sum_{k \in \mathcal{I}} w_k a_k(\mathbf{x})$ . Rearranging the inequality above, we obtain

$$w_j a_j(\mathbf{x}) \leq C z s_j \quad \forall j \in \mathcal{I}. \quad (8)$$

Conditioning on the occurrence of event  $\mathcal{E}_Z$ , we bound the relative error as

$$\begin{aligned}
 \left| \sum_{j \in \mathcal{I}} w_j a_j(\mathbf{x}) - \sum_{j \in \mathcal{I}} \hat{w}_j a_j(\mathbf{x}) \right| &= \left| \sum_{j \in \mathcal{I}} w_j a_j(\mathbf{x}) - \sum_{j \in \mathcal{I}_m} w_j a_j(\mathbf{x}) \right| \\
 &= \sum_{j \in \mathcal{I}} w_j a_j(\mathbf{x}) - \sum_{j \in \mathcal{I}_m} w_j a_j(\mathbf{x}) \\
 &= \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_m)} w_j a_j(\mathbf{x}) \\
 &\leq C z \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_m)} s_j \\
 &= \varepsilon_m z
 \end{aligned}$$

where the first equality follows by definition of  $\hat{w}$ , the second by the fact that  $w_j a_j(\mathbf{x}) \geq 0$  for all  $j \in \mathcal{I}$ , the inequality from (8) from above, and last equality by definition of  $C = 3K$  and  $\varepsilon_m$ .

This implies that if  $\mathcal{E}_Z$  occurs, then

$$\sum_{j \in \mathcal{I}} \hat{w}_j a_j(\mathbf{x}) > (1 - \varepsilon_m) \sum_{j \in \mathcal{I}} w_j a_j(\mathbf{x}).$$

Finally, we conclude by using the fact that event  $\mathcal{E}_Z$  occurs with probability at least  $1 - \frac{\delta}{2}$  as established in the proof of Lemma 1.  $\square$

## D Proofs of the Analytical Results in Section B

In the following, we show how we can generalize the results from the previous section to be applicable to all weights of a neural network. Recall that we proceed by (i) relaxing the assumption that weights are positive, (ii) showing how the guarantees for one filter apply to any filter in a layer, and (iii) outlining how we can use the error propagation techniques from Baykal et al. (2017) to extend the results to an entire network.

For sake of simplicity, we proceed in the analysis by solely relying on the sampling-based approach (RANDOMIZED) instead of the hybrid approach (SPARSIFY) as core sparsification procedure. We establish Corollary 5 to obtain the required sampling complexity  $m$  in terms of  $(\varepsilon, \delta)$ , and use the result to generalize the sparsification procedure to negative weights (Lemma 6) and an entire layer (Lemma 7).

### D.1 Proof of Corollary 5

**Corollary 5.** *For any given  $\varepsilon, \delta \in (0, 1)$ , in the context of Lemma 1, invoking RANDOMIZED with sample size*

$$m = \lceil (6 + 2\varepsilon) S K \log(4/\delta) \varepsilon^{-2} \rceil$$

*generates a filter  $\hat{w}$  such that for any arbitrary patch map  $a(\cdot) \in \mathcal{A}$  and  $x \sim \mathcal{D}$ ,*

$$\mathbb{P} \left( \sum_{k \in \mathcal{I}} \hat{w}_k a_k(x) \notin (1 \pm \varepsilon) \sum_{k \in \mathcal{I}} w_k a_k(x) \right) \leq \delta.$$

*Proof.* Rearranging the sampling complexity result of  $m = \lceil (6 + 2\varepsilon) S K \log(\varepsilon_m/\delta) \varepsilon^{-2} \rceil$  from Lemma 1, dropping the ceiling function since  $m$  is of integral value, and solving for  $\varepsilon_m$  yields

$$\varepsilon_m = \sqrt{\frac{\tilde{S}}{m} \left( \frac{\tilde{S}}{m} + 6 \right)} + \frac{\tilde{S}}{m}.$$

We observe from the expression above that in order to ensure that  $\varepsilon_m < 1$ , it suffices to have  $m > 8\tilde{S}$ , and this establishes the result.  $\square$

## D.2 Proof of Lemma 6

**Lemma 6.** *Let  $x \sim \mathcal{D}$ , let  $\Delta_r \geq \tilde{\Delta}_r(x)$  be a constant, and let  $\mathcal{S} \subseteq \mathcal{X}$  be a set of i.i.d.  $\lceil K' \log(4\eta\rho^*/\delta) \rceil$  Consider invoking RANDOMIZED separately with  $(w, \mathcal{I}^+)$  and  $(-w, \mathcal{I}^-)$  each with sample size  $m = \lceil (6 + 2\varepsilon)\Delta_r^2 \left( \sum_{j \in \mathcal{I}} s_j \right) K \log(8\eta/\delta)\varepsilon^{-2} \rceil$  as inputs, where the sum of sensitivities is with respect to  $\mathcal{I}^+$  or  $\mathcal{I}^-$ , to obtain the outputs  $\hat{w}^+$  and  $\hat{w}^-$ . Then, for any arbitrary patch map  $a(\cdot) \in \mathcal{A}$ ,  $\hat{w} = \hat{w}^+ - \hat{w}^-$  satisfies*

$$\mathbb{P} \left( \exists a(\cdot) \in \mathcal{A} : \sum_{k \in \mathcal{I}} \hat{w}_k a_k(x) \notin (1 \pm \varepsilon) \sum_{k \in \mathcal{I}} w_k a_k(x) \right) \leq \delta'$$

where  $\delta' = |\mathcal{A}|^{\delta/\eta}$ ,  $\text{nnz}(\hat{w}) \leq (7 + 2\varepsilon)\Delta_r^2 S_r K \log(8\eta/\delta)\varepsilon^{-2}$ , and  $S_r$  is the sum of sensitivities with respect to filter  $r$ .

*Proof.* Consider an arbitrary patch  $a(\cdot) \in \mathcal{A}$ . Invoking Corollary 5 separately for  $\mathcal{I}^+$  and  $\mathcal{I}^-$ , we obtain filters  $\hat{w}^+$  and  $\hat{w}^-$  satisfying

$$\begin{aligned} \mathbb{P} \left( \hat{z}^+(x) \notin (1 \pm \varepsilon_\Delta) z^+(x) \right) &\leq \frac{\delta}{2\eta}, \quad \text{and} \\ \mathbb{P} \left( \hat{z}^-(x) \notin (1 \pm \varepsilon_\Delta) z^-(x) \right) &\leq \frac{\delta}{2\eta}, \end{aligned}$$

where  $\varepsilon_\Delta = \varepsilon/\Delta_r$  and

$$\hat{z}^+(x) = \sum_{k \in \mathcal{I}_+} w^+ a_k(x) \geq 0 \quad \text{and} \quad \hat{z}^-(x) = \sum_{k \in \mathcal{I}_-} (-\hat{w}^-) a_k(x) \geq 0$$

are the approximated values with respect to  $\hat{w}^+$  and  $\hat{w}^-$ , and

$$z^+(x) = \sum_{k \in \mathcal{I}_+} w^+ a_k(x) \geq 0 \quad \text{and} \quad z^-(x) = \sum_{k \in \mathcal{I}_-} (-w^-) a_k(x) \geq 0.$$

Observe that in light of the above definitions the approximate value is given by  $\hat{z}(x) = \hat{z}^+(x) - \hat{z}^-(x)$  and the ground-truth value is given by  $z(x) = z^+(x) - z^-(x)$ .

By the triangle inequality and by union bounding over the failure events above, we obtain that with probability at least  $1 - \delta/\eta$ ,

$$\begin{aligned} |\hat{z}(x) - z(x)| &\leq |\hat{z}^+(x) - z^+(x)| + |\hat{z}^-(x) - z^-(x)| \\ &\leq \varepsilon_\Delta z^+(x) + \varepsilon_\Delta z^-(x) \\ &= \varepsilon_\Delta (z^+(x) + z^-(x)) \\ &\leq \varepsilon, \end{aligned}$$

where the last inequality follows by definition of  $\Delta_r$ . Thus, we have for the failure event,

$$\mathbb{P} \left( \hat{z}(x) \notin (1 \pm \varepsilon_\Delta) z(x) \right) \leq \frac{\delta}{\eta}.$$

Since the choice of the patch  $a(\cdot) \in \mathcal{A}$  was arbitrary, the bound above holds for any  $a(\cdot) \in \mathcal{A}$ . Thus, applying the union bound over all  $|\mathcal{A}|$  patches establishes the lemma.  $\square$

## D.3 Proof of Lemma 7

**Lemma 7.** *For any  $\varepsilon, \delta \in (0, 1)$  and layer  $\ell \in [L]$ , the approximate output  $\hat{Z}^\ell(x) \in \mathbb{R}^{s_1^\ell \times s_2^\ell \times c^\ell}$  with respect to tensor  $\hat{W}^\ell$  satisfies  $\mathbb{P} \left( \hat{Z}^\ell(x) \notin (1 \pm \varepsilon) Z^\ell(x) \right) \leq \frac{\delta \eta^\ell}{\eta}$ , where  $\text{nnz}(\hat{W}^\ell) \leq \sum_{r \in [c^\ell]} (7 + 2\varepsilon)\Delta_r^2 S_r K \log(8\eta/\delta)\varepsilon^{-2}$ .*

*Proof.* Recalling that  $\eta^\ell = |\mathcal{A}| \cdot c^\ell$ , invoking Lemma 6 for each filter  $r \in [c^\ell]$ , and union bounding over  $c^\ell$  failure events establishes the result.  $\square$

## E Experimental setup

**Training hyper-parameters** The LeNet networks were trained on the MNIST data set for 40 epochs with a batch size of 64 and weight decay of 0.0001. The initial learning rate was set to be 0.01 and decayed to 0.001 after 30 epochs. We trained Resnet20 and WRN16 on the CIFAR10 data set for 182 epochs and 200 epochs, respectively, with a batch size of 128. We used the conventional data augmentation techniques of random cropping and flipping, and padding pixels on each side. For Resnet20 and WRN16 the initial learning rate was set to 0.1. For resnet20, the rate was decayed to 0.01 after 91 epochs and then to 0.001 after 136 epochs from the start; for WRN16, the learning rate was scaled down by multiplying by it 0.2 every 60 epochs.

**Iterative prune + retrain** For each trained network with  $N$  parameters, we consider a harmonic progression of sparsity ratios: in the first iteration ( $i = 1$ ), we first prune the network so that only  $N/2$  of the  $N$  original parameters remain, and then fine-tune by retraining the network with the same hyper-parameters and epochs as the original training procedure. In the next iteration, we further prune the resulting fine-tuned network so that it contains  $N/3$  parameters, then fine-tune the network, and repeat the process iteratively. More generally, at iteration  $i$ , our sparsity ratio (with respect to  $N$ ) is  $1/(i+1)^p$  with  $p = 1$ . This progression by design leads to an aggressive pruning scheme in the initial phases – going from 100% to 33% in two iterations –, and then to a much more gradual and cautious pruning procedure as the problem becomes harder – e.g., it takes roughly 100 iterations to reach 1% sparsity. We adjust the exponent  $p \in (0, 1)$  depending on the ease of compressibility of the network.

## F Compared Methods

To empirically evaluate the performance of SiPP, we compare against various state-of-the-art algorithm in network pruning as well as matrix sparsification. In particular, we consider the following algorithms as listed below.

**Singular Value Decomposition (SVD)** Each weight matrix  $W$  is approximated by its low-rank counterpart  $\hat{W}$  of rank  $r$  by only maintaining the  $r$  largest singular values of its singular value decomposition  $W = U\Sigma V^T$ . The size of the resulting sparsification is computed based on the  $r$  left- and right-singular vectors that are kept. For the case of high-dimensional ( $> 2$ ) weight tensors, we first reshape the tensor into its two-dimensional matrix counterpart and then use standard SVD for matrices.

**Norm-based Sampling ( $\ell_1 + \ell_2/2$  or "Norm")** A provable sampling-based procedure for matrix sparsification is given in (Kundu and Drineas, 2014) by defining an importance sampling distribution over each entry  $(i, j)$  with probability

$$p_{ij} = \frac{1}{2} \left( \frac{w_{ij}^2}{\|W\|_F^2} + \frac{|w_{ij}|}{\|W\|_{\ell_1}} \right).$$

In case of multi-dimensional weight tensors, we again define the probability distribution with respect to its matrix counterpart. We note that the matrix sparsification introduced by Kundu and Drineas (2014) currently constitutes one of the state-of-the-art algorithms in data-oblivious matrix sparsification.

**Weight Thresholding (WT) (Han et al., 2015)** Consider the parameter tuple  $\theta = (W^1, \dots, W^L)$  of a neural network and let  $w_t$  denote the threshold value such that the absolute value of  $t\%$  of the parameters of  $\theta$  fall below the threshold. A weight tensor is then approximated by only keeping weight entries which absolute values are above  $w_t$ . The remaining entries are zeroed out. While there is no analysis on the theoretical performance of weight thresholding, empirical performance suggests that weight thresholding constitutes the current state-of-the-art approach in network pruning and consistently performs on par with other heuristics for network pruning.

We note that we also investigated the performance of uniform sampling as well as other provable sampling approaches to matrix sparsification (Achlioptas et al., 2013, Drineas and Zouzias, 2011). Empirically, we found that the approach presented in (Kundu and Drineas, 2014) is the most competitive comparison to SiPP. For brevity of exposition, we thus omitted the comparison to other sampling-based approaches in the presentation of the results.