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

Cenk Baykal[†*], Lucas Liebenwein[†*], Igor Gilitschenski[†], Dan Feldman[‡], Daniela Rus[†]

Abstract

We introduce a family of pruning algorithms that provably sparsifies the parameters of a trained model in a way that approximately preserves the model's predictive accuracy. Our algorithms use a small batch of input points to construct a data-informed importance sampling distribution over the network's parameters, and either use a sampling-based or deterministic pruning procedure, or an adaptive mixture thereof, to discard redundant weights. Our pruning methods are simultaneously computationally efficient, provably accurate, and broadly applicable to various network architectures and data distributions. The presented approaches are simple to implement and can be easily integrated into standard prune-retrain pipelines. We present empirical comparisons showing that our algorithms reliably generate highly compressed networks that incur minimal loss in performance relative to that of the original network.

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 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. Network pruning can also be used to reduce the burden of manually designing a small network by automatically inferring efficient architectures from larger networks. Moreover, pruning algorithm can enable novel insights into the theoretical and practical properties of neural networks, including overparameterization and generalization [5, 28].

Existing network pruning algorithms are predominantly based on data-oblivious [38, 17] or data-informed [16, 27, 31, 50, 26] heuristics that work well in practice in combination with an appropriate pruning pipeline that incorporates retraining. However, existing approaches generally lack provable guarantees (including data-informed approaches with the exception of [6] which is only applicable to multi-layer perceptrons) and thus provide little insight into the mechanics of the pruning algorithms and consequently into the pruned network.

We close this research gap by introducing SiPP, see Figure 1 for an overview, a family of network pruning algorithms that provably compresses the network's parameters in a data-informed manner.

[†]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, emails: {baykal, lucasl, igilitschenski, rus}@mit.edu

[‡]Robotics and Big Data Laboratory, University of Haifa, email: dannyf.post@gmail.com

[*]These authors contributed equally to this work

Figure 1: The overview of our randomized 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 with fewer edges.

Building and improving on state-of-the-art pruning methods, our algorithm is simultaneously provably accurate, data-informed, and applicable to various architectures including fully-connected (FNNs), convolutional (CNNs), and recurrent neural networks (RNNs). Relative to existing approaches, SiPP exhibits provable guarantees that hold regardless of the specific state of the network, i.e., it is simultaneously applicable to untrained, trained, or partially trained networks, and hence tends to perform consistently well across diverse pruning pipelines that incorporate various amounts of retraining. In addition, the theoretical analysis of SiPP provides novel analytical compression bound for deep neural networks that, e.g., can be utilized in the context of generalization bounds [6, 5, 2, 4, 35, 54].

This paper contributes the following:

1. A provable and versatile family of pruning algorithms, SiPP, that combines novel sample size allocation and adaptive sparsification procedures to prune network parameters.
2. An analysis of the resulting size and accuracy of the compressed network generated by SiPP that establishes novel compression bounds for a large class of neural networks.
3. Empirical evaluations for state-of-the-art iterative prune + retrain, random-init + prune + train scenarios on fully-connected and convolutional with comparisons to baseline pruning approaches highlighting the ability of SiPP to span both theory and practice.

2   Related Work

Traditional approaches.   Techniques such as Singular Value Decomposition (SVD) and regularized training [11, 12, 22, 23, 45, 21, 3, 51] were traditionally applied to compress networks. Other approaches in this realm exploit the structure of weight tensors to induce sparsity [53, 42, 9, 10, 48]. 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 [25] 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 [24, 13], e.g., by embedding sparsity as a constraint [20, 1, 29]. A popular weight-based pruning method is that of [17, 38], where weights with absolute values below a threshold are removed. A recent approach of [26] 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. Other data-informed techniques include [16, 30, 27, 32, 31, 50, 28]. For an extensive overview see [15, 7, 49]. Despite their favorable empirical performance, these approaches generally lack rigorous theoretical analysis of the effect that the discarded weights can have on the model's performance.

Theoretical foundations. Recently, [5] introduced a compression method based on random projections and proved norm-based bounds on the compressed network for points in the *training set only*. In contrast, our work provides approximation guarantees on the network's output that hold even for points outside the training set. A coresets-based [14, 8] approach for compressing fully-connected networks was introduced by [6] but is limited to FNNs and ReLUs. Our approach builds on this coresets-based framework to be applicable to various architectures and activation functions. Our algorithm also exhibits stronger error guarantees by mixing deterministic and sampling-based pruning strategies, by optimally allocating the sample sizes across the network to minimize the approximation error, and by establishing stronger network compression bounds using novel error propagation techniques.

3   Background

We consider a neural network $f_\theta : \mathcal{X} \to \mathcal{Y}$ consisting of L layers with parameters θ and distribution $\mathcal{D}$ over the input space $\mathcal{X}$ from which we can sample i.i.d. input/label pairs (x, y).

Network notation. For a given input $x \sim \mathcal{D}$, we denote the pre-activation and activation of layer ℓ by $Z^\ell(x)$ and $A^\ell(x)$, respectively. Note that $f_\theta(x) = A^L(x)$, $Z^0(x) = x$, and $A^\ell(x) = \phi^\ell(Z^\ell(x))$, where $\phi^\ell(\cdot)$ denotes the activation function. We consider any multi-dimensional layer that can be described by a linear map with *parameter sharing*, e.g. fully-connected layers, convolutional layers, or LSTM cells. Specifically, for a layer ℓ the pre-activation $Z^\ell(x)$ of layer ℓ is described by the linear mapping of the activation $A^{\ell-1}(x)$ with W^ℓ, i.e., $Z^\ell(x) = W^\ell * A^{\ell-1}(x)$, where $*$ denotes the operator of the linear map, e.g., the convolutional operator. Moreover, we denote by c^ℓ the number of parameter groups within a layer that do not interact with each other, e.g., individual filters in convolutional layers. Then, let $Z_i^\ell(x) = W_i^\ell * A^{\ell-1}(x)$, $i \in [c^\ell]$, denote the ith pre-activation channel of layer ℓ produced by parameter group W_i^ℓ.

Problem definition. For given $\varepsilon, \delta \in (0, 1)$, our overarching goal is to use a pruning algorithm to generate a sparse reparameterization $\hat{\theta}$ of θ such that $\|\hat{\theta}\|_0 \ll \|\theta\|_0$ and for $x \sim \mathcal{D}$ the ℓ_2-norm of reference network output $f_\theta(x)$ can be approximated by $f_{\hat{\theta}}(x)$ up to $1 \pm \varepsilon$ multiplicative error with probability greater than $1 - \delta$, i.e., $\mathbb{P}(\|f_{\hat{\theta}}(x) - f_\theta(x)\| \le \varepsilon \|f_\theta(x)\|) \ge 1 - \delta$.

Algorithm 1 SiPP $(\theta, \mathcal{B}, \mathcal{S})$

Input: $\theta = (W^1, \ldots, W^L)$: weights of the uncompressed neural network; $\mathcal{B} \in \mathbb{N}$: sampling budget; $\mathcal{S}$: a set of n i.i.d. validation points drawn from $\mathcal{D}$
Output: sparse weights $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$

1: $m_i^\ell \leftarrow$ OPTALLOC$(\theta, \mathcal{B}, \mathcal{S})$ $\forall i \in [c^\ell], \forall \ell \in [L]$ $\triangleright$ optimally allocate budget $\mathcal{B}$ across parameter groups and layers
2: **for** $\ell \in [L]$ **do**
3:     $\hat{W}^\ell \leftarrow \mathbf{0}$; $\triangleright$ Initialize a null tensor
4:     **for** $i \in [c^\ell]$ **do**
5:        $s_j \leftarrow$ EMPIRICALSENSITIVITY$(\theta, \mathcal{S}, i, \ell)$ $\forall w_j \in W_i^\ell$ $\triangleright$ Compute parameter importance for each weight w_j in the parameter group
6:        $\hat{W}_i^\ell \leftarrow$ SPARSIFY$(W_i^\ell, m_i^\ell, \{s_j\}_j)$ $\triangleright$ prune weights according to SiPPDET, SiPPRAND, or SiP-PHYBRID such that only m_i^ℓ weights remain in the parameter group
7:     **end for**
8: **end for**
9: **return** $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$;

---

4 Method

In this section, we present an overview for our family of pruning algorithms, SiPP: Sensitivity-informed Provable Pruning (see Figure 1 and Algorithm 1). In its core, SiPP proceeds as follows: (1) optimally allocate a given budget across layers and parameter groups to minimize the theoretical error bounds resulting from our analysis (OPTALLOC, Line 1); (2) compute the relative importance of individual weights within parameter groups (EMPIRICALSENSITIVITY, Line 5); (3) prune weights within each parameter group using the desired variant of SiPP according to their relative importance (SPARSIFY, Line 6); (4) repeat (2) and (3) for each parameter group and each layer.

OPTALLOC$(\theta, \mathcal{B}, \mathcal{S})$ In the course of our analysis (see Section 5) we establish relative error bounds for the approximation $\hat{Z}_i^\ell(x) = \hat{W}_i^\ell * A^{\ell-1}(x)$ of the form $\hat{Z}_i^\ell(x) \in \left(1 \pm \varepsilon_i^\ell(m_i^\ell)\right) Z_i^\ell(x)$ for individual parameter groups. Roughly speaking, the associated relative error $\varepsilon_i^\ell(m_i^\ell)$ is a (convex) function of the parameter group, the input, and the allocated budget m_i^ℓ. Thus in order to optimally utilize a desired budget $\mathcal{B}$ we aim to minimize the following objective during the allocation procedure:

$$\min_{m_i^\ell \in \mathbb{N}\, \forall i \in [c^\ell], \, \forall \ell \in [L]} \quad \sum_{\ell \in [L], \, i \in [c^\ell]} \varepsilon_i^\ell(m_i^\ell) \quad \text{s. t.} \quad \sum_{\ell \in [L], \, i \in [c^\ell]} m_i^\ell \leq \mathcal{B}.$$

We note that the integral constraint $m_i^\ell \in \mathbb{N}$ prevents us from efficiently finding a solution, we relax it to $m_i^\ell \in \mathbb{R}$ to find the optimal fractional solution. We then use a technique like randomized rounding [43] to find an approximately optimal integral solution. Depending on the variant of SiPP, however, this step is not necessary.

EMPIRICALSENSITIVITY$(\theta, \mathcal{S}, i, \ell)$ To estimate the relative importance of a weight w_j within a parameter group W_i^ℓ, we use and extend the notion of *empirical sensitivity* (ES) as first introduced in [6] for fully-connected layers only. In its essence, ES quantifies the maximum relative contribution

of a weight parameter w_j to the output (pre-activation) of the layer compared to other weights in the parameter group. More formally, let the ES s_j of w_j in parameter group W_i^ℓ be defined as

$$s_j := \max_{x \in \mathcal{S}} \max_{a(\cdot) \in \mathcal{A}} w_j a_j(x) / {\textstyle\sum_k} w_k a_k(x), \tag{1}$$

where we assume $W_i^\ell \geq 0$, $A^{\ell-1}(x) \geq 0$ for ease of exposition (see Section 5 for the generalization to all weights and activations). We note that the definition of s_j entails two maxima. The maximum over data points $\max_{x \in \mathcal{S}}$ ensures that ES approximates the relative importance of w_j sufficiently well for any i.i.d. data point $x \sim \mathcal{D}$. The maximum over patches $\mathcal{A}$, which are generated from $A^{\ell-1}(x)$, ensures that ES approximates the relative importance of w_j sufficiently well for all scalars in the output $Z_i^\ell(x)$ that require w_j (c.f. parameter sharing). To further contextualize the purpose of patches $\mathcal{A}$, consider a single parameter group within a convolutional layer, i.e., a filter. The filter gets slid across the input image to generate the output image repeatedly applying the same weights. Thus in order to quantify the importance of some weight w_j we need to consider its relative importance across all sliding windows, henceforth requiring $\max_{a(\cdot) \in \mathcal{A}}$.

SPARSIFY$(W_i^\ell, m_i^\ell, \{s_j\}_j)$ Equipped with a budget, c.f. OPTALLOC, and a notion of parameter importance, c.f. EMPIRICALSENSITIVITY, we introduce the three variants of SIPP, all of which exhibit provable guarantees as outlined in Section 5, to prune weights from a parameter group:
  1. SIPPDET: we deterministically pick the m_i^ℓ weights with largest sensitivity and zero out the rest of the weights to construct $\hat{W}_i^\ell$.
  2. SIPPRAND: we construct an importance sampling distribution over weights w_j using their associated sensitivities s_j, then sample with replacement until we obtain a set of m_i^ℓ unique weights to construct $\hat{W}_i^\ell$.
  3. SIPPHYBRID: we evaluate the theoretical error guarantees (see Section 5) associated with the two other methods, and prune using the method that incurs the lower relative error.

We note that while SIPPDET is particularly simple to implement, SIPPHYBRID provides the biggest amount of flexibility and consistently good prune results since it can adaptively choose for each parameter group whether to prune using SIPPDET or SIPPRAND.

5 Analysis

In this section, we outline the theoretical guarantees for SIPP. The full proofs can be found in the supplementary material. We start out by establishing the core lemmas that constitute the relative error guarantees for both SIPPDET and SIPPRAND for the case where $W_i^\ell \geq 0$, $A^{\ell-1}(x) \geq 0$ for ease of exposition. Specifically, we establish relative error guarantees for each individual output patch that is associated with a parameter group. We then outline the steps that are required to generalize the analysis to all weights and activations. Finally, we show – by means of composing together the error guarantees from individual output patches, parameters groups, and layers – how to derive the analytical compression bounds for the entire network.

Empirical sensitivity

In the previous section we introduce the notion of ES, see equation 1, as a means to quantify the importance of weight w_j relative to the other weights within a parameter group W_i^ℓ. Using ES we establish a key inequality that upper bounds the contribution of $w_j a_j(x)$ to its associated output

patch $z(x) = \sum_k w_k a_k(x)$ for any $x \sim \mathcal{D}$ with high probability (w.h.p.) under mild regularity assumption on the input distribution to the layer.

Lemma 1 (Informal ES inequality). *For weights w_j from parameter group W_i^ℓ and an arbitrary input patch $a(\cdot)$ we have w.h.p. for any $x \sim \mathcal{D}$ that $w_j a_j(x) \leq C s_j z(x)$, where $z(x)$ denotes the associated output patch and $C = \mathcal{O}(1)$.*

The ES inequality is a key ingredient in bounding the error of SiPPDet and SiPPRand in terms of sensitivity. Specifically, Lemma 1 puts the individual contribution of a weight to the output patch in terms of its sensitivity and the output patch itself. The inequality hereby holds *w.h.p.* for any data point $x \sim \mathcal{D}$ which enables us to bound the quality of the approximation even for previously unseen data points. We leverage Lemma 1 in the subsequent analysis to quantify the approximation error of an output patch when the output patch was only approximately computed using a subset of weights, i.e., with the weights that remain after pruning.

Error guarantees for SiPPDet

Recall that SiPPDet prunes weights by keeping the m_i^ℓ weights of parameter group W_i^ℓ with largest ES. Now let $\mathcal{I}$ denote the index set of all weights in W_i^ℓ and $\mathcal{I}_{det}$ the index set of weights with largest sensitivity that are kept after pruning such that $|\mathcal{I}_{det}| = m_i^\ell$. We bound the incurred error of the approximation by considering the difference between the output patch and the approximated output patch, i.e., the difference between $z(x) = \sum_{j \in \mathcal{I}} w_j a_j(x)$ and $\hat{z}_{det}(x) = \sum_{j \in \mathcal{I}_{det}} w_j a_j(x)$.

Lemma 2 (Informal SiPPDet error bound). *For weights w_j from parameter group W_i^ℓ, an arbitrary associated input patch $a(\cdot) \in \mathcal{A}$, and corresponding output patch $z(\cdot)$ SiPPDet generates an index set $\mathcal{I}_{det}$ of pruned weights such that for any $x \sim \mathcal{D}$ w.h.p. $|\hat{z}_{det}(x) - z(x)| \leq \varepsilon_{det} z(x)$, where $\varepsilon_{det} = C \sum_{j \in \mathcal{I} \setminus \mathcal{I}_{det}} s_j$.*

The proof of Lemma 2 follows from the fact that the difference between the approximate output patch $\hat{z}_{det}(x)$ and the unpruned output patch $z(x)$ is exactly the sum over the contributions from weights that are *not* in the pruned subset of weights $\mathcal{I}_{det}$. Using Lemma 1 we then bound the error in terms of the sensitivity of the *pruned* weights. Intuitively, ES of an individual weight precisely quantifies the relative error incurred when that weight is pruned. The resulting relative error can thus be described by the cumulative ES of pruned weights.

Error guarantees for SiPPRand

Here we prune weights from a parameter group by constructing an importance sampling distribution from the associated ESs. Specifically, some weight w_j is sampled with probability $q_j = s_j / \sum_{k \in \mathcal{I}} s_k$ and we repeatedly sample with replacement until the corresponding set of sampled weights contains m_i^ℓ unique weights. Each sampled weight is then reweighed by the number of times it was sampled divided by the total number of samples and its sample probability to construct the approximate output patch, i.e.,

$$\hat{z}_{rand}(x) = \sum_{j \in \mathcal{I}_{rand}} \hat{w}_j a_j(x) = \sum_{j \in \mathcal{I}_{rand}} \frac{n_j}{N q_j} w_j a_j(x),$$

where $\mathcal{I}_{rand}$ denotes the index set of weights that were sampled at least once, n_j denotes the number of times weight w_j was sampled, and $N = \sum_{j \in \mathcal{I}_{rand}} n_j$ denotes the total number of samples. We then bound the incurred error by analyzing the random difference between the approximated output patch and the original output patch, i.e., $|\hat{z}_{rand}(x) - z(x)|$, establishing the following error guarantee.

6

Lemma 3 (Informal SiPPRand error bound). *For weights w_j from parameter group W_i^ℓ, an arbitrary associated input patch $a(\cdot) \in \mathcal{A}$, and corresponding output patch $z(\cdot)$ SiPPRand generates a set of pruned weights such that for any $x \sim \mathcal{D}$ w.h.p. $|\hat{z}_{rand}(x) - z(x)| \le \varepsilon_{rand} z(x)$, where $\varepsilon_{rand} = \mathcal{O}(\sqrt{S/N})$ and $S = \sum_{k \in \mathcal{I}} s_k$ denote the relative error and sum of ESs, respectively.*

The proof proceeds in two steps. First, we show that the (random) approximation is an unbiased estimator of the original parameter group, i.e., $\mathbb{E}[\hat{z}_{rand}(x)] = z(x)$, which follows from the reweighing term of $\hat{w}_j$. Second, we show that using Bernstein's concentration inequality [47] the sampling distribution exhibits strong subGaussian [47] concentration around the mean, i.e., the approximate output patch is ε-close to the original, unpruned output patch w.h.p. Specifically, we leverage Lemma 1 to bound the variance of the approximate output patch using the cumulative ES $S = \sum_{k \in \mathcal{I}} s_k$ of the parameter group.

Discussion of error bounds and SiPPHybrid

Most notably, SiPPRand is an unbiased estimator regardless of the budget, while SiPPDet is always an underapproximation becoming increasingly worse in expectation with lower budget. On the other hand, if the parameter group is dominated by a few weights, SiPPDet can directly captures these weights whereas SiPPRand inherent randomness from the sampling procedure may introduce additional sources of failure. Combining the strengths of both, we introduce SiPPHybrid, which evaluates both theoretical error guarantees before pruning a parameter group to adaptively choose the better prune strategy.

Generalization to all weights

Previously, we have assumed that both the parameter group and input activations are strictly non-negative, i.e., $W_i^\ell \ge 0$ and $A^{\ell-1}(x) \ge 0$. To handle the general case, we split the parameter group and input activations each into a positive and negative part representing the four quadrants such that each quadrant is now strictly non-negative. We can then incorporate each quadrant into our pruning procedure to ensure that the error guarantees hold simultaneously for all quadrants. To obtain error bounds for the actual pre-activation we introduce Δ^ℓ, which quantifies the "sign complexity" of the overall approximation for a particular layer to quantify the additional complexity from considering the alternating signs of each quadrant, see supplementary material for more details.

Network compression bounds

In the previous section we have outlined how to obtain error guarantees for individual output patches. Naturally, since the guarantees hold for all patches within a parameter group and individual parameter groups within a layer are independent from each other, we can simultaneously establish norm-based error guarantees for the entire pre-activation of a layer, i.e., $\left\| \hat{Z}^\ell(x) - Z^\ell(x) \right\| \le \varepsilon \left\| Z^\ell(x) \right\|$ w.h.p. Moreover, assuming the activation function is entry-wise and 1-Lipschitz continuous, the same relative error guarantees hold for the activation of layer. Note that any common activation function satisfies the above assumption, including and all others listed in PyTorch's documentation [36]. Finally, we have to consider the effect of pruning *multiple layers* simultaneously and the implications on the final output $f_\theta(x) = A^L(x)$ of the network. Informally speaking, we incur two sources of error from each layer. (1) the error associated from pruning within layers and (2) the error associated with propagated the incurred error throughout the network to the output layer. We quantify the

error within layers using our patch-wise guarantees and the sign complexity Δ^ℓ of the layers. We quantify the propagated error across layers by upper bounding the layer condition number, κ^ℓ which quantifies the relative error incurred in the output for some relative error incurred within the layer. Intuitively, the concept of the layer condition number is closely related to the Lipschitz constant between some layer and the output of the network. Below, we informally state the compression bound when pruning the entire network with SiPPRand.

Theorem 4 (Informal compression bound). *For given $\delta \in (0,1)$ and budget $\mathcal{B}$ SiPP (Algorithm 1) generates a set of pruned parameters $\hat{\theta}$ such that $\|\hat{\theta}\|_0 \leq \mathcal{B}$, $\mathbb{P}_{\hat{\theta},x} \left(\left\| f_{\hat{\theta}}(x) - f_\theta(x) \right\| \leq \varepsilon \left\| f_\theta(x) \right\| \right) \geq 1 - \delta$ and $\varepsilon = \mathcal{O}(\sum_{\ell=1}^{L} \kappa^\ell \Delta^\ell \max_{i \in [c^\ell]} (S_i^\ell - S_i^\ell(N_i^\ell)))$, where S_i^ℓ and $S_i^\ell(N_i^\ell)$ is the sum over all and the largest N_i^ℓ ESs, respectively, and N_i^ℓ is the budget allocated for parameter group W_i^ℓ.*

We note that the compression bound is proportional to the sum of cumulative ESs for each parameter group, a term which arises in numerous applications of coresets [14]. Moreover, we see the layer condition number κ^ℓ and sign complexity Δ^ℓ of each layer appear in the final bound. Both terms are related to how injecting error simultaneously in each layer (by pruning the network) affects the overall output of the network and are related to concepts such as the Lipschitz constant of the network and/or interlayer cushion as introduced in related work that establishes generalization bounds for neural networks [5, 34]. Like other recent work in the field [5, 44] our work highlights the intrinsic connection between the compression ability and generalization ability of neural networks.

6   Experiments

In this section, we evaluate and compare the performance of our algorithm, SiPP, on pruning fully-connected, convolutional, and residual networks. We embed our pruning algorithm into pruning pipelines including retraining to empirically test its performance and test it for scenarios involving significant amounts of (re)-training as well as a prune pipeline that utilizes no more training epochs than regular training. To be able to compare our pruning approach SiPP to competing pruning approaches, we consider standard retraining pipelines that are network-agnostic and yield state-of-the-art pruning results [26, 38]. Specifically, we consider two scenarios – iterative prune + retrain and random-init + prune + train – as described below.

6.1   Experimental Setup

Architectures and data sets.   We train and prune networks on CIFAR10 [46] and ImageNet [39]. We consider ResNets20/56/110 [18], WideResnet16-8 [52], Densenet22 [19], VGG16 [40], CNN5 [33] and ResNet18, ResNet101 [18] for CIFAR10 and ImageNet, respectively.

Training.   For both training and retraining we deploy the standard sets of hyperparameters as described in the respective papers. All hyperparameters are listed in the supplementary material.

Pruning algorithms.   We consider the following pruning algorithms to be incorporated into the pruning pipelines discussed above:
- SiPPDet. We prune the entire network deterministically. Note that in this case (due to the sample size allocation procedure) SiPPDet corresponds to global thresholding of sensitivity (reminiscent of weight thresholding).

- SᴉPPRᴀɴᴅ. We prune the entire network using importance sampling.
- SᴉPPHʏʙʀɪᴅ. We use our combined pruning approach as outlined in Algorithm 1.
- WT. We globally prune weights according to their magnitude [17, 38].
- Sɴɪᴘ. We globally prune weights according to the (data-informed) magnitude of the product between weight and gradient [26].

We note that WT ("learning rate rewinding") is the current state-of-the-art for iterative prune+retrain pipelines [38] while Sɴɪᴘ is the current state-of-the-art for random-init + prune + train [26]. We also report comparisons against a broader set of pruning pipelines in the supplementary material.

6.2 Iterative prune + retrain

Methodology   We deploy an iterative prune + retrain scheme that proceeds as follows:

1. train network to completion;
2. prune a fixed ratio of parameters from the network;
3. retrain using the same hyperparameters as during training;
4. iteratively repeat steps 2., 3. to obtain smaller prune ratios.

This procedure as used in [38, 28] is shown to produce state-of-the-art prune results although it requires significant amount of retraining resources. We choose it for its simplicity and network-agnostic hyperparameters. Due to the expensive nature of iterative prune+retrain we choose to only evaluate it for SᴉPP but not the other variants of our algorithm as it is the

Figure 2: Results on the pruning performance of a ResNet18 trained on ImageNet using **iterative prune+retrain**.

simplest and we observed little difference in performance between the three variations. In the supplementary material, we provide additional (experimental) justification that supports our claim.

Results.   Figure 3 summarizes the results of the iterative prune + retrain procedure for various CIFAR10 networks. The results were averaged across 3 trained networks. Our empirical evaluation shows that our algorithm consistently performs comparably to state-of-the-art WT with learning rate rewinding [38]. We note that Sɴɪᴘ's performance is much lower is these scenarios. We suspect this is due to the gradients being close to zero for a fully-trained network (the pruning step is performed after training in this scenario). In Figure 2 we show results for a ResNet18 trained, pruned, and retrained on ImageNet. As in the case of CIFAR10 networks we observe that SᴉPP performs en par with WT.

6.3 Random-init + prune + train

Methodology.   On the other "extreme" of possible pruning pipeline, we consider the following scenario as described in [26]:

1. randomly initialize the network;

2. prune the network to the desired prune ratio;
3. train the network using the regular hyperparameters.

While (due to the limited amount of training) this pipeline does not achieve as high prune ratios as the above scenario, it is simple and requires much less training epochs overall. It also serves as a useful experimental platform to understand if pruning methods are able to unearth important connections inherent in the network.

Results. In Figure 4 the prune results for various CIFAR10 networks are shown. We note that for low prune ratios all pruning methods perform uniformly well, which most likely can be attributed to the overall overparameterization of the tested networks. For higher prune ratios, we observe vastly different performance. Specifically, WT's performance drops to 10% test accuracy (uniformly at random for CIFAR10) for prune ratios beyond 90%. We suspect that weights do not contain sufficient information about the importance of the connection before training and thus WT fails. On the other hand, SNIP performs consistently well due to the consideration of data and the gradients of weights. We note that SIPPHYBRID specifically, which adaptively mixes SIPP and SIPPRAND according to the theoretical bounds, performs well across all tested networks and achieves the same prune performance as SNIP. For deeper networks (ResNet20 and ResNet56) in particular, we observe all SIPP variations performing well or even outperforming SNIP.

6.4 Discussion

For our experiments we have embedded SIPP into two maximally diverse pruning pipelines in terms of the amount of (re-)training epochs, which constitutes the majority of computational cost in pruning. By doing so, we highlight the versatility and robustness of SIPP in performing well across many different tasks. While traditional pruning methods, such as WT and SNIP (for further comparisons, see the supplementary material), perform inconsistently when used in the context of alternative

Figure 3: The delta in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR10 for various target prune ratios. The networks were pruned using the **iterative prune+retrain** pipeline.

Figure 4: The test accuracy for the generated pruned models trained on CIFAR10 for various target prune ratios using the **random-init + prune + train** pipeline.

pruning pipelines we observe that SiPP serves as a consistent plug-and-play solution to the core pruning method of a pruning pipeline. Among the SiPP variants, we see that SiPPDet tends to perform particularly well for small prune ratios (such as in the case of iterative prune+retrain) while SiPPRand performs the best for extreme prune ratio (such as in the case of random-init + prune + train). SiPPHybrid usually finds a close-to-optimal mixture of strategies and thus provides the most versatility among the SiPP variants, which comes at the cost of increased implementation effort.

7 Conclusion

In this work, we presented a simultaneously provably and practical family of network pruning methods, SiPP, that is grounded in a data-informed measure of sensitivity. Our analysis establishes provable guarantees that quantify the trade-off between the desired model sparsity and resulting accuracy of the pruned model establishing novel analytical compression bounds for a large class of neural networks. SiPP's versatility in providing strong prune results across a variety of tasks 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. We envision that SiPP can spur further research into network pruning by providing a robust core pruning method that can be reliably integrated into any pruning pipelines with close-to-optimal prune performance.

Acknowledgments

References

[1] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, pages 3180–3189, 2017.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems 32*, pages 6158–6169. Curran Associates, Inc., 2019.

[3] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

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

[5] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

[6] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.

[7] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020*, pages 129–146, 2020.

[8] Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint arXiv:1612.00889*, 2016.

[9] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

[10] Anna Choromanska, Krzysztof Choromanski, Mariusz Bojarski, Tony Jebara, Sanjiv Kumar, and Yann LeCun. Binary embeddings with structured hashed projections. In *International Conference on Machine Learning*, pages 344–353, 2016.

[11] Misha Denil, Babak Shakibi, Laurent Dinh, Marc Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156, 2013.

[12] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR*, abs/1404.0736, 2014.

[13] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017.

[14] Dan Feldman and Michael Langberg. 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, 2011.

[15] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[16] Noah Gamboa, Kais Kudrolli, Anand Dhoot, and Ardavan Pedram. Campfire: Compressible, regularization-free, structured sparse training for hardware accelerators. *arXiv preprint arXiv:2001.03253*, 2020.

[17] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[20] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[21] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

[22] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[23] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[24] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2554–2564. IEEE, 2016.

[25] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[26] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[27] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5623–5632, 2019.

[28] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

[29] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2178–2188, 2017.

[30] Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

[31] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[32] 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.

[33] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020.

[34] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations*, 2018.

[35] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019.

[36] PyTorch contributors. Non-linear activations (weighted sum, nonlinearity). `https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity`, 2020. [Online; accessed 4-June-2020].

[37] PyTorch contributors. Unfold. `https://pytorch.org/docs/master/generated/torch.nn.Unfold.html`, 2020. [Online; accessed 9-June-2020].

[38] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020.

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[42] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015.

[43] Aravind Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29(2):648–670, 1999.

[44] Taiji Suzuki, Hiroshi Abe, and Tomoaki Nishimura. Compression based bound for non-compressed network: unified generalization error analysis of large compressible deep neural network. In *International Conference on Learning Representations*, 2020.

[45] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[46] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[47] Roman Vershynin. High-dimensional probability. *An Introduction with Applications*, 2016.

[48] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[49] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

[50] 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. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[51] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 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, 2017.

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

[53] Liang Zhao, Siyu Liao, Yanzhi Wang, Jian Tang, and Bo Yuan. Theoretical properties for neural networks with weight matrices of low displacement rank. *CoRR*, abs/1703.00144, 2017.

[54] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the imagenet scale: a pac-bayesian compression approach. In *International Conference on Learning Representations*, 2018.

A   Overview of Supplementary Material

In the following, we provide a quick overview of the material discussed in the supplementary material. We start out by providing a more complete problem definition including introducing additional notation that is required for our analysis (Section B). Subsequently, we introduce SiPP in full length including the generalization to all weights (Section C). We then provide a detailed analysis of SiPP and its variants to back up the informal claims of the main paper (Section D). Finally, we provide details and hyperparameters for our experimental setup and additional experimental results (Section E).

B   Notation and Problem Definition

The set of parameters θ of a neural network with L layers is a tuple of multi-dimensional weight tensors corresponding to each layer, i.e., $\theta = (W^1, \ldots, W^L)$. The set of parameters θ defines the mapping $f_\theta : \mathcal{X} \to \mathcal{Y}$ from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. We consider the setting where we have access to independent and identically distributed (i.i.d.) samples (x, y) from a joint distribution defined on $\mathcal{X} \times \mathcal{Y}$ from which we can gather a training, test, and validation data set. To this end, we let $\mathcal{D}$ denote the marginal distribution over the input space $\mathcal{X}$.

B.1   Network Notation

Layers.   For a given input $x \sim \mathcal{D}$, we denote the pre-activation and activation of layer ℓ by $Z^\ell(x)$ and $A^\ell(x)$, respectively. Note that

$$f_\theta(x) = A^L(x), \qquad A^0(x) = x, \qquad \text{and} \qquad A^\ell(x) = \phi^\ell(Z^\ell(x)),$$

where $\phi^\ell(\cdot)$ denotes the activation function for layer ℓ. We consider any multi-dimensional layer that can be described by a linear map with *parameter sharing*, e.g. fully-connected layers, convolutional layers, or LSTM cells. Specifically, for a layer ℓ the pre-activation $Z^\ell(x)$ of layer ℓ is described by the linear mapping of the activation $A^{\ell-1}(x)$ with W^ℓ, i.e.,

$$Z^\ell(x) = W^\ell * A^{\ell-1}(x),$$

where $*$ denotes the operator of the linear map, e.g., the convolutional operator.

Parameter groups.   We denote by c^ℓ the number of parameter groups within a layer ℓ that do not interact with each other, e.g., individual filters in convolutional layers. Then, let

$$Z_i^\ell(x) = W_i^\ell * A^{\ell-1}(x), \quad i \in [c^\ell],$$

denote the ith pre-activation channel of layer ℓ produced by parameter group W_i^ℓ. Then the entire pre-activations $Z^\ell(x)$ of a layer ℓ is constructed by appropriately concatenating the individual pre-activations from individual parameter groups, i.e.,

$$Z^\ell(x) = \left[Z_1^\ell(x), \ldots, Z_{c^\ell}^\ell(x) \right].$$

Moreover, we let $\eta^\ell \in \mathbb{N}$ denote the number of scalar values of $Z^\ell(\cdot)$ and let $\eta = \sum_{\ell=1}^L \eta^\ell$. Finally, let ρ denote the maximum number of parameters within a parameter group, i.e., $\rho = \max_{i,\ell} \|W_i^\ell\|_0$.

Patches. Within a parameter group, parameters may be used multiple times, c.f. *parameter sharing*, in order to produce the output $Z_i^\ell(x) = W_i^\ell * A^{\ell-1}(x)$. For example, in case of a convolutional layer the filter W_i^ℓ gets "slid" across the input $A^{\ell-1}(x)$ of the layer in order to produce one output pixel after another. Hereby, the filter acts on a distinct *patch* of the layer input $A^{\ell-1}(x)$ in order to produce a specific output pixel $z(x) \in Z_i^\ell(x)$, where with slight abuse of notation $z(x)$ denotes a scalar entry of $Z_i^\ell(x)$. To precisely specify the associated operation that produces the output $z(x)$ we define by $\mathcal{A}_i^\ell$ the *set of patches* of the layer input $A^{\ell-1}(x)$ that are required to produce the output $Z_i^\ell(x)$. Specifically, let $a(\cdot) \in \mathcal{A}_i^\ell$ denote some patch of $\mathcal{A}_i^\ell$. Then, $a(\cdot) \subseteq A^{\ell-1}(\cdot)$ is defined such that a dot product between the parameter group W_i^ℓ and the patch $a(\cdot)$ produces the associated output scalar $z(x)$, i.e.,

$$z(x) = \langle W_i^\ell, a(x) \rangle = \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x),$$

where $\mathcal{I}_i^\ell$ denotes the index set of weights for the parameter group W_i^ℓ and $w_k, a_k(x)$ denote a scalar entry of the parameter group and patch for some input $x \sim \mathcal{D}$, respectively. Note that $\eta^\ell = \sum_{i \in [c^\ell]} |\mathcal{A}_i^\ell|$.

The notation of patch maps $\mathcal{A}$ lets us conveniently abstract away some of the implementation details of the linear map $*$ without restricting ourselves to a particular type of linear map $*$. For example in the context of convolutional layers, the actual linear map $*$ can significantly vary depending on the parameter settings such as stride length, padding, and so forth. It also enables us to consider other layers, such as recurrent layers, at the same time. In this case, $\mathcal{A}$ can be generated by considering each recursive input to the layer as a separate patch.

In the case of two-dimensional convolutions (i.e. for images), we note that our notion of patch maps corresponds to the UNFOLD operation in PyTorch [37], which we find to be a helpful reference to further contextualize the concept of patch maps.

B.2 Problem Definition

We now proceed to formally state the problem definition that motivates the use of SIPP and subsequent analysis. To this end, let the size of the parameter tuple θ, $\|\theta\|_0$, to be the number of all non-zero entries in the weight tensors $W^1, \ldots, W^L$.

Problem 1. *For given $\varepsilon, \delta \in (0,1)$, our overarching goal is to use a pruning algorithm to generate a sparse reparameterization $\hat{\theta}$ of θ such that $\|\hat{\theta}\|_0 \ll \|\theta\|_0$ and for $x \sim \mathcal{D}$ the ℓ_2-norm of the reference network output $f_\theta(x)$ can be approximated by $f_{\hat{\theta}}(x)$ up to $1 \pm \varepsilon$ multiplicative error with probability greater than $1 - \delta$, i.e.,*

$$\mathbb{P}_{\hat{\theta},x}\left(\left\|f_{\hat{\theta}}(x) - f_\theta(x)\right\| \leq \varepsilon \left\|f_\theta(x)\right\|\right) \geq 1 - \delta,$$

where $\|\cdot\| = \|\cdot\|_2$ and $\mathbb{P}_{\hat{\theta},x}$ considers the randomness over both the pruning algorithm and the network's input.

C Method

In this section, we provide additional details for SIPP as introduced in the main part of the paper.

Algorithm 2 SiPP $(\theta, \mathcal{B}, \delta)$

Input: $\theta = (W^1, \ldots, W^L)$: weights of the uncompressed neural network; $\mathcal{B} \in \mathbb{N}$: sampling budget; $\delta \in (0,1)$: failure probability;
Output: $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$: sparse weights

1: $\mathcal{S} \leftarrow$ Uniform sample (without replacement) of $K \log(8\,\eta\,\rho/\delta)$ points from validation set
2: $\{m_i^\ell\}_{i,\ell} \leftarrow \text{OptAlloc}(\theta, \mathcal{B}, \mathcal{S}) \; \forall i \in [c^\ell], \; \forall \ell \in [L]$ ▷ optimally allocate budget $\mathcal{B}$ applying Lemma 7 to evaluate the resulting relative error guarantees
3: **for** $\ell \in [L]$ **do**
4:     $\hat{W}^\ell \leftarrow \mathbf{0}$;   ▷ Initialize a null tensor
5:     **for** $i \in [c^\ell]$ **do**
6:         $\{s_j\}_j \leftarrow \text{EmpiricalSensitivity}(\theta, \mathcal{S}, i, \ell) \; \forall w_j \in W_i^\ell$ ▷ Compute parameter importance for each weight w_j in the parameter group according to Definitions 2 and 3
7:         $\hat{W}_i^\ell \leftarrow \text{Sparsify}(W_i^\ell, m_i^\ell, \{s_j\}_j)$ ▷ prune weights according to SiPPDet, SiPPRand, or SiP-PHybrid such that only m_i^ℓ weights remain in the parameter group
8:     **end for**
9: **end for**
10: **return** $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$;

C.1 Overview

Algorithm 2 provides an extended over view of SiPP. Moreover, in Algorithm 3 we present Sparsify, which is the sub-routine to adaptively prune weights from a parameter group according to either SiPPDet, SiPPRand, or SiPPHybrid.

C.2 Details regarding OptAlloc

As mentioned in Section 4, OptAlloc proceeds by minimizing the sum of relative error guarantees associated with each parameter group for a given overall weight budget $\mathcal{B}$, i.e.,

$$\min_{m_i^\ell \in \mathbb{N} \; \forall i \in [c^\ell], \; \forall \ell \in [L]} \quad \sum_{\ell \in [L], \, i \in [c^\ell]} \varepsilon_i^\ell(m_i^\ell) \quad \text{s. t.} \quad \sum_{\ell \in [L], \, i \in [c^\ell]} m_i^\ell \leq \mathcal{B}.$$

Hereby, m_i^ℓ and $\varepsilon_i^\ell(m_i^\ell)$ denote the desired number of weights and the associated theoretical error in parameter group W_i^ℓ after pruning, respectively. We evaluate the theoretical error according to Lemmas 6 and Lemma 7 when pruning with SiPPDet and SiPPHybrid or SiPPRand, respectively. Note that in order to evaluate Lemma 7 we have to first convert m_i^ℓ to the expected number of required samples in order to obtain m_i^ℓ unique samples, which is also shown in Line 4 of Algorithm 3.

C.3 Details regarding EmpiricalSensitivity

We note that the empirical sensitivity (ES) s_j of a weight w_j in the parameter group W_i^ℓ is given by Definition 2, where we define ES as the maximum of the relative parameter importance $g_j(x)$ over a set $\mathcal{S}$ of i.i.d. data points. To account for both negative weights and activations, we utilize the generalized parameter importance as defined in Definition 3 to compute $g_j(x)$ for a particular input x.

Algorithm 3 SPARSIFY($W_i^\ell, m_i^\ell, \{s_j\}_j$)

Input: W_i^ℓ: parameter group to be pruned; m_i^ℓ: assigned budget; $\{s_j\}_j$: sensitivities associated with weights in parameter group
Output: $\hat{W}_i^\ell$: sparse parameter group

1: $S \leftarrow \sum_{j \in \mathcal{I}_i^\ell} s_j$ ▷ Compute sum of sensitivities
2: $\tilde{S} \leftarrow \frac{SC}{3} \log(16\eta/\delta)$
3: $\{q_j\}_j \leftarrow \frac{s_j}{S}$ ▷ Compute sample probabilities for SIPPRAND
4: $N \leftarrow N(m_i^\ell, \{q_j\}_j)$ ▷ Get expected number of required samples to obtain m_i^ℓ unique samples
5: $\mathcal{I}_{det} \leftarrow$ subset of indices from $\mathcal{I}_i^\ell$ corresponding to the largest m_i^ℓ sensitivities (c.f. Lemma 6)
6: $\varepsilon_{rand} \leftarrow \frac{\tilde{S} + \sqrt{\tilde{S}(\tilde{S}+6N)}}{N}$ ▷ c.f. Lemma 7
7: $\varepsilon_{det} \leftarrow C \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_{det})} s_j$ ▷ c.f. Lemma 6
8: **if** ($\varepsilon_{rand} > \varepsilon_{det}$ **or** always SIPPDET) **and** not always SIPPRAND **then**
9:    $\hat{W}_i^\ell \leftarrow$ prune weights from W_i^ℓ, i.e set to 0, that are not in $\mathcal{I}_{det}$ and keep the rest
10: **else**
11:    $\{n_j\}_j \sim$ MULTINOMIAL($\{q_j\}_j, N$)   ▷ Sample N times and return the counts
12:    $\hat{W}_i^\ell \leftarrow$ prune weights such that $\hat{w}_j = \frac{n_j}{Nq_j} w_j$ for each weight $\hat{w}_j$, $j \in \mathcal{I}_i^\ell$ in the parameter group
13: **end if**
14: **return** $\hat{W}_i^\ell$

To ensure that ES holds with probability at least $1 - \delta$ for all patches and parameters simultaneously we have to appropriately choose the size of $\mathcal{S}$, c.f. Line 1 of Algorithm 2 and Section D.4.

C.4   Details regarding SPARSIFY

In Algorithm 3 we present the pruning strategy for both SIPPDET and SIPPRAND as shown in Line 9 and Lines 11, 12, respectively. Recall that SIPPHYBRID adaptively chooses between both strategies according to the associated error guarantees, which get computed in Lines 7 and 6 for SIPPDET and SIPPRAND, respectively. We then choose the better strategy accordingly, see Line 8. We can also choose to always prune using SIPPRAND or SIPPDET as indicated in Line 8.

C.5   Simple SIPP

We can greatly simplify our pruning algorithm if we prune all parameter groups using SIPPDET. To see this consider the solution to OPTALLOC when evaluating the relative error according to Lemma 6. Since the relative error for a particular parameter group in this case is the sum over sensitivities that were not included and the objective of OPTALLOC is to minimize the sum over all relative errors the optimal solution is to *globally* keep the weights with largest sensitivity. In other words, pruning with SIPPDET only results in global thresholding of weights according to their sensitivity. The resulting procedure is shown in Algorithm 4. Note that this procedure is very reminiscent of simple, global weight thresholding [17, 38] but using sensitivity instead of the magnitude of the weights as prune criterion. In contrast to weight thresholding, however, SIPPSIMPLE still exhibits the same theoretical error guarantees as SIPP.

Algorithm 4 SIPPSIMPLE $(\theta, \mathcal{B}, \delta)$

Input: $\theta = (W^1, \ldots, W^L)$: weights of the uncompressed neural network; $\mathcal{B} \in \mathbb{N}$: sampling budget; $\delta \in (0, 1)$: failure probability;
Output: $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$: sparse weights

1: $\mathcal{S} \leftarrow$ Uniform sample of $K \log (16 \, \eta \, \rho / \delta)$ points from validation set

2: Compute sensitivity for all weights in the network using $\mathcal{S}$

3: Prune weights globally by keeping the $\mathcal{B}$ weights with largest sensitivity

4: Return $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$

D   Analysis

In this section, we establish the theoretical guarantees of SIPP as presented in Algorithm 2 and state our main compression theorem.

D.1   Outline

We begin by considering the sparsification of an arbitrary output patch in an arbitrary parameter group and layer assuming that both the input to the layer and the weights are non-negative. To this end, we first establish the empirical sensitivity (ES) inequality that quantifies the contribution of an individual scalar weight to an output patch (Section D.2 and informal Lemma 5.1). We then establish the relative error guarantees for each variation of SIPP for an arbitrary output patch (Section D.3 and informal Lemmas 5.2, 5.3). Next, we formally generalize the approximation scheme to arbitrary weights and input activations (Section D.4). Finally, we provide our formal network compression bounds by composing together the error guarantees from individual layers and parameter groups. (Section D.5).

D.2   Empirical Sensitivity

Recall that an arbitrary parameter group indexed by $i \in [c^\ell]$ in an arbitrary layer $\ell \in [L]$, is denoted by W_i^ℓ and $\mathcal{I}$ denotes its parameter index set. Moreover, let w_j denote some scalar entry of W_i^ℓ for some $j \in \mathcal{I}$. Also as before, $A^{\ell-1}(x)$ denotes the input activation to layer ℓ and $Z_i^\ell(x) = W_i^\ell * A^{\ell-1}(x)$ denotes the output pre-activation of parameter group W_i^ℓ. Finally, recall that $a(\cdot) \in \mathcal{A}_i^\ell$ denotes some patch of $\mathcal{A}_i^\ell$ and that the patch $a(\cdot)$ produces the associated output scalar $z(x)$, i.e., $z(x) = \langle W_i^\ell, a(x) \rangle = \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x)$, We now proceed with the formal definition of relative parameter importance and empirically sensitivity, which is defined as the maximum relative parameter importance over multiple data points.

Definition 1 (Relative parameter importance)**.** *For a scalar parameter w_j, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ in layer ℓ, its relative importance $g_j(x)$ is given by*

$$g_j(x) = \max_{a(\cdot) \in \mathcal{A}_i^\ell} \frac{w_j \, a_j(x)}{\sum_{k \in \mathcal{I}_i^\ell} w_k \, a_k(x)},$$

where $\mathcal{A}_i^\ell$ denotes the set of patches for parameter group W_i^ℓ.

Definition 2 (Empirical sensitivity). *Let $\mathcal{S}$ be a set of i.i.d. samples from the validation data set. Then, the empirical sensitivity $s_j(x)$ of a scalar parameter w_j, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ in layer ℓ is given by*

$$s_j(x) = \max_{x \in \mathcal{S}} g_j(x).$$

We note that, for ease of notation, we do not explicitly enumerate ES over i and ℓ for parameter groups and layers, respectively.

To ensure that a *small* batch of points $\mathcal{S}$ suffices for an accurate approximation of parameter importance, we impose the following mild regularity assumption on the Cumulative Distribution Function (CDF) of $g_j(x)$ similar to the assumption of [6].

Assumption 1 (Regularity assumption). *There exist universal constants $C, K > 0$ such that for all $j \in \mathcal{I}_i^\ell$, the CDF of the random variable $g_j(x) \in [0, 1]$ for $x \sim \mathcal{D}$, denoted by $F_j(\cdot)$, satisfies*

$$F_j\left(1/C\right) \leq \exp\left(-1/K\right).$$

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'. In other words, Assumption 1 ensures that there are no outliers of $g_j(x)$ with non-negligible probability that are *not* within a constant multiplicative factor of most other values of $g_j(x)$. Capturing outliers that are within a constant multiplicative factor, on the other hand, can be captured by considering an appropriate scaling factor of ES in the ES inequality (see Lemma 5 below). However, we cannot capture these non-negligible outliers (unless we significantly increase the cardinality of $\mathcal{S}$) when they are not within a constant multiplicative factor.

We now proceed to state the ES inequality as informally stated in Lemma 5.1 in the main body of the paper. We note that, intuitively, the ES inequality enables us to quantify ,i.e. upper-bound, the contribution $w_j a_j(x)$ coming from an individual weight w.h.p. in terms of the output patch $z(x)$ and the sensitivity s_j of the weight.

Lemma 5 (ES inequality). *For $\delta \in (0, 1)$, the ES s_j of the scalar parameter w_j, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ computed with a set $\mathcal{S}$ of i.i.d. data points, $|\mathcal{S}| = K \log(\rho/\delta)$, satisfies*

$$\mathbb{P}_x\left(w_j a_j(x) \leq C s_j z(x)\right) \geq 1 - \delta \quad \forall j \in \mathcal{I}_i^\ell,$$

for some input $x \sim \mathcal{D}$ and some fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$, where C, K are the universal constants of Assumption 1 and $z(x) = \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x)$.

Proof. We consider a fixed weight w_j from the parameter group and a fixed input patch $a(\cdot)$. Note that

$$\frac{w_j a_j(x)}{z(x)} \leq g_j(x) \tag{2}$$

by definition of $g_j(x)$ since the relative parameter importance is the maximum over patches for a specific input x. We now consider the probability that $s_j(\mathcal{S}) = \max_{x' \in \mathcal{S}} g_j(x')$ is not an upper bound for $g_j(x)$ when appropriately scaled for random draws over $\mathcal{S}$, where we explicitly denote the dependency of $s_j(\mathcal{S})$ on $\mathcal{S}$ for the purpose of this proof. By showing this occurs with low probability we can then conclude that s_j is indeed an upper bound for $g_j(x)$ most of the time. Specifically,

$$\mathbb{P}_{\mathcal{S}}\left(C s_j(\mathcal{S}) \leq g_j(x)\right) = \mathbb{P}_{\mathcal{S}}\left(s_j(\mathcal{S}) \leq g_j(x)/C\right)$$

$$\leq \underset{\mathcal{S}}{\mathbb{P}}\left(s_j(\mathcal{S}) \leq \mathrm{1}/C\right) \qquad\qquad\qquad \text{since } g_j(x) \leq 1$$

$$= \underset{\mathcal{S}}{\mathbb{P}}\left(\max_{x' \in \mathcal{S}} g_j(x') \leq \mathrm{1}/C\right) \qquad\qquad\qquad \text{by definition of } s_j(\mathcal{S})$$

$$= \left(\underset{x'}{\mathbb{P}}\left(g_j(x') \leq \mathrm{1}/C\right)\right)^{|\mathcal{S}|} \qquad\qquad \text{since } \mathcal{S} \text{ is } |\mathcal{S}| \text{ i.i.d. draws from } \mathcal{D}$$

$$= F_j\left(\mathrm{1}/C\right)^{|\mathcal{S}|} \qquad\qquad\qquad \text{since } F_j(\cdot) \text{ is the CDF of } g_j(x')$$

$$\leq \exp\left(-|\mathcal{S}|/K\right) \qquad\qquad\qquad\qquad \text{by Assumption 1}$$

$$= \frac{\delta}{\rho} \qquad\qquad\qquad \text{since } |\mathcal{S}| = K \log\left(\rho/\delta\right) \text{ by definition.}$$

Thus, we can conclude that for a fixed weight w_j and some input $x \sim \mathcal{D}$ its relative contribution $g_j(x)$ is upper bound by its sensitivity s_j. Moreover, the inequality also holds for any weight w_j by the union bound, i.e.,

$$\underset{\mathcal{S}}{\mathbb{P}}\left(\exists j \in \mathcal{I}_i^\ell | C s_j(\mathcal{S}) \leq g_j(x)\right) \leq \left|\mathcal{I}_i^\ell\right| \underset{\mathcal{S}}{\mathbb{P}}\left(C s_j(\mathcal{S}) \leq g_j(x)\right) \qquad \text{by the union bound}$$

$$\leq \left|\mathcal{I}_i^\ell\right| \frac{\delta}{\rho} \qquad\qquad\qquad \text{by the analysis above}$$

$$\leq \rho \frac{\delta}{\rho} \qquad\qquad\qquad \text{since } \rho = \max_{i,\ell} \left|\mathcal{I}_i^\ell\right|$$

$$= \delta$$

We thus have with probability at least $1 - \delta$ over the construction of s_j that

$$C s_j \geq g_j(x) \quad \forall j \in \mathcal{I}_i^\ell$$

and by (2) that

$$\frac{w_j a_j(x)}{z(x)} \leq g_j(x) \leq C s_j,$$

which concludes the proof since the above inequality holds for any $x \sim \mathcal{D}$. $\qquad\qquad\square$

D.3 Error Guarantees for positive weights and activations

Equipped with Lemma 5 we now proceed to establish the relative error guarantees for the three variants of SIPP. As before, we consider a fixed output patch for a fixed parameter group and we assume that both input activations and the weights are non-negative.

D.3.1 Error Guarantee for SIPPDET

Recall that SIPPDET prunes weights from a parameter group by keeping only the weights with largest sensitivity. Let the index set of weights kept be denoted by $\mathcal{I}_{det}$. Below we state the formal error guarantee for a fixed output patch of a parameter group when we only keep the weights indexed by $\mathcal{I}_{det}$. Note that the below error guarantee holds for any index set $\mathcal{I}_{det}$ that we decide to keep. Naturally, however, it makes sense to keep the weights with largest sensitivity as this minimizes the associated relative error.

Lemma 6 (SiPPDet error bound). *For $\delta \in (0,1)$, pruning parameter group W_i^ℓ by keeping only the weights indexed by $\mathcal{I}_{det} \subseteq \mathcal{I}_i^\ell$ generates a pruned parameter group $\hat{W}_i^\ell$ such that for a fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$ and $x \sim \mathcal{D}$*

$$\mathbb{P}\left(|\hat{z}_{det}(x) - z(x)| \geq \varepsilon_{det}z(x)\right) \leq \delta \quad with \quad \varepsilon_{det} = C \sum_{j \in \mathcal{I}\backslash\mathcal{I}_{det}} s_j \in (0,1),$$

where $z(x) = \langle W_i^\ell, a(x) \rangle = \sum_{j \in \mathcal{I}_i^\ell} w_j a_j(x)$ and $\hat{z}_{det}(x) = \langle \hat{W}_i^\ell, a(x) \rangle = \sum_{j \in \mathcal{I}_{det}} w_j a_j(x)$ denote the unpruned and approximate output patch, respectively, associated with the input patch $a(\cdot)$. The sensitivities $\{s_j\}_{j \in \mathcal{I}_i^\ell}$ are hereby computed over a set $\mathcal{S}$ of $K \log(\rho/\delta)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. We proceed by considering the absolute difference $|z(x) - \hat{z}_{det}(x)|$ and note that

$$|z(x) - \hat{z}_{det}(x)| = \left| \langle W_i^\ell, a(x) \rangle - \langle \hat{W}_i^\ell, a(x) \rangle \right|$$

$$= \left| \sum_{j \in \mathcal{I}_i^\ell} w_j a_j(x) - \sum_{j \in \mathcal{I}_{det}} w_j a_j(x) \right|$$

$$= \sum_{j \in \mathcal{I}_i^\ell \backslash \mathcal{I}_{det}} w_j a_j(x)$$

Invoking Lemma 5 we know that with probability at least $1 - \delta$ each individual weight term in the above sum is upper bound by its sensitivity, i.e.,

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

We now bound the error in terms of sensitivity as

$$|z(x) - \hat{z}_{det}(x)| = \sum_{j \in \mathcal{I}_i^\ell \backslash \mathcal{I}_{det}} w_j a_j(x)$$

$$\leq \sum_{j \in \mathcal{I}_i^\ell \backslash \mathcal{I}_{det}} C s_j z(x) \qquad \text{using the above inequality}$$

$$= \varepsilon_{det} z(x) \qquad\qquad \text{by definition of } \varepsilon_{det}.$$

We conclude by mentioning that above error bound holds with probability at least $1 - \delta$ since the associated ES inequalities hold with probability at least $1 - \delta$. $\qquad\square$

D.3.2 Error Guarantee for SiPPRand

As before, we consider a fixed parameter group W_i^ℓ, which has been assigned a budget of m_i^ℓ unique weights to be kept. Recall that SiPPRand is a sampling procedure that proceeds as follows:

1. Assign probabilities $q_j = s_j / \sum_{k \in \mathcal{I}_i^\ell} s_k$ for all $j \in \mathcal{I}_i^\ell$.

2. Compute the expected number of samples, N, to obtain m_i^ℓ unique weights from the sampling procedure.

3. Sample weights N times with replacement from W_i^ℓ according to q_j.

4. Reweigh the weights, $\hat{w}_j$, to obtain the approximate weights such that $\hat{w}_j = \frac{n_j}{Nq_j}w_j$, where n_j denotes the number of times w_j was sampled.

We note that if a weight has not been sampled, i.e. $n_j = 0$, we can drop it since the resulting weight is 0. We now consider the resulting error bound when sampling N times with replacement.

Lemma 7 (SIPPRAND error bound). *For $\delta \in (0,1)$, pruning parameter group W_i^ℓ by sampling weights $N = N(m_i^\ell)$ times with replacement, such that weight w_j is sampled with probability $q_j = s_j / \sum_{k \in \mathcal{I}_i^\ell} s_k$, generates a pruned parameter group $\hat{W}_i^\ell$ such that for a fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$ and $x \sim \mathcal{D}$*

$$\mathbb{P}\left(|\hat{z}_{rand}(x) - z(x)| \geq \varepsilon_{rand}z(x)\right) \leq \delta \text{ and } \varepsilon_{rand} = \left(\sqrt{\frac{\tilde{S}}{N}\left(\frac{\tilde{S}}{N} + 6\right)} + \frac{\tilde{S}}{N}\right) \in (0,1),$$

where $\hat{z}_{rand}(x) = \langle \hat{W}_i^\ell, a(x) \rangle$ and $z(x) = \langle W_i^\ell, a(x) \rangle$ are with respect to patch map $a(\cdot)$ as before, $\tilde{S} = \frac{SC}{3}\log(4/\delta)$, and $S = \sum_{j \in \mathcal{I}_i^\ell} s_j$. The sensitivities $\{s_j\}_{j \in \mathcal{I}_i^\ell}$ are hereby computed over a set $\mathcal{S}$ of $K \log(2\rho/\delta)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. Our proof closely follows the proof of Lemma 1 of [6]. The sampling procedure of sampling N with replacement is equivalent to sequentially constructing a *multiset* consisting of N samples from $\mathcal{I}_i^\ell$ where each $j \in \mathcal{I}_i^\ell$ is sampled with probability q_j. Now, let $\mathcal{C} = \{c_1, \ldots, c_N\}$ be that multiset of weight indices $\mathcal{I}_i^\ell$ used to construct $\hat{W}_i^\ell$. Let $a(\cdot) \in \mathcal{A}_i^\ell$ be arbitrary and fixed, let $x \sim \mathcal{D}$ be an i.i.d. sample from $\mathcal{D}$, and let

$$\hat{z}_{rand}(x) = \langle \hat{W}_i^\ell, a(x) \rangle = \sum_{j \in \mathcal{C}} \frac{w_j}{Nq_j}a_j(x)$$

be the approximate intermediate value corresponding to the sparsified tensor $\hat{W}_i^\ell$ and let

$$z(x) = \sum_{j \in \mathcal{I}_i^\ell} w_j a_j(x)$$

as before. Define N random variables $T_{c_1}, \ldots, T_{c_N}$ such that for all $j \in \mathcal{C}$

$$T_j = \frac{w_j a_j(x)}{Nq_j} = \frac{Sw_j a_j(x)}{Ns_j}. \tag{3}$$

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

$$\mathbb{E}[T_j] = \sum_{k \in \mathcal{I}_i^\ell} \frac{w_k a_k(x)}{Nq_k}q_k = \frac{z(x)}{N}.$$

Let $T = \sum_{j \in \mathcal{C}} T_j = \hat{z}_{rand}(x)$ denote our approximation and note that by linearity of expectation,

$$\mathbb{E}[T] = \sum_{j \in \mathcal{C}} \mathbb{E}[T_j] = z(x).$$

Thus, $\hat{z}_{rand}(x) = T$ is an unbiased estimator of $z(x)$ for any $x \sim \mathcal{D}$.

For the remainder of the proof we will assume that $z(x) > 0$, since otherwise, $z(x) = 0$ 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

24

with the case where $z(x) > 0$ and invoke Lemma 5 (ES inequality) with $\mathcal{S}$ consisting of $K \log (2\,\rho/\delta)$ i.i.d. data points, which implies that

$$w_j a_j(x) \leq C s_j z(x) \quad \forall j \in \mathcal{I}_i^\ell \qquad \text{with probability at least} \qquad 1 - \frac{\delta}{2}. \tag{4}$$

Consequently, we can bound the variance of each T_j, $j \in \mathcal{C}$ with probability at least $1 - \delta/2$ as follows

$$\begin{aligned}
\mathrm{Var}(T_j) &\leq \mathbb{E}\left[T_j^2\right] \\
&= \sum_{k \in \mathcal{I}_i^\ell} \frac{(w_k a_k(x))^2}{(N q_k)^2} q_k \\
&= \frac{S}{N^2} \sum_{k \in \mathcal{I}_i^\ell} \frac{w_k a_k(x)}{s_j} w_k a_k(x) \\
&\leq \frac{S}{N^2} C z(x) \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x) \qquad \text{by the ES inequality as stated in (4)} \\
&= \frac{S C z(x)^2}{N^2}.
\end{aligned}$$

Since T is a sum of independent random variables, we obtain

$$\mathrm{Var}(T) = N \, \mathrm{Var}(T_j) \leq \frac{S C z(x)^2}{N} \tag{5}$$

for the overall variance.

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

$$\tilde{T}_j = T_j - \mathbb{E}\left[T_j\right] = T_j - z(x),$$

and let $\tilde{T} = \sum_{j \in \mathcal{C}} \tilde{T}_j$. Note that by the definition of T_j and the ES inequality (4) we have that

$$T_j = \frac{S w_j a_j(x)}{N s_j} \leq \frac{S C z(x)}{N}$$

and consequently for the centered random variable $\tilde{T}_j$ that

$$\left|\tilde{T}_j\right| = \left|T_j - \frac{z(x)}{N}\right| \leq \frac{S C z(x)}{N} =: M, \tag{6}$$

which holds with probability at least $1 - \delta/2$ for any $x \sim \mathcal{D}$. Also note that $\mathrm{Var}(\tilde{T}) = \mathrm{Var}(T)$.

Now conditioned on the ES inequality (4) holding, applying Bernstein's inequality to both $\tilde{T}$ and $-\tilde{T}$ we have by symmetry and the union bound,

$$\begin{aligned}
\mathbb{P}\left(\left|\tilde{T}\right| \geq \varepsilon_{rand} z(x)\right) &= \mathbb{P}\left(|T - z(x)| \geq \varepsilon_{rand} z(x)\right) \\
&\leq 2 \exp\left(-\frac{\varepsilon_{rand}^2 z(x)^2}{2\,\mathrm{Var}(T) + \frac{2\varepsilon_{rand} z(x) M}{3}}\right) \qquad \text{by Bernstein's inequality}
\end{aligned}$$

25

$$\leq 2 \exp\left(-\frac{\varepsilon_{rand}^2 z(x)^2}{\frac{2SCz(x)^2}{N} + \frac{2SC\varepsilon_{rand}z(x)^2}{3N}} \right) \qquad \text{by (5) and (6)}$$

$$= 2 \exp\left(-\frac{3\varepsilon_{rand}^2 N}{SC(6 + 2\varepsilon_{rand})} \right)$$

$$\leq \frac{\delta}{2} \qquad \text{by our choice of } \varepsilon_{rand}$$

Note that the (undesired) event $|T - z(x)| \geq \varepsilon_{rand}z(x) = |\hat{z}_{rand}(x) - z(x)| \geq \varepsilon_{rand}z(x)$ occurs with probability at most $\delta/2$, which was conditioned on the ES inequality holding, which occurs with probability at least $1 - \delta/2$. Thus by the union bound, the overall failure probability is at most δ, which concludes the proof. $\qquad\square$

D.3.3 Error Guarantee for SiPPHybrid

We note that the error guarantee for SiPPHybrid follow straightforward from the error guarantees for SiPPDet and SiPPRand as stated in Lemma 6 and 7, respectively, since SiPPHybrid chooses the strategy among those two for which the associated error guarantee is lower. We can therefore state the error guarantee as follows.

Lemma 8 (SiPPHybrid error bound)**.** *In the context of Lemmas 6 and 7, for $\delta \in (0, 1)$ SiPPHybrid generates a pruned parameter group $\hat{W}_i^\ell$ such that for a fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$, output patch $z(x)$, and $x \sim D$*

$$\mathbb{P}\left(|\hat{z}_{hybrid}(x) - z(x)| \geq \varepsilon_{hybrid}z(x) \right) \leq \delta \quad \text{with} \quad \varepsilon_{hybrid} = \min\{\varepsilon_{det}, \varepsilon_{rand}\} \in (0, 1),$$

where $z_{hybrid}(x)$ is the associated approximate output patch.

D.4 Generalization to all weights and activations

In this section, we generalize our analysis from the previous section to include all weights and activations. We also adapt the resulting error guarantees to simultaneously hold for all patches of all parameter groups within a layer instead of a fixed patch.

We handle the general case by splitting both the input activations and the weights into their respective positive and negative parts representing the four quadrants, i.e.,

$$z^{++}(x) = \langle W_i^{\ell,+}, a(x)^+ \rangle \qquad\qquad z^{+-}(x) = \langle W_i^{\ell,+}, a(x)^- \rangle$$
$$z^{-+}(x) = \langle W_i^{\ell,-}, a(x)^+ \rangle \qquad\qquad z^{--}(x) = \langle W_i^{\ell,+}, a(x)^- \rangle,$$

where

$$W_i^\ell = W_i^{\ell,+} - W_i^{\ell,-}, \quad W_i^{\ell,+}, W_i^{\ell,-} \geq 0,$$
$$a(x) = a(x)^+ - a(x)^-, \quad a(x)^+, a(x)^- \geq 0.$$

First, consider negative activations for a non-negative parameter group. Specifically, when computing sensitivities over some set $\mathcal{S}$ we split the input activations into their respective positive and negative part, and take an additional maximum over both parts. Henceforth the ES inequality 5 can be applied to the positive and negative part of $a(x)$ at the same time. Similarly, we can split

the parameter group into its positive and negative part when computing sensitivity such that the ES inequality 5 holds for both parts of the parameter group as well.

More formally, the generalized relative parameter importance $g_j(x)$ for some parameter w_j of parameter group W_i^ℓ can be defined as follows.

Definition 3 (Generalized relative parameter importance). *For a scalar parameter $w_j = w_j^+ - w_j^-$, $w_j^+, w_j^- \geq 0$, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ in layer ℓ, its generalized relative importance $g_j(x)$ is given by the maximum over its quadrant-wise relative importances, i.e.,*

$$g_j(x) = \max\{g_j^{++}(x), g_j^{+-}(x), g_j^{-+}(x), g_j^{--}(x)\},$$

where

$$g_j^{++}(x) = \max_{a(\cdot) \in \mathcal{A}_i^\ell} \frac{w_j^+ a_j^+(x)}{\sum_{k \in \mathcal{I}_i^\ell} w_k^+ a_k^+(x)}, \text{ and so forth,}$$

and where $\mathcal{A}_i^\ell$ denotes the set of patches for parameter group W_i^ℓ and $\mathcal{A}_i^\ell \ni a(\cdot) = a^+(\cdot) - a^-(\cdot)$, $a^+(\cdot), a^-(\cdot) \geq 0$.

The definition of generalized ES does not change compared to Definition 2 and henceforth we do not re-state it explicitly. We proceed by re-deriving the ES inequality for the generalized parameter importance and any patch of the parameter group.

Lemma 9 (Generalized ES inequality). *For $\delta \in (0,1)$, the ES s_j of the scalar parameter w_j, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ computed with a set $\mathcal{S}$ of i.i.d. data points, $|\mathcal{S}| = K \log(\rho/\delta)$, satisfies for each quadrant*

$$\mathbb{P}_x\left(w_j^+ a_j^+(x) \leq C s_j z^{++}(x)\right) \geq 1 - \delta \quad \forall j \in \mathcal{I}_i^\ell, \text{ and so forth,}$$

for some input $x \sim \mathcal{D}$ and some fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$, where C, K are the universal constants of Assumption 1 and $z^{++}(x) = \sum_{k \in \mathcal{I}_i^\ell} w_k^+ a_k^+(x)$, and so forth, denotes the quadrant-wise output patch.

Proof. The proof follows the steps of Lemma 5 with the exception of Equation (2). To adapt it to the general case note that

$$\frac{w_j^+ a_j^+(x)}{z^{++}(x)} \leq g_j^{++}(x) \leq g_j(x),$$

and so forth, for each quadrant. □

Consequently, we can re-derive Lemmas 6-8 such that they hold for each quadrant of a fixed patch. The derivations are analogues to the derivations in Section D.3. Finally, we adapt our guarantees to hold quadrant-wise for all patches of all parameter groups and layers simultaneously. We note that we can achieve this by appropriately adjusting the failure probability for Lemmas 6-8 such that, by the union bound, the overall failure probability is bounded δ. Specifically, we can invoke Lemmas 6-8 with $\delta' = \delta/4\eta$ such that

$$\tilde{S} = \frac{SC}{3} \log(16\eta/\delta) \qquad \text{and} \qquad |\mathcal{S}| = K \log(8\eta\,\rho/\delta),$$

where η denotes the number of total patches across all layers and parameter groups. The rest of the Lemmas remains unchanged. Therefore, we have that for all quadrant-wise patches our error guarantees hold. We utilize our patch-wise bounds as outlined in Section C to optimally allocate our budget across layers to minimize the relative error within each quadrant of the parameter groups and prune each parameter group according to the budget and the desired variant of SiPP.

D.5 Network compression bounds

Up to this point we have established patch-wise and quadrant-wise error guarantees for the network, which suffices to prune the network according to Algorithm 2. However, we can also leverage our theoretical guarantees to establish network-wide compression bounds of the form

$$\mathbb{P}_{\hat{\theta},x}\left(\left\|f_{\hat{\theta}}(x) - f_{\theta}(x)\right\| \leq \varepsilon\left\|f_{\theta}(x)\right\|\right) \geq 1 - \delta,$$

for given $\varepsilon, \delta \in (0,1)$ as described in Problem 1.

We will restrict ourselves to analyzing the general case for SiPPDet but we note that each step can be applied analogously for SiPPRand and SiPPHybrid. We begin by generalizing Lemma 6 to establish norm-based bounds for each quadrant of the pre-activation. To this end, let

$$Z^{\ell++}(x) = W^{\ell+} * A^{\ell-1,+}(x), \quad \hat{Z}^{\ell++}(x) = \hat{W}^{\ell+} * A^{\ell-1,+}(x), \quad \text{and so forth}$$

denote the unpruned and approximate pre-activation quadrants, respectively. Moreover, let S_i^{ℓ} denote the sum of ES for parameter group W_i^{ℓ} as before and let $S_i^{\ell}(N_i^{\ell})$ denote the sum over the N_i^{ℓ} largest ES for parameter group W_i^{ℓ}.

Corollary 10. *For $\delta \in (0,1)$, pruning layer ℓ according to SiPPDet generates a pruned weight tensor $\hat{W}^{\ell}$ such that for a fixed quadrant and $x \sim \mathcal{D}$*

$$\mathbb{P}\left(\left\|\hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x)\right\| \geq \varepsilon^{\ell}\left\|Z^{\ell++}(x)\right\|\right) \leq \delta \quad with \quad \varepsilon^{\ell} = \max_{i \in c^{\ell}} C\left(S_i^{\ell} - S_i^{\ell}(N_i^{\ell})\right),$$

where N_i^{ℓ} denotes the number of samples allocated to parameter group W_i^{ℓ}. The ESs are hereby computed over a set $\mathcal{S}$ of $K \log\left(\eta^{\ell}\rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. Let $Z_i^{\ell++}(x)$ denote the pre-activation quadrant associated with parameter group W_i^{ℓ}. Invoking Lemma 6 with a set $\mathcal{S}$ of $K \log\left(\eta^{\ell}\rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$ and N_i^{ℓ} samples for the respective parameter group implies that any associated patch, i.e. entry, of $Z_i^{\ell++}(x)$ is approximated with relative error at most $\varepsilon_i^{\ell} = C\left(S_i^{\ell} - S_i^{\ell}(N_i^{\ell})\right)$ with probability at least $1 - \delta/\eta^{\ell}$. Consequently, $\left\|Z_i^{\ell++}(x)\right\|$ is also preserved with relative error ε_i^{ℓ}. Thus we have w.h.p. that

$$\begin{aligned}
\left\|Z^{\ell++}(x) - \hat{Z}^{\ell++}(x)\right\|^2 &= \sum_{i \in [c^{\ell}]} \left\|Z_i^{\ell++}(x) - \hat{Z}_i^{\ell++}(x)\right\|^2 \\
&\leq \sum_{i \in [c^{\ell}]} (\varepsilon_i^{\ell})^2 \left\|Z_i^{\ell++}(x)\right\|^2 \\
&\leq (\varepsilon^{\ell})^2 \sum_{i \in [c^{\ell}]} \left\|Z_i^{\ell++}(x)\right\|^2 \qquad \text{by definition of } \varepsilon^{\ell} \\
&= (\varepsilon^{\ell})^2 \left\|Z^{\ell++}(x)\right\|^2
\end{aligned}$$

Taking a union bound over all η^{ℓ} patches in the pre-activation $Z^{\ell}(x)$ concludes the proof. $\qquad\square$

We note that Corollary 10 is stated for $Z^{\ell++}(x)$ but naturally extends to the other quadrants as well.

As a next step, we establish guarantees to approximate $Z^{\ell}(x)$ by leveraging the guarantees for each quadrant. To this end, note that $Z^{\ell}(x) = Z^{\ell++}(x) - Z^{\ell+-}(x) - Z^{\ell-+}(x) + Z^{\ell--}(x)$. Further, let Δ^{ℓ} denote the "sign complexity" of approximating the overall pre-activation, which is defined as

28

Definition 4 (Sign complexity). *For layer ℓ, its sign complexity Δ^ℓ is given by*

$$\Delta^\ell = \max_{x \in \mathcal{S}} \frac{\left\| Z^{\ell++}(x) \right\| + \left\| Z^{\ell+-}(x) \right\| + \left\| Z^{\ell-+}(x) \right\| + \left\| Z^{\ell--}(x) \right\|}{\left\| Z^\ell(x) \right\|}$$

where $\mathcal{S}$ denotes a set of i.i.d. data points drawn from $\mathcal{D}$.

Intuitively, Δ^ℓ captures the additional complexity of approximating the layer when considering the actual signs of the quadrants as opposed to treating them separately. We can now state the error guarantees for SiPP in context of Corollary 10 for the overall pre-activation.

Lemma 11 (Layer error bound). *For given $\delta \in (0,1)$ and sample budget N_i^ℓ for each parameter group, invoking SiPPDet to prune W^ℓ generates a pruned weight tensor $\hat{W}^\ell$ such that for $x \sim \mathcal{D}$*

$$\mathbb{P}\left(\left\| \hat{Z}^\ell(x) - \hat{Z}^\ell(x) \right\| \geq \varepsilon^\ell \Delta^\ell \left\| Z^\ell(x) \right\| \right) \leq \delta \quad \text{with} \quad \varepsilon^\ell = \max_{i \in c^\ell} C\left(S_i^\ell - S_i^\ell(N_i^\ell) \right),$$

where N_i^ℓ denotes the number of samples allocated to parameter group W_i^ℓ. The ESs are hereby computed over a set $\mathcal{S}$ of $K \log\left(5\eta^\ell \rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. Consider invoking Corollary 10 with a set $\mathcal{S}$ of $K \log\left(5\eta^\ell \rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$. Then for each quadrant we have w.h.p. that

$$\left\| \hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x) \right\| \leq \varepsilon^\ell \left\| \hat{Z}^{\ell++}(x) \right\|, \text{ and so forth,}$$

for an appropriate notion of high probability specified subsequently. Note that

$$Z^\ell(x) = Z^{\ell++}(x) - Z^{\ell+-}(x) - Z^{\ell-+}(x) + Z^{\ell--}(x)$$

and so w.h.p. we have that

$$\begin{aligned}
\left\| \hat{Z}^\ell(x) - \hat{Z}^\ell(x) \right\| &= \left\| \hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x) - \hat{Z}^{\ell+-}(x) + \hat{Z}^{\ell+-}(x) \right. \\
&\quad \left. - \hat{Z}^{\ell-+}(x) + \hat{Z}^{\ell-+}(x) + \hat{Z}^{\ell--}(x) - \hat{Z}^{\ell--}(x) \right\| \\
&\leq \left\| \hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x) \right\| + \left\| \hat{Z}^{\ell+-}(x) - \hat{Z}^{\ell+-}(x) \right\| \\
&\quad + \left\| \hat{Z}^{\ell-+}(x) - \hat{Z}^{\ell-+}(x) \right\| + \left\| \hat{Z}^{\ell--}(x) - \hat{Z}^{\ell--}(x) \right\| \\
&\leq \varepsilon^\ell \left(\left\| \hat{Z}^{\ell++}(x) \right\| + \left\| \hat{Z}^{\ell+-}(x) \right\| + \left\| \hat{Z}^{\ell-+}(x) \right\| + \left\| \hat{Z}^{\ell--}(x) \right\| \right) \\
&= \varepsilon^\ell \frac{\left\| Z^{\ell++}(x) \right\| + \left\| Z^{\ell+-}(x) \right\| + \left\| Z^{\ell-+}(x) \right\| + \left\| Z^{\ell--}(x) \right\|}{\left\| Z^\ell(x) \right\|} \left\| Z^\ell(x) \right\| \\
&\leq \varepsilon^\ell \Delta^\ell \left\| Z^\ell(x) \right\|,
\end{aligned}$$

where the last step followed from our definition of Δ^ℓ. By imposing a regularity assumption on Δ^ℓ similar to that of ES, we can show that Δ^ℓ is an upper bound for any $x \sim \mathcal{D}$ w.h.p. following the proof of the ES inequality (Lemma 5).

To specify the appropriate notion of high probability, we consider the individual failure cases and apply the union bound. In particular, for our choice of for the size of $\mathcal{S}$, we have that for a

particular quadrant the approximation fails with probability at most $\delta/5$. Thus across all quadrants we have a overall failure probability of at most $4\delta/5$. Finally, we consider the event that Δ^ℓ does not upper bound the hardness for some input $x \sim \mathcal{D}$, which occurs with probability at most $\delta/5$ by our choice for the size of $\mathcal{S}$. Henceforth, our overall failure probability is at most δ, again by the union bound, which concludes the proof. $\qquad\square$

We now consider the effect of pruning multiple layers at the same time and analyze the final resulting error in the output. To this end, consider the activation $\phi^\ell(\cdot)$ for which we assume the following.

Assumption 2. *For layer $\ell \in [L]$, the activation function, denoted by $\phi^\ell(\cdot)$, is Lipschitz continuous with Lipschitz constant K^ℓ.*

Without loss of generality, we will further assume that the activation function is 1-Lipschitz, which is the case, e.g., for ReLU and Softmax, to avoid introducing additional notation. We now state a lemma pertaining to the error resulting from pruning multiple layers simultaneously, which will provide the basis for establishing error bounds across the entire network.

Lemma 12 (Error propagation). *Let $\hat{A}^\ell(x)$, $\ell \leq L$, denote the activation of layer ℓ when we have pruned layers $1, \ldots, \ell$ according to Lemma 11. Then the overall approximation in layer ℓ is bounded by*

$$\left\| \hat{A}^\ell(x) - A^\ell(x) \right\| \leq \sum_{k=1}^{\ell} \left(\prod_{k'=k+1}^{\ell} \left\| W^{k'} \right\|_F \right) \varepsilon^k \Delta^k \left\| Z^k(x) \right\|$$

with probability at least $1 - \delta$. The ESs are hereby computed over a set $\mathcal{S}$ of $K \log \left(5 \sum_{k \in [\ell]} \eta^k \, \rho/\delta \right)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. We prove the above statement by induction. For layer $\ell = 1$, we have that

$$
\begin{aligned}
\left\| \hat{A}^1(x) - A^1(x) \right\| &= \left\| \phi^1(\hat{W}^1 * \hat{A}^0(x)) - \phi^1(\hat{W}^1 * A^0(x)) \right\| \\
&\leq \left\| \hat{W}^1 * \hat{A}^0(x) - \hat{W}^1 * A^0(x) \right\| && \text{since the } \phi^1(\cdot) \text{ is 1-Lipschitz} \\
&= \left\| \hat{W}^1 * A^0(x) - \hat{W}^1 * A^0(x) \right\| && \text{since } \hat{A}^0(x) = A^0(x) = x \\
&= \left\| \hat{Z}^1(x) - Z^1(x) \right\| && \text{by definition of } \hat{Z}^1(x) \text{ and } Z^1(x) \\
&\leq \varepsilon^1 \Delta^1 \left\| Z^1(x) \right\| && \text{by Lemma 11,}
\end{aligned}
$$

which proves that the base case holds.

We now proceed with the inductive step. Assuming the inequality is true for layer ℓ, we have for layer $\ell + 1$ that

$$
\begin{aligned}
\left\| \hat{A}^{\ell+1}(x) - A^{\ell+1}(x) \right\| &= \left\| \phi^{\ell+1}(\hat{W}^{\ell+1} * \hat{A}^\ell(x)) - \phi^{\ell+1}(W^{\ell+1} * A^\ell(x)) \right\| \\
&\leq \left\| \hat{W}^{\ell+1} * \hat{A}^\ell(x) - W^{\ell+1} * A^\ell(x) \right\| && \text{since } \phi^{\ell+1}(\cdot) \text{ is 1-Lipschitz} \\
&= \left\| \hat{W}^{\ell+1} * \hat{A}^\ell(x) - \hat{W}^{\ell+1} * A^\ell(x) + \hat{W}^{\ell+1} * A^\ell(x) - W^{\ell+1} * A^\ell(x) \right\| \\
&\leq \left\| \hat{W}^{\ell+1} * (\hat{A}^\ell(x) - A^\ell(x)) \right\| + \left\| (\hat{W}^{\ell+1} - W^{\ell+1}) * A^\ell(x) \right\|
\end{aligned}
$$

30

Note that we can bound the first term by

$$\left\| \hat{W}^{\ell+1} * (\hat{A}^\ell(x) - A^\ell(x)) \right\| \leq \left\| \hat{W}^{\ell+1} \right\|_{op} \left\| \hat{A}^\ell(x) - A^\ell(x) \right\|$$

$$\leq \left\| \hat{W}^{\ell+1} \right\|_F \left\| \hat{A}^\ell(x) - A^\ell(x) \right\|$$

$$\leq \left\| W^{\ell+1} \right\|_F \left\| \hat{A}^\ell(x) - A^\ell(x) \right\| \qquad \text{since } \hat{W}^{\ell+1} \text{ is a subset of } W^{\ell+1}$$

where $\|\cdot\|_{op}$ and $\|\cdot\|_F$ denote the ℓ_2-induced operator norm and Frobenius norm, respectively. The second term is bounded by Lemma 11, i.e.,

$$\left\| (\hat{W}^{\ell+1} - W^{\ell+1}) * A^\ell(x) \right\| = \left\| \hat{Z}^{\ell+1}(x) - Z^{\ell+1}(x) \right\| \leq \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\|.$$

Putting both terms back together we have that

$$\left\| \hat{A}^{\ell+1}(x) - A^{\ell+1}(x) \right\|$$

$$\leq \left\| W^{\ell+1} \right\|_F \left\| \hat{A}^\ell(x) - A^\ell(x) \right\| + \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\|$$

$$\leq \left\| W^{\ell+1} \right\|_F \left(\sum_{k=1}^{\ell} \left(\prod_{k'=k+1}^{\ell} \left\| W^{k'} \right\|_F \right) \varepsilon^k \Delta^k \left\| Z^k(x) \right\| \right) + \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\|$$

$$= \sum_{k=1}^{\ell} \left(\prod_{k'=k+1}^{\ell+1} \left\| W^{k'} \right\|_F \right) \varepsilon^k \Delta^k \left\| Z^k(x) \right\| + \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\|$$

$$= \sum_{k=1}^{\ell+1} \left(\prod_{k'=k+1}^{\ell+1} \left\| W^{k'} \right\|_F \right) \varepsilon^k \Delta^k \left\| Z^k(x) \right\|,$$

where the second inequality followed from our induction hypothesis. Finally, we note that, by our choice for the size of $\mathcal{S}$ and the union bound, the overall failure probability is bounded above by δ.
$\square$

From the analysis the term $\prod_{k'=k+1}^{\ell} \left\| W^{k'} \right\|_F$ arises, which is an upper bound for the Lipschitz constant of the network starting from layer $k+1$. Moreover, the coefficient of the propagated error is closely related to the condition number between layer ℓ and the network's output. To this end, consider the following upper bound on the condition number.

Definition 5 (Layer condition number). *For layer ℓ, the condition number from the pre-activation of layer ℓ to the output of the network (activation of layer L) is given by*

$$\kappa^\ell = \max_{x \in S} \left(\prod_{k=\ell+1}^{L} \left\| W^\ell \right\|_F \right) \frac{\|Z^\ell(x)\|}{\|A^L(x)\|},$$

where $\mathcal{S}$ denotes a set of i.i.d. data points drawn from $\mathcal{D}$.

To see that κ^ℓ is indeed an upper bound on the condition number we note that the condition number is defined as the maximum relative change in the output over the maximum relative change

in the input, i.e.,

$$\max_{x,x' \in \mathcal{S}} \frac{\frac{\left\| A^L(x) - A^L(x') \right\|}{\left\| A^L(x) \right\|}}{\frac{\left\| Z^\ell(x) - Z^\ell(x') \right\|}{\left\| Z^\ell(x) \right\|}} = \max_{x,x' \in \mathcal{S}} \frac{\left\| A^L(x) - A^L(x') \right\|}{\left\| Z^\ell(x) - Z^\ell(x') \right\|} \frac{\left\| Z^\ell(x) \right\|}{\left\| A^L(x) \right\|}.$$

The first term can be upper bounded as

$$
\begin{aligned}
\frac{\left\| A^L(x) - A^L(x') \right\|}{\left\| Z^\ell(x) - Z^\ell(x') \right\|} &\leq \frac{\left\| Z^L(x) - Z^L(x') \right\|}{\left\| Z^\ell(x) - Z^\ell(x') \right\|} \\
&= \frac{\left\| W^L * (A^{L-1}(x) - A^{L-1}(x')) \right\|}{\left\| Z^\ell(x) - Z^\ell(x') \right\|} \\
&\leq \left\| W^L \right\|_F \frac{\left\| (A^{L-1}(x) - A^{L-1}(x')) \right\|}{\left\| Z^\ell(x) - Z^\ell(x') \right\|} \\
&\leq \dots \\
&\leq \prod_{k=\ell+1}^{L} \left\| W^k \right\|_F \frac{\left\| A^\ell(x) - A^\ell(x') \right\|}{\left\| Z^\ell(x) - Z^\ell(x') \right\|} \\
&\leq \prod_{k=\ell+1}^{L} \left\| W^k \right\|_F,
\end{aligned}
$$

which plugged back in above yields the definition of the layer condition number κ^ℓ.

Equipped with Lemma 12 and Definition 5 we are now ready to state our main compression bound over the entire network.

Theorem 13 (Network compression bound). *For given $\delta \in (0,1)$, a set of parameters $\theta = (W^1, \dots, W^L)$, and a sample budget $\mathcal{B}$ SiPP (Algorithm 2) generates a set of compressed parameters $\hat{\theta} = (\hat{W}^1, \dots, \hat{W}^L)$ such that $\|\hat{\theta}\|_0 \leq \mathcal{B}$, $\|W_i^\ell\|_0 \leq N_i^\ell$, $\forall i \in [c^\ell]$, $\ell \in [L]$,*

$$\mathop{\mathbb{P}}_{\hat{\theta},x} \left(\left\| f_{\hat{\theta}}(x) - f_\theta(x) \right\| \leq \varepsilon \left\| f_\theta(x) \right\| \right) \geq 1 - \delta \quad \text{and} \quad \varepsilon = C \sum_{\ell=1}^{L} \kappa^\ell \Delta^\ell \max_{i \in [c^\ell]} \left(S_i^\ell - S_i^\ell(N_i^\ell) \right),$$

where S_i^ℓ is the sum of sensitivities for parameter group W_i^ℓ computed over a set $\mathcal{S}$ of $K \log(6\eta\rho/\delta)$ i.i.d. data points.

Proof. Invoking Lemma 12 for $\ell = L$ implies with high probability that

$$
\begin{aligned}
\left\| \hat{A}^L(x) - A^L(x) \right\| &\leq \sum_{\ell=1}^{L} \left(\prod_{k=\ell+1}^{L} \left\| W^k \right\|_F \right) \varepsilon^\ell \Delta^\ell \left\| Z^\ell(x) \right\| \\
&= \sum_{\ell=1}^{L} \left(\prod_{k=\ell+1}^{L} \left\| W^k \right\|_F \right) \frac{\left\| Z^\ell(x) \right\|}{\left\| A^L(x) \right\|} \Delta^\ell \varepsilon^\ell \left\| A^L(x) \right\| \\
&\leq \sum_{\ell=1}^{L} \kappa^\ell \Delta^\ell \varepsilon^\ell \left\| A^L(x) \right\|
\end{aligned}
$$

32

$$= \varepsilon \left\| A^L(x) \right\|,$$

where the last inequality followed from our definition of the layer condition number κ^ℓ. Moreover, following the analysis of Lemma 5 we can establish that κ^ℓ is an upper bound for any $x \sim \mathcal{D}$ with high probability. Finally, we note that the overall failure probability is bounded by δ by our choice for the size of $\mathcal{S}$ and by a union bound over the failure probabilities of Lemma 12 and of κ^ℓ not being an upper bound for some $x \sim \mathcal{D}$. $\qquad\square$

E   Experimental details

E.1   Setup and Hyperparameters

All hyperparameters for training, retraining, and pruning are outlined in Table S1. For training CIFAR10 networks we used the training hyperparameters outlined in the respective original papers, i.e., as described by [18], [41], [19], and [52] for ResNets, VGGs, DenseNets, and WideResNets, respectively. For retraining, we did not change the hyperparameters and repurposed the training hyperparameters. We added a warmup period in the beginning where we linearly scale up the learning rate from 0 to the nominal learning rate. Iterative pruning is conducted by repeatedly removing the same ratio of parameters (denoted by α in Table S1). The prune parameter δ describes the failure probability of SiPP. We note no other additional hyperparameters are required to run SiPP.

For ImageNet, we show experimental results for a ResNet18 and a ResNet101. As in the case of the CIFAR10 networks, we re-purpose the same training hyperparameters as indicated in the original paper. We also use the same hyperparameters for retraining. The hyperparameters are summarized in Table S2.

		VGG16	Resnet20/56/110	DenseNet22	WRN-16-8
	test error	7.19	8.6/7.19/6.43	10.10	4.81
	loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy
	optimizer	SGD	SGD	SGD	SGD
	epochs	300	182	300	200
Train	warm-up	5	5	5	5
	batch size	256	128	64	128
	LR	0.05	0.1	0.1	0.1
	LR decay	0.5@{30, …}	0.1@{91, 136}	0.1@{150, 225}	0.2@{60, …}
	momentum	0.9	0.9	0.9	0.9
	Nesterov	No	No	Yes	Yes
	weight decay	5.0e-4	1.0e-4	1.0e-4	5.0e-4
Prune	δ	1.0e-16	1.0e-16	1.0e-16	1.0e-16
	α	0.85	0.85	0.85	0.85

Table S1: We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on CIFAR-10. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. During retraining we used the same hyperparameters. {30,…} denotes that the learning rate is decayed every 30 epochs.

		ResNet18/101
	top-1 test error	30.26/22.63
	top-5 test error	10.93/6.45
	loss	cross-entropy
	optimizer	SGD
	epochs	90
Train	warm-up	5
	batch size	256
	LR	0.1
	LR decay	0.1@{30, 60, 80}
	momentum	0.9
	Nesterov	No
	weight decay	1.0e-4
Prune	δ	1.0e-16
	α	0.90

Table S2: We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on ImageNet. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs.

E.2   Iterative prune+retrain results for CIFAR10

In Figure S2 we show the results and comparisons when using iterative prune+retrain as outlined in Section 6. We highlight that SiPP performs en par with WT while SNIP performs significantly worse than WT.

 We also compare the performance of the three variations of our algorithm, see Figure S1. Note that the performance for all of them is very similar, henceforth we choose SiPPDet for its simplicity when comparing to other methods for this expensive iterative prune+retrain pipeline.

E.3   Random-init+prune+train results for CIFAR10

The results for this pipeline are shown in Figure S3. We note that WT performs significantly worse in this case whereas SNIP and SiPP clearly outperform WT. Overall, we find that sometimes SiPP can even outperform SNIP. More importantly, however, these experiments highlight the versatile nature of SiPP, i.e., it performs consistently well across multiple prune pipelines hence serving as a reliable and useful plug-and-play solution within a bigger pipeline. We conjecture that this is due to the provable nature of SiPP.

E.4   Iterative prune+retrain results for ImageNet

Finally, we show results for a ResNet18 and ResNet101 trained on ImageNet, see Figure S4. From the results, we can conclude that SiPP scales well to larger architectures and datasets, such as ImageNet, and can perform en par with existing state-of-the-art methods.

(a) Resnet20   (b) Resnet56

Figure S1: The delta in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR10 for various target prune ratios. The networks were pruned using the **iterative prune+retrain** pipeline.

(a) Resnet20

(b) Resnet56

(c) Resnet110

(d) VGG16

(e) DenseNet22

(f) WRN16-8

Figure S2: The delta in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR10 for various target prune ratios. The networks were pruned using the **iterative prune+retrain** pipeline.

(a) Resnet20

(b) Resnet56

(c) Resnet110

(d) VGG16

(e) DenseNet22

(f) WRN16-8

Figure S3: The delta in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR10 for various target prune ratios. The networks were pruned using the **random-init+ prune+train** pipeline.

(a) ResNet18, Top 1

(b) ResNet18, Top 5

(c) ResNet101, Top 1

(d) ResNet101, Top 5

Figure S4: The accuracy of the generated pruned **ResNet18** and **ResNet101** models trained on ImageNet for the evaluated pruning schemes for various target prune ratios.