Data-Efficient Structured Pruning via Submodular Optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

Structured pruning is an effective approach for compressing large pre-trained neural networks without significantly affecting their performance, which involves removing redundant regular regions of weights. However, current structured pruning methods do not provide any performance guarantees, and often require fine-tuning, which makes them inapplicable in the limited-data regime. We propose a principled data-efficient structured pruning method based on submodular optimization. In particular, for a given layer, we select neurons/channels to prune and corresponding new weights for the next layer, that minimize the change in the next layer's input induced by pruning. We show that this selection problem is a weakly submodular maximization problem, thus it can be provably approximated using an efficient greedy algorithm. Our method is one of the few in the literature that uses only a limited-number of training data and no labels. Our experimental results demonstrate that our method outperforms state-of-the-art methods in the limited-data regime.

1 Introduction

As modern neural networks (NN) grow increasingly large, with some models reaching billions of parameters [McGuffie and Newhouse, 2020], they require an increasingly large amount of memory, power, hardware, and inference time, which makes it necessary to compress them. This is especially important for models deployed on resource-constrained devices like mobile phones and smart speakers, and for latency-critical applications such as self-driving cars.

Several approaches exist to compress NNs. Some methods approximate model weights using quantization and hashing [Gong et al., 2014, Courbariaux et al., 2015], or low-rank approximation and tensor factorization [Denil et al., 2013, Lebedev et al., 2015, Su et al., 2018]. In another class of methods called knowledge distillation, a small network is trained to mimic a much larger network [Bucila et al., 2006, Hinton et al., 2015]. Other methods employ sparsity and group-sparsity regularisation during training, to induce sparse weights [Collins and Kohli, 2014, Voita et al., 2019].

In this work, we follow the network pruning approach, where the redundant units (weights, neurons or filters/channels) of a pre-trained NN are removed; see [Kuzmin et al., 2019, Blalock et al., 2020, Hoefler et al., 2021] for recent surveys. We also focus on the limited-data regime, where only few training data is available and data labels are unavailable. The advantage of pruning approaches is that, unlike weights approximation-based methods, they preserve the network structure, allowing retraining after compression, and unlike training-based approaches, they do not require training from scratch, which is costly and requires large training data. It is also possible to combine different compression approaches to compound their benefits, see e.g., [Kuzmin et al., 2019, Section 4.3.4].

Pruning methods date back to at least the 1990s with the works of [LeCun et al., 1990, Hassibi et al., 1993]. Existing pruning methods fall into two main categories: unstructured pruning methods which

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

prune individual weights leading to irregular sparsity patterns, and structured pruning methods which prune regular regions of weights, such as neurons, channels, or attention heads. Structured pruning methods are generally preferable as the resulting pruned models can work with off-the-shelf hardware or kernels, as opposed to models pruned with unstructured pruning which require specialized ones.

The majority of existing structured pruning methods are heuristics that do not offer any theoretical guarantees. Moreover, most pruning methods are inapplicable in the limited-data regime, as they rely on fine-tuning with large training data at least for a few epochs to recover some of the accuracy lost with pruning. Mariet and Sra [2015] proposed a "reweighting" procedure applicable to any pruning method, optimize the weights of the next layer to minimize the change in the input to the next layer Their empirical results on pruning single linear layers suggest that reweighting can provide a similar boost to performance as fine-tuning, without the need for data labels.

Our contributions We propose a principled data-efficient structured pruning method based on submodular optimization. In each layer, our method simultaneously selects neurons to prune and new weights for the next layer, that minimize the change in the next layer's input induced by pruning. The optimization with respect to the weights, for a fixed selection of neurons, is the same one used for reweighting in [Mariet and Sra, 2015]. The resulting subset selection problem is intractable, but we show that it can be formulated as a weakly submodular maximization problem (see definition in Section 2). This formulation allows us to use the standard greedy algorithm to obtain $(1 - e^{-\gamma})$-approximation to the optimal solution, where γ is non-zero if we use sufficient training data.

Our method uses only limited training data and no labels. Similar to [Mariet and Sra, 2015], we observe that reweighting provide a significant boost in performance not only to our method, but also to other baselines we consider. However unlike [Mariet and Sra, 2015], we only use a small fraction of the training data, around $\sim 1\%$ in our experiments. We further adapt our method to prune any regular regions of weights; we focus in particular on pruning channels in convolution layers. To prune multiple layers in the network, we apply our method to each layer independently or sequentially. Our experimental results demonstrate that our method outperforms state-of-the-art pruning methods in the limited-data regime, and it is among the best performing methods even in the full-data setting.

Related work A large variety of structured pruning approaches has been proposed in the literature, based on different selection schemes and algorithms to solve them. Ye et al. [2020] propose to prune neurons/channels that minimize the loss of the pruned network, using a layerwise Greedy forward selection method, with Frank-Wolfe like updates. In the special case of two layer NNs, the square loss of the resulting pruned network approximates the best loss that can be achieved with all neurons in the hidden layer, under some regularity conditions. No guarantees are provided for general NNs. This method is expensive as it requires a forward pass to evaluate every neuron/channel at each iteration.

Some works prune neurons/channels individually based on some importance score [He et al., 2014, Li et al., 2017, Liebenwein et al., 2020, Mussay et al., 2020, 2021, Molchanov et al., 2017, Srinivas and Babu, 2015]. Such methods are efficient and easy to implement, but they fail to capture higher-order interactions between the pruned parameters. Most do not provide any performance guarantee. One exception are the sampling-based methods of [Liebenwein et al., 2020, Mussay et al., 2020, 2021], who provide high probability bounds on the difference between the output of the pruned and original model, as a function of number of neurons/channels kept, under some assumptions on the model activations. These theoretical guarantees are not comparable to the ones we provide (see Section 3.1 for details). Empirically, we show that our method significantly outperforms [Liebenwein et al., 2020].

Closer to our approach are methods that aim to prune neurons/channels that minimize the change induced by pruning in the output of the layer being pruned, or its input to the next layer [Luo et al., 2017, He et al., 2017, Zhuang et al., 2018]. Although these criteria are easier to optimize than the loss or accuracy of the NN, they still yield an intractable combinatorial problem. Existing methods either use a heuristic greedy algorithm to solve it [Luo et al., 2017, Zhuang et al., 2018], or they solve instead its ℓ_1-relaxation using alternating minimization [He et al., 2017].

Mariet and Sra [2015] depart from the usual strategy of pruning parameters whose removal influences the network the least. They instead select a subset of diverse neurons to keep in each layer by sampling from a Determinantal Point Process, then they apply their reweighting procedure. Though, their experimental results show that the advantage of their method is mostly due to reweighting.

2 Preliminaries

We begin by introducing our notation and some relevant background from submodular optimization.

Notation: Given a ground set $V = \{1, 2, \cdots, d\}$ and a set function $F : 2^V \to \mathbb{R}_+$, we denote the *marginal gain* of adding a set $I \subseteq V$ to another set $S \subseteq V$ by $F(I \mid S) = F(S \cup I) - F(S)$, which quantifies the change in value when adding I to S. The cardinality of a set S is written as $|S|$. Given a vector $x \in \mathbb{R}^d$, we denote its support set by $\text{supp}(x) = \{i \in V | x_i \neq 0\}$, and its ℓ_2-norm by $\|x\|_2$. Given a matrix $X \in \mathbb{R}^{d' \times d}$, we denote its i-th column by X_i, and its Frobenius norm by $\|X\|_F$. Given a set $S \subseteq V$, X_S is the matrix with columns X_i for all $i \in S$, and 0 otherwise, and $\mathbf{1}_S$ is the indicator vector of S, with $[\mathbf{1}_S]_i = 1$ for all $i \in S$, and 0 otherwise.

Algorithm 1 GREEDY

1: **Input:** Ground set V, set function $F : 2^V \to \mathbb{R}_+$, budget $k \in \mathbb{N}_+$
2: $S \leftarrow \emptyset$
3: **while** $|S| < k$ **do**
4: $\quad i^* \leftarrow \arg\max_{i \in V \setminus S} F(i \mid S)$
5: $\quad S \leftarrow S \cup \{i^*\}$
6: **end while**
7: **Output:** S

Weakly submodular maximization: A set function F is *submodular* if it has diminishing marginal gains: $F(i \mid S) \geq F(i \mid T)$ for all $S \subseteq T$, $i \in V \setminus T$. We say that F is *normalized* if $F(\emptyset) = 0$, and non-decreasing if $F(S) \leq F(T)$ for all $S \subseteq T$.

Given a non-decreasing submodular function F, selecting a set $S \subseteq V$ with cardinality $|S| \leq k$ that maximize $F(S)$ can be done efficiently using the GREEDY algorithm (Alg. 1). The returned solution is guaranteed to satisfy $F(\hat{S}) \geq (1 - 1/e) \max_{|S| \leq k} F(S)$ [Nemhauser et al., 1978]. In general though maximizing a non-submodular function over a cardinality constraint is NP-Hard [Natarajan, 1995]. However, Das and Kempe [2011] introduced a notion of *weak submodularity* which is sufficient to obtain provable bounds of the GREEDY algorithm's performance.

Definition 2.1. Given a set function $F : 2^V \to \mathbb{R}$, $U \subseteq V, k \in \mathbb{N}_+$, we say that F is $\gamma_{U,k}$-weakly submodular, with $\gamma_{U,k} > 0$ if

$$\gamma_{U,k} F(S|L) \leq \sum_{i \in S} F(i|L),$$

for every two disjoint sets $L, S \subseteq V$, such that $L \subseteq U, |S| \leq k$.

The parameter $\gamma_{U,k}$ is called the *submodularity ratio* of F. It characterizes how close a set function is to being submodular. If F is non-decreasing then $\gamma_{U,k} \in [0, 1]$, and F is submodular if and only if $\gamma_{U,k} = 1$ for all $U \subseteq V, k \in \mathbb{N}_+$. Given a non-decreasing $\gamma_{\hat{S},k}$-weakly submodular function F, the Greedy algorithm is guaranteed to return a solution $F(\hat{S}) \geq (1 - e^{-\gamma_{\hat{S},k}}) \max_{|S| \leq k} F(S)$ [Das and Kempe, 2011]. Hence, the closer F is to being submodular, the better is the approximation guarantee.

3 Reweighted input change pruning

In this section, we introduce our approach for pruning neurons in a single layer. Given a large pre-trained NN, n training data samples, and a layer ℓ with n_ℓ neurons, our goal is to select a small number k out of the n_ℓ neurons to keep, and prune the rest, in a way that influences the network the least. One way to achieve this is by minimizing the change in input to the next layer, $\ell + 1$, induced by pruning. However, simply throwing away the activations from the dropped neurons is wasteful. Instead, we optimize the weights of the next layer to reconstruct the inputs from the remaining neurons.

Formally, let $A^\ell \in \mathbb{R}^{n \times n_\ell}$ be the activation matrix of layer ℓ with columns $a_1^\ell, \cdots, a_{n_\ell}^\ell$, where $a_i^\ell \in \mathbb{R}^n$ is the vector of activations of the ith neuron in layer ℓ for each training input, and let $W^{\ell+1} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ be the weight matrix of layer $\ell + 1$ with columns $w_1^{\ell+1}, \cdots, w_{n_{\ell+1}}^{\ell+1}$, where $w_i^{\ell+1} \in \mathbb{R}^{n_\ell}$ is the vector of weights connecting the ith neuron in layer $\ell + 1$ to the neurons in layer ℓ.

When a neuron is pruned in layer ℓ, the corresponding column of weights in W^ℓ and row in $W^{\ell+1}$ are removed. Pruning $n_\ell - k$ neurons in layer ℓ reduces the number of parameters and computation cost by $(n_\ell - k)/n_\ell$ for both layer ℓ and $\ell + 1$.

Let $V_\ell = \{1, \cdots, n_\ell\}$. Given a set $S \subseteq V_\ell$, we denote by A_S^ℓ the matrix with columns a_i^ℓ for all $i \in S$, and 0 otherwise. That is, A_S^ℓ is the activation matrix of layer ℓ after pruning. We choose a set of neurons $S \subseteq V_\ell$ to keep and new weights $\tilde{W}^{\ell+1} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ that minimize:

$$\min_{|S| \leq k, \tilde{W}^{\ell+1} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}} \|A^\ell W^{\ell+1} - A_S^\ell \tilde{W}^{\ell+1}\|_F^2 \tag{1}$$

Note that $A^\ell W^{\ell+1}$ are the original inputs of layer $l + 1$, and $A_S^\ell \tilde{W}^{\ell+1}$ are the inputs after pruning and reweighting, i.e., replacing the weights $W^{\ell+1}$ of layer $\ell + 1$ with the new weights $\tilde{W}^{\ell+1}$.

3.1 Greedy selection

Solving Problem (1) exactly is NP-Hard [Natarajan, 1995]. However, we show below that it can be formulated as a weakly submodular maximization problem, hence it can be efficiently approximated. Let

$$F(S) = \|A^\ell W^{\ell+1}\|_F^2 - \min_{\tilde{W}^{\ell+1}} \|A^\ell W^{\ell+1} - A_S^\ell \tilde{W}^{\ell+1}\|_F^2, \tag{2}$$

then Problem (1) is equivalent to $\max_{|S| \leq k} F(S)$.

Proposition 3.1. *Given $U \subseteq V$, $k \in \mathbb{N}_+$, F is a normalized non-decreasing $\gamma_{U,k}$-weakly submodular function, with*

$$\gamma_{U,k} \geq \frac{\min_{\|z\|_2=1, \|z\|_0 \leq |U|+k} \|A^\ell z\|_2^2}{\max_{\|z\|_2=1, \|z\|_0 \leq |U|+1} \|A^\ell z\|_2^2}.$$

The proof of Proposition 3.1 follows by writing F as the sum of $n_{\ell+1}$ sparse linear regression problems $F(S) = \sum_{m=1}^{n_{\ell+1}} \|A^\ell w_m^{\ell+1}\|_2^2 - \min_{\text{supp}(\tilde{w}_m) \subseteq S} \|A^\ell w_m^{\ell+1} - A^\ell \tilde{w}_m\|_2^2$, and from the relation established in [Elenberg et al., 2016, Das and Kempe, 2011] between weak submodularity and sparse eigenvalues of the covariance matrix (see Appendix B.1).

We use the GREEDY algorithm to select a set $\hat{S} \subseteq V_\ell$ of k neurons to keep in layer ℓ. As discussed in Section 2, the returned solution is guaranteed to satisfy

$$F(\hat{S}) \geq (1 - e^{-\gamma_{\hat{S},k}}) \max_{|S| \leq k} F(S) \tag{3}$$

The submodularity ratio $\gamma_{\hat{S},k}$ is non-zero if any $2k$ columns of A^ℓ are linearly independent. If the number of training data is larger than the number of neurons, i.e., $n > n_\ell$, this is likely to be satisfied for any $k \leq n_\ell/2$; it is unlikely for the activations of a neuron to match exactly a linear combination of other neurons. We verify that this is indeed the case in our experiments.

We show in Appendix C that F satisfies an even stronger notion of approximate submodularity than weak submodularity, which implies a better approximation guarantee for GREEDY than the one provided in Eq. (3). Though, this requires a somewhat stronger assumption: any $k + 1$ columns of A^ℓ should be linearly independent and all rows of $W^{\ell+1}$ should be linearly independent. In particular, we would need that $n_\ell \leq n_{\ell+1}$, which is not always satisfied.

Note that our performance guarantees are of a different kind than the ones provided in [Liebenwein et al., 2020, Mussay et al., 2020, 2021]; they provide high probability bounds of the form $A_S^\ell \tilde{W}^{\ell+1} \in (1 \pm \epsilon) A^\ell W^{\ell+1}$ (entry-wise), as a function of the number of neurons kept k, which they propagate to obtain bounds on the difference between the output of the pruned and original model.

3.2 Reweighting

For a fixed $S \subseteq V_\ell$, the reweighted input change $\|A^\ell W^{\ell+1} - A_S^\ell \tilde{W}^{\ell+1}\|_F^2$ is minimized by setting

$$\tilde{W}^{\ell+1} = x^S(A^\ell) W^{\ell+1}, \tag{4}$$

where $x^S(A^\ell) \in \mathbb{R}^{n_\ell \times n_\ell}$ is the matrix with columns $x^S(a_j^\ell)$ such that

$$x^S(a_j^\ell) \in \operatorname*{arg\,min}_{\text{supp}(x) \subseteq S} \|a_j^\ell - Ax\|_2^2 \text{ for all } j \in V_\ell. \tag{5}$$

Note that $x^S(a_j) = \mathbf{1}_j$ for all $j \in S$, and that $x^S(A^\ell)$ has zero rows for all $i \notin S$. Hence, the new weights are given by $\tilde{w}_{im}^{\ell+1} = w_{im}^{\ell+1} + \sum_{j \notin S}[x^S(A^\ell)]_{ij} w_{jm}^{\ell+1}$ for all $i \in S$, and $\tilde{w}_{im}^{\ell+1} = 0$ for all $i \notin S, m \in V_{\ell+1}$. Namely, the new weights merge the weights from the dropped neurons into the selected ones. This is the same reweighting procedure introduced in [Mariet and Sra, 2015]. But instead of applying it only at the end to the selected neurons $\hat{S}$, it is implicitly done at each iteration of our pruning method, as it is required to evaluate F. We discuss next how this can be done efficiently.

3.3 Cost

Each iteration of GREEDY requires $O(n_\ell)$ function evaluations of F. Computing $F(S)$ from scratch needs $O(k \cdot (n_\ell \cdot n_{\ell+1} + n \cdot (n_\ell + n_{\ell+1}))$ time, so a naive implementation of GREEDY is too expensive. The following Proposition outlines how we can efficiently evaluate $F(S+i)$ given that $F(S)$ was computed in the previous iteration.

Proposition 3.2. *Given $S \subseteq V_\ell$ such that $|S| \leq k$, $i \notin S$, let $\mathrm{proj}_S(a_j^\ell) = A_S^\ell x^S(a_j^\ell)$ be the projection of a_j^ℓ onto the column space of A_S^ℓ, $R_S(a_i^\ell) = a_i^\ell - \mathrm{proj}_S(a_i^\ell)$ and $\mathrm{proj}_{R_S(a_i^\ell)}(a_j^\ell) \in \arg\min_{z = R_S(a_i^\ell)\gamma, \gamma \in \mathbb{R}} \|a_j^\ell - z\|_2^2$ the corresponding residual and the projection of a_j^ℓ onto it. We can write*

$$F(i|S) = \sum_{m=1}^{n^{\ell+1}} \|\mathrm{proj}_{R_S(a_i^\ell)}(A_{V \setminus S}^\ell) w_m^{\ell+1}\|_2^2,$$

where $\mathrm{proj}_{R_S(a_i)}(A_{V \setminus S}^\ell)$ is the matrix with columns $\mathrm{proj}_S(a_j^\ell)$ for all $j \notin S$, 0 otherwise. Assuming $F(S), \mathrm{proj}_S(a_j^\ell)$ and $x^S(a_j^\ell)$ for all $j \notin S$ were computed in the previous iteration, we can compute $F(S+i), \mathrm{proj}_{S+i}(a_j^\ell)$ and $x^{S+i}(a_j^\ell)$ for all $j \notin (S+i)$ in

$$O(n_\ell \cdot (n_{\ell+1} + n + k)) \text{ time.}$$

The optimal weights in Eq. (4) can then be computed in $O(k \cdot n_\ell \cdot n_{\ell+1})$ time, at the end of GREEDY.

The proof is given in Appendix B.2, and relies on using optimality conditions to construct the least squares solution $x^{S+i}(a_j^\ell)$ from $x^S(a_j^\ell)$.

In total GREEDY's runtime is then $O(k \cdot (n_\ell)^2 \cdot (n_{\ell+1} + n + k))$. In other words, our pruning method costs as much as $O(k)$ forward passes in layer $\ell+1$ with a batch size of n (assuming $n^{\ell+1} = O(n^\ell)$). Using a faster variant of GREEDY proposed in [Li et al., 2022] further reduces the cost to $O((n_\ell)^2 \cdot (n_{\ell+1} + n + k)/\epsilon)$ while maintaining almost the same approximation guarantee $(1 - e^{-\gamma_{\hat{S},k}} - \epsilon)$. [1] Note also that computing the solutions for different budgets $k' \leq k$ can be done at the cost of one by running GREEDY with budget k. Our method is more expensive than methods which prune neurons individually [He et al., 2014, Li et al., 2017, Liebenwein et al., 2020, Mussay et al., 2020, 2021, Molchanov et al., 2017, Srinivas and Babu, 2015], but much less expensive than a loss-based method like [Ye et al., 2020], which requires $O(kn^\ell)$ forward passes in the full network, for each layer.

4 Pruning regular regions of neurons

In this section, we discuss how to adapt our approach to pruning regular regions of neurons. This is easily achieved by mapping any set of regular regions to the corresponding set of neurons, then applying the same method in Section 3. In particular, we focus on pruning channels in CNNs.

Given a layer ℓ with n_ℓ output channels, let $X^\ell \in \mathbb{R}^{n \cdot p_\ell \times n_\ell \times r_h \times r_w}$ be its activations for each output channel and training input, where p_ℓ is number of patches obtained by applying a filter of size $r_h \times r_w$, and let $F^{\ell+1} \in \mathbb{R}^{n_{\ell+1} \times n_\ell \times r_h \times r_w}$ be the weights of layer $\ell+1$, corresponding to n_ℓ filters of size $r_h \times r_w$ for each of its output channels.

When an output channel is pruned in layer ℓ, the corresponding weights in F^ℓ and $F^{\ell+1}$ are removed. Pruning $n^\ell - k$ output channels in layer ℓ reduces the number of parameters and computation cost by $(n^\ell - k)/n^\ell$ for both layer ℓ and $\ell+1$. If layer ℓ is followed by a batch norm layer, the weights therein corresponding to the pruned channels are also removed.

[1] The results in [Li et al., 2022] only apply to submodular functions, but it is straighforward to extend them to weakly submodular functions.

We arrange the activations $X_c^\ell \in \mathbb{R}^{n \cdot p_\ell \times r_h \cdot r_w}$ of each channel c into $r_h r_w$ columns of $A^\ell \in \mathbb{R}^{n \cdot p_\ell \times n_\ell \cdot r_h \cdot r_w}$, i.e., $A^\ell = [X_1^\ell, \cdots, X_{n_\ell}^\ell]$. Similarly, we arrange the weights $F_c^{\ell+1} \in \mathbb{R}^{n_{\ell+1} \times r_h \times r_w}$ of each channel c into $r_h \cdot r_w$ rows of $W^{\ell+1} \in \mathbb{R}^{n_\ell \cdot r_h \cdot r_w \times n_{\ell+1}}$, i.e., $(W^{\ell+1})^\top = [(F_1^\ell)^\top, \cdots, (F_{n_\ell}^\ell)^\top]$. Recall that $V_\ell = \{1, \cdots, n_\ell\}$, and let $V_\ell' = \{1, \cdots, r_h r_w n_\ell\}$. We define a function $M : 2^{V_\ell} \to 2^{V_\ell'}$ which maps every channel c to its corresponding $r_h r_w$ columns A^ℓ. Let $G(S) = F(M(S))$, with F defined in Eq. (2), then minimizing the reweighted input change $\|A^\ell W^{\ell+1} - A_{M(S)}^\ell \tilde{W}^{\ell+1}\|_F^2$ with a budget k is equivalent to $\max_{|S| \leq k} G(S)$. The following proposition shows that this remains a weakly submodular maximization problem.

Proposition 4.1. *Given $U \subseteq V_\ell$, $k \in \mathbb{N}_+$, G is a normalized non-decreasing $\gamma_{U,k}$-weakly submodular function, with*

$$\gamma_{U,k} \geq \frac{\min_{\|z\|_2=1, \|z\|_0 \leq r_h r_w (|U|+k)} \|A^\ell z\|_2^2}{\max_{\|z\|_2=1, \|z\|_0 \leq r_h r_w (|U|+1)} \|A^\ell z\|_2^2}.$$

Proof sketch. G is $\gamma_{U,k}$-weakly submodular iff F satisfies

$$\gamma_{U,k} F(M(S)|M(L)) \leq \sum_{i \in S} F(M(i)|M(L)),$$

for every two disjoint sets $L, S \subseteq V_\ell$, such that $L \subseteq U, |S| \leq k$. We extend the relation established in [Elenberg et al., 2016, Das and Kempe, 2011] between weak submodularity and sparse eigenvalues of the covariance matrix to this case. The proof then follows in a similar way to Proposition 3.1. □

As before, we use the GREEDY algorithm, with function G, to select a set $\hat{S} \subseteq V_\ell$ of k channels to keep in layer ℓ. We get the same approximation guarantee $G(\hat{S}) \geq (1 - e^{-\gamma_{\hat{S},k}}) \max_{|S| \leq k} G(S)$. The submodularity ratio $\gamma_{\hat{S},k}$ is non-zero if any $2k r_h r_w$ columns of A^ℓ are linearly independent. In our experiments, we observe that linear independence only holds for small $k \leq 0.2 n_\ell$ in certain layers. This is due to the correlation between patches which overlap. To remedy that, we experimented with using only $r_h r_w$ random patches from each image, instead of using all patches. This indeed raises the rank of A^ℓ, such that the range where linear independence is satisfied is around $k \leq 0.4 n_\ell$. However, the obtained results are very similar to the ones with all patches, we thus omit them. Note that having linear dependence does not necessarily imply that $\gamma_{\hat{S},k} = 0$; our method still performs well in these cases.

For a fixed $S \subseteq V_\ell$, the optimal weights are again given by $\tilde{W}^{\ell+1} = x^{M(S)}(A^\ell) W^{\ell+1}$. The cost of running GREEDY and reweighting is the same as before (see Appendix B.2).

5 Pruning multiple layers

In this section, we explain how to apply our pruning method to prune multiple layers of a NN.

5.1 Reweighted input change pruning variants

We consider three variants of our method: LAYERINCHANGE, SEQINCHANGE, and ASYMINCHANGE. In LAYERINCHANGE, we prune each layer independently, i.e., we apply exactly the method in Section 3 or 4, according to the layer's type. This is the fastest variant; it has the same cost as pruning a single layer, as each layer can be pruned in parallel, and it only requires one forward pass to get the activations of all layers. However, it does not take into account the effect of pruning one layer on subsequent layers.

In SEQINCHANGE, we prune each layer sequentially, starting from the earliest layer to latest one. For each layer ℓ, we apply our method with A^ℓ replaced by the *updated* activations B^ℓ after having pruned previous layers, i.e., we solve $\min_{|S| \leq k, \tilde{W}^{\ell+1} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}} \|B^\ell W^{\ell+1} - B_S^\ell \tilde{W}^{\ell+1}\|_F^2$. In ASYMINCHANGE, we also prune each layer sequentially, but to avoid the accumulation of error, we use an asymmetric formulation of the reweighted input change, where instead of approximating the *updated* input $B^\ell W^{\ell+1}$, we approximate the *original* input $A^\ell W^{\ell+1}$, i.e., we solve $\min_{|S| \leq k, \tilde{W}^{\ell+1} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}} \|A^\ell W^{\ell+1} - B_S^\ell \tilde{W}^{\ell+1}\|_F^2$. This problem is still a weakly submodular maximization problem, with the same submodularity ratio given in Propositions 3.1 and 4.1, with A^ℓ

replaced by B^ℓ therein (see Appendix B.1). Hence, the same approximation guarantee as in the symmetric formulation holds here. Moreover, a better approximation guarantee can again be obtained with a stronger notion of approximate submodularity, under stronger assumptions (see Appendix C). The cost of running GREEDY with the asymmetric formulation and reweighting is also the same as before (see Appendix B.2).

We evaluate all three variants in our experiments. As expected, ASYMINCHANGE usually performs the best, and LAYERINCHANGE the worst.

5.2 Per-layer budget selection

Another important design choice is how much to prune in each layer, given a desired global compression ratio (see Appendix F for the effect of this choice on performance). In our experiments, we use the compression ratio selection method introduced in [Kuzmin et al., 2019, Section 3.4.1], which can be applied to any layerwise pruning method, thus enabling us to have a fair comparison.

Given a network with L layers to prune, let $k = \alpha \sum_{\ell=1}^{L} n_\ell$, for some $\alpha \in [0, 1]$, be the global number of neurons/channels to keep. We want to select for each layer ℓ, the number of neurons/channels $k_\ell = \alpha_\ell n_\ell$ to keep, with α_ℓ chosen from a set of possible values, e.g., $\alpha_\ell \in \{0.05, 0.1, \cdots, 1\}$. We define a layerwise accuracy metric $P_\ell(k_\ell)$ as the accuracy obtained after pruning layer ℓ, with a budget k_ℓ, while other layers are kept intact, evaluated on a verification set. We set aside a subset of the training set to use as a verification set. Let P_{orig} be the original model accuracy. We select the per-layer budgets that minimize the per-layer accuracy drop:

$$\min_{k_1, \cdots, k_L} \{\tau : \forall \ell \in [L], P_\ell(k_\ell) \geq P_{\text{orig}} - \tau, \sum_{\ell=1}^{L} k_\ell \leq k\}. \tag{6}$$

We can solve the selection problem (6) using binary search, if the layerwise accuracy $P_\ell(k_\ell)$ is a non-decreasing function of k_ℓ. Empirically, this is not always the case, the general trend is non-decreasing, but some fluctuations occur. In such cases, we use interpolation to ensure monotonicity.

Alternatively, another simple strategy is to prune each layer until the perlayer error (the reweighted input change in our case) reaches some threshold ϵ, and vary ϵ to obtain the desired global compression ratio, as done in [Zhuang et al., 2018, Ye et al., 2020].

6 Empirical Evaluation

In this section, we examine the performance of our proposed pruning method in the limited-data regime. To that end, we focus on one-shot pruning, in which a pre-trained model is compressed in a single step, without any fine-tuning. We study the effect of fine-tuning with limited and full training data in Appendix E. We compare the three variants of our method, LAYERINCHANGE, SEQINCHANGE, and ASYMINCHANGE, with the following baselines:

- LAYERGREEDYFS [Ye et al., 2020]: for each layer, first removes all neurons/channels in that layer, then gradually adds back the neuron/channel that yields the largest decrease of the loss, evaluated on one batch of training data. Layers are pruned sequentially from the input to the output layer.
- LAYERSAMPLING [Liebenwein et al., 2020]: samples neurons/channels, in each layer, with probabilities proportional to sensitivities based on (activations $\times$ weights), and prunes the rest.
- ACTGRAD [Molchanov et al., 2017]: prunes neurons/channels with the lowest (activations $\times$ gradients), averaged over the training data, with layerwise ℓ_2-normalization. This criteria is the best performing criteria among the ones evaluated in [Molchanov et al., 2017].
- LAYERACTGRAD: prunes neurons/channels with the lowest (activations $\times$ gradients), averaged over the training data, in each layer. This is the layerwise variant of ACTGRAD.
- WEIGHTNORM [He et al., 2014]: prunes neurons/channels with the lowest output weights ℓ_1-norm, normalized by the number of neurons/channels in their layer. This is the best performing criteria among the ones considered in [He et al., 2014].
- LAYERWEIGHTNORM [Li et al., 2017]: prunes neurons/channels with the lowest output weights ℓ_1-norm, in each layer. This is the layerwise version of WEIGHTNORM,

- RANDOM: prunes randomly selected neurons/channels globally across layers in the network.
- LAYERRANDOM: prunes randomly selected neurons/channels in each layer.

We evaluate the performance of these methods on the LeNet model [LeCun et al., 1989] on the MNIST dataset [Lecun et al., 1998], and on the ResNet56 [He et al., 2016] and the VGG11 [Simonyan and Zisserman, 2015] models on the CIFAR-10 dataset [Krizhevsky et al., 2009]. To ensure a fair comparison, all experiments are based on our own implementation of all the compared methods.

To compute the gradients and activations used for pruning in LAYERSAMPLING, ACTGRAD, LAYERACTGRAD, and our method's variants, we use four batches of size 128 of training images, i.e., $n = 512$, which corresponds to $\sim 1\%$ of the training data in MNIST and CIFAR10. We consider two variants of the method proposed in [Ye et al., 2020]: a limited-data variant LAYERGREEDYFS which only uses the same four batches of data used in our method, and a full-data variant LAYERGREEDYFS-fd with access to the full training data.

We report top-1 accuracy results evaluated on the validation set, as we vary the fraction α of prunable (i.e., belonging to layers being pruned) neurons/channels kept. Accordingly, when pruning L layers, the global budget is set to $k = \alpha \sum_{\ell=1}^{L} n_\ell$, where n_ℓ is the number of neurons/channels in layer ℓ. Unless otherwise specified, we use the per-layer budget selection method described in Section 5.2 for all the layerwise pruning methods, except for LAYERSAMPLING for which we use its own budget selection strategy provided in [Liebenwein et al., 2020]. We use a subset of the training set, of the same size as the validation set, as a verification set for the budget selection method. Since LAYERSAMPLING is a random method, the number of neurons/channels it keeps does not always match exactly the desired budget k. To correct for that, we add or remove neurons/channels with the lowest/highest probability, as needed.

To disentangle the benefit of using our pruning method from the benefit of reweighting (Section 3.2), we report results with reweighting applied to all pruning methods, or none of them. Though, we will focus our analysis on the more interesting results with reweighting, with the plots without reweighting mostly serving as a demonstration of the benefit of reweighting.

Results are averaged over five random runs, with standard deviations plotted as error bars. We exclude WEIGHTNORM from plots, as it is always the worst performing method. We report Top-5 accuracy, compression ratio ($\frac{\text{original size}}{\text{pruned size}}$), and speedup ($\frac{\text{original number of FLOPs}}{\text{pruned number of FLOPs}}$) values in Appendix F.1. For additional details on the experimental set-up, see Appendix D. The code for reproducing all experiments is included in the supplementary materials.

LeNet on MNIST We pre-train LeNet model on MNIST achieving 97.75% top-1 accuracy. We prune all layers except the last classifier layer. Results are presented in Figure 1 (left). All three variants of our method consistently outperform other baselines, even when reweighting is applied to them, with ASYMINCHANGE doing the best and LAYERINCHANGE the worst. We also observe that reweighting significantly improves performance for all methods, except LAYERGREEDYFS. It also reduces variance in results; the bottom subplot has larger error bars than the top one.

ResNet56 on CIFAR-10 We use the ResNet56 model pre-trained on CIFAR-10 provided in ShrinkBench [Blalock et al., 2020], which achieves 92.27% top-1 accuracy. We prune all layers except the last layer in each residual branch, the last layer before each residual branch, and the last classifier layer. Results are presented in Figure 1 (middle). The sequential variants of our method perform the best. Their performance is closely matched by LAYERWEIGHTNORM for most fractions, except very small ones. LAYERINCHANGE performs significantly worst here than the sequential variants of our method. This is likely due to the larger number of layers pruned in ResNet56 compared to LeNet (27 vs 4 layers), which increases the effect of pruning earlier layers on subsequent ones.

VGG11 on CIFAR-10 We pre-train VGG11 model obtaining 90.11% top-1 accuracy. We prune all layers except the last features layer and the last classifier layer. Results are presented in Figure 1 (right). The three variants of our method perform the best. Their performance is closely matched by LAYERWEIGHTNORM for most fractions, except very small ones. LAYERINCHANGE performs similarly to the sequential variants of our method here, even slightly better at fraction 0.1, perhaps because the number of layers being pruned is again relatively small (9 layers).

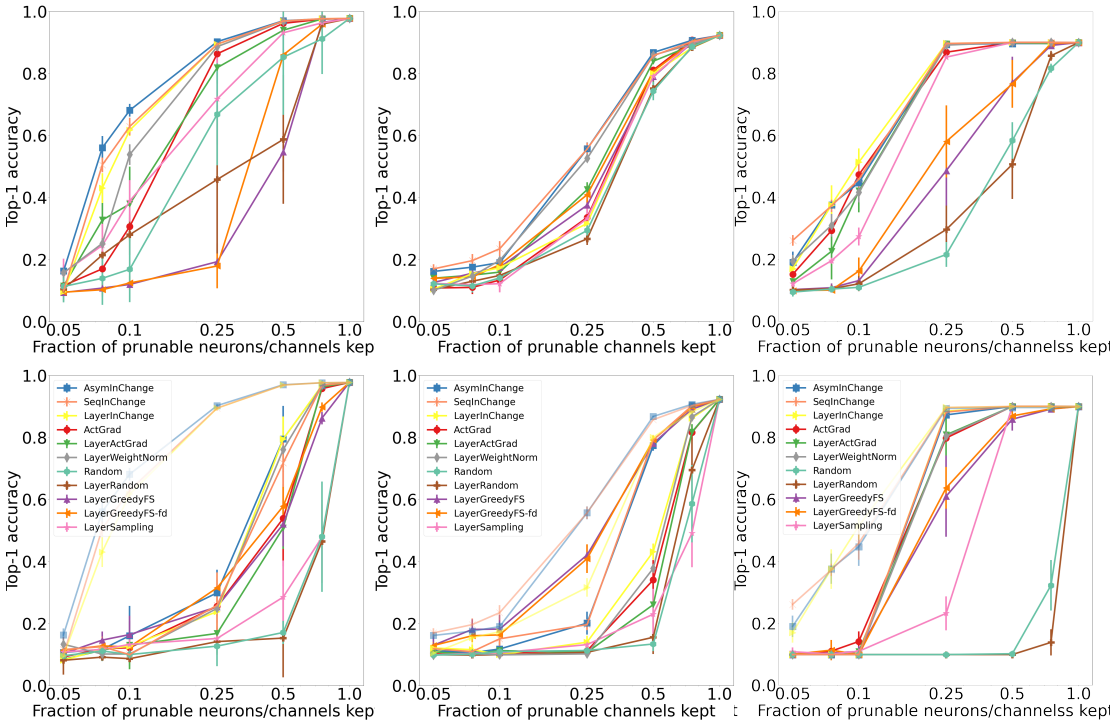

Figure 1: Top-1 Accuracy of different pruning methods applied to LeNet on MNIST (left), ResNet56 on CIFAR10 (middle), and VGG11 on CIFAR10 (right), with different fractions of remaining neurons/channels (in log-scale), with (top) and without (bottom) reweighting. We include the three reweighted variants of our method in the bottom plots (faded) for reference.

Discussion We summarize our observations from the empirical results:

- Our proposed pruning method outperforms state-of-the-art structure pruning method in various one shot pruning settings. As expected, ASYMINCHANGE is the best performing variant of our method, and LAYERINCHANGE the worst, with its performance deteriorating with deeper models. Our results also illustrate the robustness of our method, as it reliably yields the best results in various settings, while other baselines perform well in some settings but not in others.
- Reweighting significantly improves performance and reduces variance for all methods, except LAYERGREEDYFS and LAYERGREEDYFS-fd. We suspect that reweighting does not help in this case because this method already scales the next layer weights, and it takes into account this scaling when selecting neurons/channels to keep, so replacing it with reweighting can hurt performance.
- The choice of how much to prune in each layer given a global budget can have a drastic effect on performance, as illustrated in Appendix F.
- Fine-tuning with full-training data boosts performance more than reweighting, while fine-tuning with limited data helps less, as illustrated in Appendix E. Reweighting still helps even when fine-tuning with limited-data is used, except for LAYERGREEDYFS and LAYERGREEDYFS-fd, but it can actually deteriorate performance when fine-tuning with full-data is used. Our method still outperforms other baselines after fine-tuning with limited-data, and is among the best performing methods even in the full-data setting.

7 Conclusion

We proposed a data-efficient structured pruning method, based on submodular optimization. By casting the layerwise subset selection problem as a weakly submodular optimization problem, we are able to use the GREEDY algorithm to provably approximate it. Empirically, our method consistently outperforms existing structured pruning methods on different network architectures and datasets.

References

A. A. Bian, J. M. Buhmann, A. Krause, and S. Tschiatschek. Guarantees for greedy maximization of non-submodular functions with applications. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 498–507. JMLR. org, 2017. (Cited on 18)

D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020. (Cited on 1, 8, 20)

C. Bucila, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150464. URL https://doi.org/10.1145/1150402.1150464. (Cited on 1)

S. Buschjäger, P.-J. Honysz, and K. Morik. Very fast streaming submodular function maximization, 2020. (Cited on 20)

M. D. Collins and P. Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014. (Cited on 1)

M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015. (Cited on 1)

A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *arXiv preprint arXiv:1102.3975*, 2011. (Cited on 3, 4, 6)

M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/7fec306d1e665bc9c748b5d2b99a6e97-Paper.pdf. (Cited on 1)

M. El Halabi, F. Bach, and V. Cevher. Combinatorial penalties: Structure preserved by convex relaxations. *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, 2018. (Cited on 19)

E. R. Elenberg, R. Khanna, A. G. Dimakis, and S. Negahban. Restricted strong convexity implies weak submodularity. *arXiv preprint arXiv:1612.00804*, 2016. (Cited on 4, 6, 14)

Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. (Cited on 1)

B. Hassibi, D. Stork, and G. Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, pages 293–299 vol.1, 1993. doi: 10.1109/ICNN.1993.298572. (Cited on 1)

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (Cited on 8)

T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu. Reshaping deep neural network for fast decoding by node-pruning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 245–249. IEEE, 2014. (Cited on 2, 5, 7)

Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017. (Cited on 2)

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *Neural Information Processing Systems (NeurIPS) Workshops*, 2015. (Cited on 1)

T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021. (Cited on 1)

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009. (Cited on 8)

A. Kuzmin, M. Nagel, S. Pitre, S. Pendyam, T. Blankevoort, and M. Welling. Taxonomy and evaluation of structured compression of convolutional neural networks. *arXiv preprint arXiv:1912.09802*, 2019. (Cited on 1, 7)

V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations*, 2015. (Cited on 1)

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. (Cited on 8)

Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990. URL https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf. (Cited on 1)

Y. Lecun, C. Cortes, and C. Burges. The mnist databaseof handwritten digits, 1998. (Cited on 8)

B. Lehmann, D. Lehmann, and N. Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006. (Cited on 18)

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJqFGTslg. (Cited on 2, 5, 7)

W. Li, M. Feldman, E. Kazemi, and A. Karbasi. Submodular maximization in clean linear time, 2022. URL https://arxiv.org/abs/2006.09327. (Cited on 5)

L. Liebenwein, C. Baykal, H. Lang, D. Feldman, and D. Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxkOlSYDH. (Cited on 2, 4, 5, 7, 8, 20)

J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017. (Cited on 2)

Z. Mariet and S. Sra. Diversity networks: Neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015. (Cited on 2, 5)

K. McGuffie and A. Newhouse. The radicalization risks of gpt-3 and advanced neural language models. *arXiv preprint arXiv:2009.06807*, 2020. (Cited on 1)

P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2017. (Cited on 2, 5, 7)

B. Mussay, M. Osadchy, V. Braverman, S. Zhou, and D. Feldman. Data-independent neural pruning via coresets. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gmHaEKwB. (Cited on 2, 4, 5)

B. Mussay, D. Feldman, S. Zhou, V. Braverman, and M. Osadchy. Data-independent structured pruning of neural networks via coresets. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2021. doi: 10.1109/TNNLS.2021.3088587. (Cited on 2, 4, 5)

B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2): 227–234, 1995. (Cited on 3, 4)

G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions — I. *Mathematical Programming*, 14(1):265–294, 1978. (Cited on 3)

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. (Cited on 20)

H. Phan. huyvnphan/pytorch_cifar10, Jan. 2021. URL https://doi.org/10.5281/zenodo.4431043. (Cited on 20)

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556. (Cited on 8)

S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 31.1–31.12. BMVA Press, September 2015. (Cited on 2, 5)

J. Su, J. Li, B. Bhattacharjee, and F. Huang. Tensorial neural networks: Generalization of neural networks and application to model compression. *arXiv preprint arXiv:1805.10352*, 2018. (Cited on 1)

M. Sviridenko, J. Vondrák, and J. Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. *Mathematics of Operations Research*, 42(4):1197–1218, 2017. (Cited on 19, 20)

E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, 2019. (Cited on 1)

M. Ye, C. Gong, L. Nie, D. Zhou, A. Klivans, and Q. Liu. Good subnetworks provably exist: Pruning via greedy forward selection. *ICML*, 2020. (Cited on 2, 5, 7, 8)

Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/55a7cf9c71f1c9c495413f934dd1a158-Paper.pdf. (Cited on 2, 7)

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We specify that our focus is on the limited-data regime in both the abstract and introduction. See also the discussions on lines 143-151 and 207-213 on when our approximation guarantee is non-zero, and on lines 173-181 on the cost of our method.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include the code with instructions on how to reproduce all our experimental results in the supplemental material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 6 and Appendix D

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] See Appendix D

(b) Did you mention the license of the assets? [Yes] The license of the assets we used are included in the code we provide.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our code in the supplemental material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]