Neuron-level Structured Pruning using Polarization Regularizer

Tao Zhuang¹, Zhixuan Zhang¹∗, Yuheng Huang², Xiaoyi Zeng¹, Kai Shuang², Xiang Li³
¹Alibaba Group
²Beijing University of Posts and Telecommunications
{zhuangtao.zt, zhibing.zzx}@alibaba-inc.com, hyhlryf@bupt.edu.cn,
yuanhan@taobao.com, shuangk@bupt.edu.cn, leo.lx@alibaba-inc.com

Abstract

Neuron-level structured pruning is a very effective technique to reduce the computation of neural networks without compromising prediction accuracy. In previous works, structured pruning is usually achieved by imposing L1 regularization on the scaling factors of neurons, and pruning the neurons whose scaling factors are below a certain threshold. The reasoning is that neurons with smaller scaling factors have weaker influence on network output. A scaling factor close to 0 actually suppresses a neuron. However, L1 regularization lacks discrimination between neurons because it pushes all scaling factors towards 0. A more reasonable pruning method is to only suppress unimportant neurons (with 0 scaling factors), and simultaneously keep important neurons intact (with larger scaling factor). To achieve this goal, we propose a new regularizer on scaling factors, namely polarization regularizer. Theoretically, we prove that polarization regularizer pushes some scaling factors to 0 and others to a value $a > 0$. Experimentally, we show that structured pruning using polarization regularizer achieves much better results than using L1 regularizer. Experiments on CIFAR and ImageNet datasets show that polarization pruning achieves the state-of-the-art result.

1 Introduction

Network pruning is proved to effectively reduce the computational cost of inference without significantly compromising accuracy [Liu et al., 2019a]. There are two major branches of network pruning. One branch is unstructured pruning, which prunes at the level of individual weights [LeCun et al., 1989; Han et al., 2015; Zhou et al., 2019; Ding et al., 2019; Frankle and Carbin, 2019]. The other branch is structured pruning, which prunes at the level of neurons (or channels) [Wen et al., 2016; Liu et al., 2017; Ye et al., 2018]. Although unstructured pruning usually reduces more weights than structured pruning [Liu et al., 2019a], it has the drawback that the resulting weight matrices are sparse, which cannot lead to speedup without dedicated hardware or libraries [Han et al., 2016]. In this paper, we focus on neuron-level structured pruning, which does not require additional hardware or libraries to reduce computation on common GPU/CPU devices.

For structured pruning, one promising method is to associate each neuron with a scaling factor, and regularize these scaling factors in training. Then neurons with scaling factors below a certain threshold are pruned [Huang and Wang, 2018]. The regularizer on scaling factors is usually chosen to be L1 [Liu et al., 2017]. However, L1 regularizer tries to push all scaling factors to 0. It is often difficult to find a reasonable pruning threshold. For example, the distribution of scaling factors under L1 regularization for VGG-16 network trained on CIFAR-10 dataset is shown in Figure 1(a), where

*Corresponding author

the scaling factors distribute densely around the threshold value. A cut at the threshold value is not very reasonable because there is no margin around the threshold to separate the pruned neurons from the preserved ones. Pruning with this threshold will lead to severe accuracy drop. A more reasonable regularizer should separate the pruned and the preserved neurons more obviously, with a larger margin between them. To attain this goal, we propose a novel regularizer named polarization. Different from L1 regularizer that pushes all scaling factors to 0, polarization simultaneously pushes a proportion of scaling factors to 0 (thus pruning these neurons), and the rest scaling factors to a value larger than 0 (thus preserving these neurons). Intuitively, instead of suppressing all neurons in pruning, polarization tries to suppress only a proportion of neurons while keeps others intact. Polarization regularizer naturally makes a distinction between pruned and preserved neurons. And the resulted scaling factors are more separable. As shown in Figure 1(b), polarization leads to an obvious margin between scaling factors of the pruned neurons (the orange part) and the preserved ones (the blue part). Pruning using polarization is more reasonable because the pruned neurons has much smaller influence on network output than the preserved neurons. Our code is available at https://github.com/polarizationpruning/PolarizationPruning.

We summarize our contributions as follows:

- We propose a novel regularizer, namely polarization, for structured pruning of neural networks. We theoretically analyzed the properties of polarization regularizer and proved that it simultaneously pushes a proportion of scaling factors to 0 and others to values larger than 0.
- We verify the effectiveness of polarization pruning on the widely-used CIFAR and ImageNet datasets, and achieve state-of-the-art pruning results.

2 Related Work

Neuron-level structured pruning is also called channel/filter pruning in works focusing on convolutional neural network (CNN) structures. It is equivalent to unstructured pruning with the constraint that all weights connected to one neuron must be pruned or preserved together. There are usually three stages for structured pruning [Liu et al., 2019a]: 1) train an over-parameterized model, 2) prune the trained model according to some criteria, and 3) fine-tune the pruned model to boost performance. Structured pruning methods are characterized by how to handle these 3 stages, especially the training stage. Some methods [Liu et al., 2017, He et al., 2018a, 2019] do not add extra sparsity regularizer on the weights of networks in the training stage, while other works [Wen et al., 2016, Zhou et al., 2016, Alvarez and Salzmann, 2016, Lebedev and Lempitsky, 2016, He et al., 2017] impose group sparsity regularization on network weights during training. Yang et al. [2020] propose to use the Hoyer regularizer for pruning. Peng et al. [2019] exploit the inter-channel dependency to determine the optimal combination of preserved channels in the pre-trained model. Luo and Wu [2020] insert binary mask layers after convolution layers during training and removes filters corresponding to zero mask values. Zhuang et al. [2018] fine-tune the pre-trained model with discrimination-aware losses to choose the channels of each layer in the training stage. A group of sparsity regularization pruning
Another remotely related area is Neural Architecture Search (NAS), which is employed to find computationally efficient architectures [He et al., 2018b; Liu et al., 2019b]. And some works [Cai et al., 2020] require much more computation in the training and search processes. Our method is a sparsity regularization pruning method. So we do not focus on comparisons with these NAS methods in this paper.

3 Structured Pruning through Polarization

3.1 Preliminaries

Given a training set \{ (x_i, y_i) \}_{i=1}^N, where \( x_i, y_i \) denote the input feature and the label of sample \( i \) respectively, we need to train a neural network \( f(x; \theta) \) where \( \theta \) denotes the parameters of the network. We introduce a scaling factor for each neuron and represent the scaling factors as a vector \( \gamma \in \mathbb{R}^n \), where \( n \) is the number of neurons in the network. As in [Liu et al., 2017], we use the scale factor in Batch Normalization (BN) as the scaling factor for each neuron, because of the wide adoption of BN in modern neural networks. The objective function for network training with regularization on scaling factors is:

\[
\min_{\theta} \frac{1}{N} \sum_{i=1}^N L(f(x_i; \theta), y_i) + R(\theta) + \lambda R_s(\gamma)
\]

where \( L(\cdot) \) is the loss function, \( R(\cdot) \) is usually the L2 regularization on weights of the network, and \( R_s(\cdot) \) is the sparsity regularization on scaling factors of neurons. In pruning, a threshold is chosen and neurons with scaling factors below the threshold are pruned. In [Liu et al., 2017], the sparsity regularization is chosen to be L1, i.e. \( R_s(\gamma) = \| \gamma \|_1 \). The effect of L1 regularization is to push all scaling parameters to 0. Therefore, L1 regularization lacks discrimination between pruned and preserved neurons. A more reasonable pruning method is to only suppress unimportant neurons (with 0 scaling factor) and simultaneously keep important neurons intact (with larger scaling factor). To achieve this goal, we propose a new regularizer on scaling factors, namely polarization regularizer.

3.2 Polarization Regularizer

Let \( \gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n) \), and \( \mathbf{1}_n = (1, 1, \cdots, 1) \in \mathbb{R}^n \). Let \( \bar{\gamma} \) denote the mean of \( \gamma_1, \ldots, \gamma_n \):

\[
\bar{\gamma} := \frac{1}{n} \mathbf{1}_n^\top \gamma = \frac{1}{n} \sum_{i=1}^n \gamma_i
\]

\( \bar{\gamma}_i \) is the scaling factor for neuron \( i \). Obviously, a scaling factor should be positive and bounded. So it is reasonable to constrain each scaling factor to a range: \( \bar{\gamma}_i \in [0, a], \) where \( a > 0 \). L1 regularizer pushes all scaling factors to 0, because the optimal solution of \( \min_{\gamma \in [0, a]^n} \| \gamma \|_1 \) is 0. To obtain the polarization effect, we need to prevent the scaling factors from converging to one value. So we define the polarization regularizer to be:

\[
R_s(\gamma) = t \| \gamma \|_1 - \| \gamma - \bar{\gamma} \mathbf{1}_n \|_1
\]

\[
= \sum_{i=1}^n t |\gamma_i| - |\gamma_i - \bar{\gamma}|, \quad (t \in \mathbb{R}, \gamma_i \in [0, a])
\]

In Equation (2), we add the new term \(- \| \gamma - \bar{\gamma} \mathbf{1}_n \|_1 \) to the L1 term \( \| \gamma \|_1 \). The effect of \(- \| \gamma - \bar{\gamma} \mathbf{1}_n \|_1 \) is to separate \( \gamma_i \), \( 1 \leq i \leq n \) from their average as far as possible. Actually, \(- \| \gamma - \bar{\gamma} \mathbf{1}_n \|_1 \) reaches its
We use subgradients on these nondifferentiable points. To implement the constraint that we simply clamp the value of \( \gamma \), as we will show later, \( t \) also controls the proportion of scaling factors that equal 0 under polarization regularizer.

The polarization regularizer has several nice properties. The first property is permutation invariance: Let \( \pi(\gamma) \) denote the vector obtained by an arbitrary permutation of \( \gamma \) on its \( n \) dimensions. Then it is obvious that \( R_s(\pi(\gamma)) = R_s(\gamma) \), which means that the polarization regularizer is permutation invariant. This property ensures all neurons be equally treated in pruning: no prior pruning bias for any neuron. The second property is concavity, as stated in Lemma 3.1.

**Lemma 3.1.** \( R_s(\gamma) \) defined in Equation (2) is concave.

To see the effect of polarization regularizer more clearly, we need to solve the following minimization problem:

\[
\min_{\gamma \in [0,a]^n} R_s(\gamma) = \sum_{i=1}^{n} t|\gamma_i| - |\gamma_i - \bar{\gamma}|, \ (t \in \mathbb{R})
\]

(3)

This is a minimization of \( R_s(\gamma) \) on an \( n \)-dimensional cube. Utilizing the concavity of \( R_s(\gamma) \), we prove that the optimal value of Equation (3) is attained on vertices of the \( n \)-dimensional cube \([0,a]^n\). It is well known that an \( n \)-dimensional cube has \( 2^n \) vertices. Utilizing the permutation invariance property of \( R_s(\gamma) \), we finally prove the following theorem:

**Theorem 3.2.** A class of optimal solutions of Equation (3) is that either \([n\rho]\) or \([n\rho] + 1\) number of \( \gamma_i, (1 \leq i \leq n) \) are \( a \), and the rest are 0, where:

\[
\rho = \begin{cases} 
-\frac{t}{4} + 1/2, & -2 \leq t \leq 2 \\
0, & t > 2 \\
1, & t < -2
\end{cases}
\]

(4)

The detailed proof of Lemma 3.1 and Theorem 3.2 is in Appendix A. Theorem 3.2 states that the effect of polarization regularizer is to push a proportion of scaling factors to 0 and the rest scaling factors to \( a \). The proportion \( \rho \) depends piecewise linearly on the hyper-parameter \( t \) as in Equation (4). Therefore, Theorem 3.2 precisely shows the effect of polarization regularizer, and tells that the proportion of 0s is determined by \( t \).

As for the choice of a scaling factor for a neuron, it is convenient to reuse the scale factor in BN of this neuron if it has BN. It is also feasible to introduce an extra scaling factor on the output of a neuron as in [Huang and Wang 2018]. Our polarization regularizer is applicable for both choices of scaling factors above. To fully separate the two poles, the upper bound \( a \) of scaling factors should not be too small, especially when reusing the scale factors in BN. The reason is as follows. Let \( z_{in}^{(i)} \) and \( z_{out}^{(i)} \) be the input and output of BN on neuron \( i \), then BN performs the following transformation:

\[
\frac{\hat{z}}{z_{out}^{(i)}} = \frac{z_{in}^{(i)} - E[z_{in}^{(i)}]}{\sqrt{\text{Var}[z_{in}^{(i)}]}} = \gamma_i \hat{z} + \beta_i
\]

where \( E[z_{in}^{(i)}] \) and \( \text{Var}[z_{in}^{(i)}] \) are the mean and variance of \( z_{in}^{(i)} \) respectively. As pointed out in [Ioffe and Szegedy 2015], it is important for BN to include the *identity transform* by setting \( \gamma_i = \sqrt{\text{Var}[z_{in}^{(i)}]} \) and \( \beta_i = E[z_{in}^{(i)}] \). Therefore, \( a \) should be sufficiently large such that \( \forall 1 \leq i \leq n, a > \sqrt{\text{Var}[z_{in}^{(i)}]} \).

**Implementation Details** In the training stage, we optimize Equation (1) using Stochastic Gradient Descent (SGD). Due to the L1 norm in \( R_s(\cdot) \), there exist nondifferentiable points in Equation (1). We use subgradients on these nondifferentiable points. To implement the constraint that \( \gamma_i \in [0,a] \), we simply clamp the value of \( \gamma_i \) to \( a \) when it is larger than \( a \), and clamp it to 0 when it is smaller than \( 0 \) during the training process. All scaling factors are initialized to be 0.5 in image classification tasks as in [Liu et al. 2017]. We find that in our neural networks, the standard deviation of input to BN is mostly smaller than 1. Therefore, we set the upper bound of scaling factor \( a \) to 1 in this paper. Empirically, we find that the pruning result is insensitive to the exact value of \( a \).
3.3 Pruning Strategy

After training with the polarization regularizer, we obtain the values and distribution of scaling factors. We still need a threshold value to prune away neurons with small scaling factors. In [Liu et al., 2017], the threshold is determined by the percentile of neurons to be pruned. Although this method precisely controls the percentile of neurons to be pruned, it often leads to an unreasonable threshold, as shown in Figure 1(a). We utilize the polarization effect and propose a more reasonable strategy to automatically set the threshold. Because of the polarization effect, the distribution graph always has at least two local maximums (peaks): one is centered near 0, and others are centered around larger values, as is shown in Figure 2. Our strategy is to only prune the neurons that belong to the peak nearest to 0. So the threshold is located at the tail of the peak nearest to 0, as shown by the example in Figure 2. More specifically, when we draw the distribution histogram, we set the bin width to 0.01. Then we scan the bins in the histogram from left to right and find the first local minimum bin. Then the horizontal coordinate of this bin is the threshold for pruning. Using this strategy, we automatically prune the neurons with scaling factors nearest to 0. And empirically we find that this strategy produces good pruning results very robustly.

In our method, we control the number of reduced Floating-point Operations (FLOPs) by adjusting two hyper-parameters: $\lambda$ in Equation (1) and $t$ in Equation (2). Given a targeted number of reduced FLOPs, our aim is to choose values of $\lambda$ and $t$ so that polarization pruning will actually reduce the targeted number of FLOPs. Note that the number of reduced FLOPs is positively related to the number of pruned neurons. Theorem 3.2 gives the equation on how the ratio of 0 scaling factors, which equals $1 - \rho$, is determined by hyper-parameter $t$ under polarization regularizer alone. However, when minimizing the training objective in Equation (1), where polarization regularizer is just one part of the objective, there is no theory to quantitatively describe the relation between $\rho$ and $t$. It is obvious that the hyper-parameter $\lambda$ in Equation (1) also affects the number of reduced FLOPs. We will present experiment results on the effects of the hyper-parameters $\lambda$ and $t$ in Section 4.4.

Given a targeted reduced FLOPs, we use a search strategy to determine the hyper-parameters $\lambda$ and $t$. We first empirically set $\lambda$ and $t$ to a range and a value, and then proceed by line search until we are close to the targeted FLOPs. The search algorithm aims to find hyper-parameters that lead to a FLOPs reduction $F \in (F^* - \delta, F^* + \delta)$, where $F^*$ is the target reduction and $\delta$ is a permissible range. Our algorithm first performs a coarse search on $\lambda$, then carries out a fine search on $t$, as described in Algorithm 1. In the image classification experiments in this paper, the permissible range $\delta$ is set to 5%. Empirically, our algorithm needs to try out about 5 groups of hyper-parameters to stop. For each group of hyper-parameters we only need to train 30 epochs on ImageNet dataset for the FLOPs reduction to be stable, rather than the 120 epochs in full training. So the computational overhead of our search is not too much in practice. Note that Algorithm 1 is empirically designed for the image classification tasks in this paper. Strictly speaking, lines 11-18 in Algorithm 1 have a risk of being an infinite loop. However, empirically Algorithm 1 works well and efficiently in our experiments.

After pruning, we fine-tune the pruned network on the training data.

4 Experiments

In this section, we carried out extensive experiments to evaluate our approach on the image classification task, which is the most widely-used task to test pruning methods. We presents the our main
Algorithm 1 Search for hyper-parameters of Polarization Pruning

Input: Target FLOPs reduction $F^*$ with a permissible range $\delta$.

Output: Hyper-parameter values $(\lambda, t)$, such that, after pruning, the FLOPs reduction $F \in (F^* - \delta, F^* + \delta)$.

1. Initialize $(\lambda_l, \lambda_u, t) = (0, 2.0 \times 10^{-4}, 1.2)$

   // Fix $t$ and search for $\lambda$ such that $|F - F^*| < 2\delta$ using binary search.

2. do
3.   $\lambda \leftarrow (\lambda_l + \lambda_u)/2$;
4.   Train network with $(\lambda, t)$ until the corresponding FLOPs reduction $F$ is stable;
5.   if $F - F^* \geq 2\delta$ then
6.     $\lambda_u \leftarrow \lambda$;
7.   else if $F - F^* \leq -2\delta$ then
8.     $\lambda_l \leftarrow \lambda$;
9.   end
10. while $|F - F^*| \geq 2\delta$;

   // Fix $\lambda$ and search for $t$ such that $|F - F^*| < \delta$.

11. while $|F - F^*| \geq \delta$ do
12.   if $F - F^* \geq \delta$ then
13.     $t \leftarrow t - 0.1$
14.   else
15.     $t \leftarrow t + 0.1$
16.   end
17. Train network with $(\lambda, t)$ until the corresponding FLOPs reduction $F$ is stable;
18. end

experiment results in Section 4.3. In Section 4.4, we discuss the effect of hyper-parameters in our approach. To compare the effects of L1 and polarization regularizers, we visualize the distributions of scaling factors in the training process. We also compare the distributions of the scale factors in BN induced by different regularizers with that of the baseline model. At last, we experiment on a wider spectrum of FLOPs reduction.

4.1 Datasets and Compared Methods

As pointed out in [Gale et al., 2019], the same pruning method may behave differently on small datasets and large datasets. Therefore, we evaluate our method on both small datasets (CIFAR 10/100 [Krizhevsky et al., 2009]) and a large dataset (ImageNet [Russakovsky et al., 2015]). We also experiment with three widely-used deep CNN structures: the VGG-Net [Simonyan and Zisserman, 2015], ResNet [He et al., 2016], and MobileNet V2 [Sandler et al., 2018]. This also makes difficulties for us to compare with previous works, because many works only performed experiment in one dataset, or on one network structure. Therefore, on each dataset and network structure, we only compare with those works that have published experiment results on this dataset and network structure. For a method that released codes, we also run it in our experimental setting. If we get better results for this method than the originally published ones, we will use the better results for this method in our comparisons, and append a “(Our-impl.)” suffix for this method in our result tables. Otherwise we will use the originally published results for this method. We have reviewed all the compared methods in Section 2. Some methods, such as CCP-AC in [Peng et al., 2019], insert additional auxiliary classifiers into the network. As pointed out in [Peng et al., 2019], it is unfair to directly compare these methods with other methods. So we do not compare with these methods in our experiments.

4.2 Experimental Setup

The base VGG model is implemented following [Liu et al., 2017], which add BN to the fully connected layers of VGG. The base ResNet model is implemented following [He et al., 2016]. Specifically, our code implementation is based on PyTorch and Torchvision [Paszke et al., 2019]. We list the detailed parameters in our training in Appendix B. We adjust the hyper-parameters $\lambda$ and $t$ in our polarization regularizer to control the reduced FLOPs as described in Section 3.3. For ResNet and MobileNet V2, we prune blocks with residual connections as [He et al., 2019], where the last
Table 1: Results on CIFAR-10 and CIFAR-100. Best results are bolded.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>Approach</th>
<th>Baseline</th>
<th>Pruned</th>
<th>FLOPs Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Acc. (%)</td>
<td>Acc. (%)</td>
<td>Acc. (%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Drop (%)</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>ResNet-56</td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>93.80</td>
<td>93.27</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CP [He et al., 2017]</td>
<td>92.80</td>
<td>91.80</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AMC [He et al., 2018b]</td>
<td>92.80</td>
<td>91.90</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCP [Zhuang et al., 2018]</td>
<td>93.80</td>
<td>93.49</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCP-adapt [Zhuang et al., 2018]</td>
<td>93.80</td>
<td>93.81</td>
<td>-0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SFP [He et al., 2018a]</td>
<td>93.59</td>
<td>93.35</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FPGM [He et al., 2019]</td>
<td>93.59</td>
<td>93.49</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCP [Peng et al., 2019]</td>
<td>93.50</td>
<td>93.46</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeepHoyer [Yang et al., 2020] (Our-impl.)</td>
<td>93.80</td>
<td>93.54</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td>93.80</td>
<td>93.83</td>
<td>-0.03</td>
</tr>
<tr>
<td></td>
<td>VGG-16</td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>93.58</td>
<td>93.54</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td>93.88</td>
<td>93.62</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td>93.88</td>
<td>93.92</td>
<td>-0.04</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>ResNet-56</td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>72.49</td>
<td>71.40</td>
<td>1.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td>72.49</td>
<td>72.46</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>VGG-16</td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>73.83</td>
<td>74.20</td>
<td>-0.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COP [Wang et al., 2019]</td>
<td>72.59</td>
<td>71.77</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ours</td>
<td>73.83</td>
<td>74.25</td>
<td>-0.42</td>
</tr>
</tbody>
</table>

convolution layer in each residual block is padded with zeros to align with the input to the block and the details on pruning ResNet-50 are shown in Appendix C.

Table 2: Results on ImageNet. Best results are bolded.

<table>
<thead>
<tr>
<th>Model</th>
<th>Approach</th>
<th>Baseline</th>
<th>Pruned</th>
<th>FLOPs Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc. (%)</td>
<td>Acc. (%)</td>
<td>Acc. (%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-50</td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>76.15</td>
<td>74.88</td>
<td>1.27</td>
</tr>
<tr>
<td></td>
<td>SSS [Huang and Wang, 2018]</td>
<td>76.12</td>
<td>71.82</td>
<td>4.30</td>
</tr>
<tr>
<td></td>
<td>DCP [Zhuang et al., 2018]</td>
<td>76.01</td>
<td>74.95</td>
<td>1.06</td>
</tr>
<tr>
<td></td>
<td>FPGM [He et al., 2019]</td>
<td>76.15</td>
<td>74.13</td>
<td>2.02</td>
</tr>
<tr>
<td></td>
<td>CCP [Peng et al., 2019]</td>
<td>76.15</td>
<td>75.21</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>MetaPruning [Liu et al., 2019b]</td>
<td>76.6</td>
<td>75.4</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>SFP [He et al., 2018a]</td>
<td>76.15</td>
<td>62.14</td>
<td>14.01</td>
</tr>
<tr>
<td></td>
<td>PFP [Liebenwein et al., 2020]</td>
<td>76.13</td>
<td>75.21</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>AutoPruner [Luo and Wu, 2020]</td>
<td>76.15</td>
<td>74.76</td>
<td>1.39</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>76.15</td>
<td>75.63</td>
<td>0.52</td>
</tr>
<tr>
<td>MobileNet v2</td>
<td>AMC [He et al., 2018b]</td>
<td>71.8</td>
<td>70.8</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>MetaPruning [Liu et al., 2019b]</td>
<td>72.0</td>
<td>71.2</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>DeepHoyer [Yang et al., 2020] (Our-impl.)</td>
<td>72.0</td>
<td>71.7</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>72.0</td>
<td>71.8</td>
<td>0.2</td>
</tr>
</tbody>
</table>

4.3 Experiment Results

We compare the performances of different pruning methods in terms of the accuracy and FLOPs reduction. For each method, we report its baseline model accuracy (Baseline Acc.), its fine-tuned accuracy after pruning (Pruned Acc.), and the accuracy drop (Acc. Drop) between baseline and pruned accuracy. The FLOPs are reported as the pruning ratio, i.e. the ratio of the FLOPs reduced by pruning to the FLOPs of the baseline model.

The CIFAR dataset is a relatively small dataset for image classification. It is the most widely used dataset to compare neural network pruning methods. The experiment results on CIFAR datasets are shown in Table 1. Note that a negative accuracy drop in Table 1 means that the pruned model gets better accuracy than its unpruned baseline model. On CIFAR-10, ResNet-56 task, our method achieves the smallest accuracy drop (-0.03%) and the best pruned accuracy while pruning 47%
FLOPs, which is much better than other methods. On CIFAR-10, VGG-16 task, our method also achieves the smallest accuracy drop and the best pruned accuracy, with FLOPs reduction similar to NS. On the CIFAR-100 dataset, few previous works on channel pruning have experimented and reported their results. So we only compare with NS and COP on CIFAR-100. And our method still achieves the best results as shown in Table 1.

The ImageNet dataset is a much larger dataset compared to CIFAR. The experiment results on ImageNet dataset are shown in Table 2. On ImageNet, ResNet-50 task, our method achieves the smallest accuracy drop and the best pruned accuracy with a FLOPs very close to other methods. MobileNet V2 is optimized for computation efficiency on mobile devices and contains less redundant computations. Therefore, it is more difficult to prune compared to ResNet-50. On ImageNet, MobileNet task, our method still has the smallest accuracy drop with similar FLOPs reduction to other methods. In summary, our method achieves state-of-the-art channel pruning results on both CIFAR and ImageNet datasets.

4.4 Analysis

The Effect of Hyper-parameters

In Section 3.3, we point out that there is no theory to quantitatively describe how the number of reduced FLOPs depends on the hyper-parameters $\lambda$ in Equation (1) and $t$ in Equation (2). As shown in Figure 3, we empirically studied the effect of the hyper-parameters $t$ and $\lambda$ on the number of reduced FLOPs as well as the accuracy drop between baseline and the pruned model. Figure 3 shows that when $t$ gets larger, the reduced FLOPs also gets larger. This is reasonable because according to Theorem 3.2 when $t$ gets larger, polarization regularizer will push more scaling factors to 0, and thus more neurons/FLOPs will be pruned. At the same time, the overall trend of the accuracy drop is to gets larger, but with more fluctuations in the curve. This shows that the accuracy drop does not increase monotonically with respect to $t$. A larger $t$, and thus more FLOPs reduction, does not always lead to a larger accuracy drop. When $\lambda$ gets larger, both of the FLOPs reduction and accuracy drop get larger monotonically. This is also reasonable because $\lambda$ is the overall weight for polarization regularizer in Equation (1). As $\lambda$ gets larger, polarization regularizer has more influence in training, causing more neurons being pruned and larger accuracy drop.

Visualizing the Effects of Regularizers in Pruning

The effect of L1 and polarization regularizers can be visualized through the distribution/histogram of scaling factors. Although Figure 1 already shows an example, we visualize them in more details in Figure 4. Figure 4 shows the evolution of the distributions of scaling factors in the training process. In Figure 4, subfigures (a, b, c) correspond to the training process of baseline ResNet-50, pruning ResNet-50 using L1 regularizer, and pruning ResNet-50 using polarization regularizer on the ImageNet dataset respectively. In subfigure (c), we can clearly see that polarization regularizer gradually pushes the scaling factors to two clusters, with one cluster located at 0, and the other cluster located at a larger value. There is a clear margin between these two clusters. On the other hand, subfigure (b) shows that L1 regularizer pushes the scaling factors to one cluster located near 0.09. Visualization in this subsection verified our conclusion in Section 3.2 that polarization regularizer has the effect of pushing scaling factors to different poles, thus making the pruned and preserved neurons more separable.
Experiments on Wider Spectrum of FLOPs Reduction

Previous works on channel pruning usually experiment with a FLOPs reduction no more than 50%. A larger FLOPs reduction may not favor a regularization-based pruning method like ours, due to the huge regularization imposed in the training loss. So we add experiments on larger FLOPs reduction. We compare with a simple channel pruning method that is not based on regularization: the Uniform Channel Scaling (UCS) method, which uses a width-multiplier to uniformly increases or decreases the channel counts across all layers with the same proportion. We also compare with two regularization based methods: NS and DeepHoyer. The original DeepHoyer uses pre-trained models before pruning with the regularizer, which requires extra pre-training epochs before pruning. For fair comparisons, we run all methods using the same experimental setup in our paper, which means exactly the same number of training epochs are used for all methods. The results are in Table 3, which shows that our method performs consistently better under a wide spectrum of FLOPs reductions.

Table 3: Results on large FLOPs reduction.

<table>
<thead>
<tr>
<th>Dataset / Model</th>
<th>Approach</th>
<th>Baseline</th>
<th>Pruned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc. (%)</td>
<td>Acc. (%)</td>
</tr>
<tr>
<td>CIFAR10 / ResNet56</td>
<td>UCS</td>
<td>93.80</td>
<td>92.25</td>
</tr>
<tr>
<td></td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>93.80</td>
<td>91.20</td>
</tr>
<tr>
<td></td>
<td>DeepHoyer [Yang et al., 2020]</td>
<td>93.80</td>
<td>91.26</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>93.80</td>
<td>92.63</td>
</tr>
<tr>
<td>ImageNet / ResNet50</td>
<td>UCS</td>
<td>76.15</td>
<td>73.78</td>
</tr>
<tr>
<td></td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>76.15</td>
<td>70.61</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>76.15</td>
<td>74.15</td>
</tr>
<tr>
<td>ImageNet / MobileNetv2</td>
<td>UCS</td>
<td>72.0</td>
<td>63.71</td>
</tr>
<tr>
<td></td>
<td>NS [Liu et al., 2017] (Our-impl.)</td>
<td>72.0</td>
<td>65.65</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>72.0</td>
<td>67.49</td>
</tr>
</tbody>
</table>

5 Conclusion

In this paper, we propose a novel polarization regularizer and perform structured pruning by applying it on the scaling factors of neurons. Different from existing regularizers which push parameters to one pole (usually 0), polarization regularizer pushes parameters to two poles. We theoretically analyzed the properties of our polarization regularizer and empirically validate our analysis by examining the distribution of scaling factors in pruning. Extensive experiments on image classification tasks show that polarization regularizer produces the state-of-the-art pruning results, much better than the commonly used L1 regularizer. Our method is easy to implement and computationally efficient, requiring only one pass of network “pruning+fine-tuning”. The polarization regularizer can be conveniently added to neural network training, suppressing some neurons and activating others simultaneously. In the future, we will find more applications where the polarization effect is needed.
6 Broader Impact

This work has the following potential positive impact in the society: because our work saves the computational cost of neural network inference, we can apply it to online inference services to reduce energy consumption, which is beneficial for environmental protection. We have not noticed any obvious negative consequences of our work.

Acknowledgments and Disclosure of Funding

This work is supported by Alibaba Group. The authors would like to thank the Search and Recommendation Division of Alibaba Group for support of this work. The authors are also grateful to their colleagues in Alibaba who helped to provide the computing resources: Tao Lan, Pengfei Fan, Xiaochuan Tang, and Xiaowei Lu.

References


