24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Dynamic Thresholding for Learning Sparse Neural Networks

Jin-Woo Park and **Jong-Seok Lee***

Abstract. This paper proposes a method called Dynamic Thresholding, which can dynamically adjust the size of deep neural networks by removing redundant weights during training. The key idea is to learn the pruning threshold values applied for weight removal, instead of fixing them manually. We approximate a discontinuous pruning function with a differentiable form involving the thresholds, which can be optimized via the gradient descent learning procedure. While previous sparsity-promoting methods perform pruning with manually determined thresholds, our method can directly obtain a sparse network at each training iteration and thus does not need a trial-and-error process to choose proper threshold values. We examine the performance of the proposed method on the image classification tasks including MNIST, CIFAR10, and ImageNet. It is demonstrated that our method achieves competitive results with existing methods and, at the same time, requires smaller numbers of training iterations in comparison to other approaches based on train-prune-retrain cycles.

1 Introduction

In recent years, 'going deeper' has been a driving force of the development of neural network architectures. Constructing deeper networks can be interpreted as using more parameters to solve a given task, thus is expected to bring performance improvement. However, this is considered as a significant problem in many practical use cases such as mobile and embedded applications. Networks having huge amounts of model parameters are difficult not only to be stored but also to be run on such cases with limited computing and memory resources.

To solve this problem, there have been many studies to compress and accelerate network architectures. Pruning is one of the approaches, referring to cutting off redundant weights in a neural network to reduce the number of weight parameters. Typically, the importance of the weights in the network is measured, unimportant weights are removed, where it is often assumed that smaller weights are less important, and then the pruned network is retrained, which is repeatedly applied [7, 8].

Recently, a variety of pruning methods have been developed [6, 10, 12, 14, 17, 23]. These studies report good results in removing a significant portion of unimportant weights in networks, however, need a retraining process after the pruning step to recover the performance (e.g., classification accuracy) comparable to that before pruning. This causes a few limitations. First, the necessity of the repeated pruning-retraining cycle may impose computational burdens. Second, the performance is not guaranteed to be recovered after retraining. Third, it is not easy to determine the threshold value defining which weights to be pruned, which usually needs to be determined via trial-and-error.

Some recent studies propose ways to train structurally efficient neural networks referred to as sparse neural networks. Unlike the aforementioned pruning techniques, they promote structural sparsity during training [13, 15, 16, 18–20]. Many of these sparsity-promoting techniques induce redundant weights to have near zero values during training. However, they usually do not make weights exactly zero and thus require an additional pruning step with a small threshold (and often also a retraining step). As a result, most of the aforementioned limitations of the pruning methods still exist. For instance, the methods in [13, 15] operate with steps of training with regularization, pruning, and then retraining, which include limitations of the uncertainty of the performance after retraining and the difficulty of determining the threshold. The method in [18] also has the difficulty of determining the threshold (see below and Figure 1).

In order to address the limitations of the previous approaches, we propose *dynamic thresholding*, a novel technique for training sparse neural networks from scratch with reduced trial-and-error in its learning process. We make the threshold parameters as trainable variables by approximating the pruning step as a continuous pruning function and train the threshold parameters with the gradient descent algorithm. Therefore, the thresholds are dynamically changed so as to prune or splice the weights during training. After the whole training procedure finishes, the final pruned network is directly obtained without additional pruning and retraining steps. Our dynamic thresholding method makes the thresholds themselves contribute to both inducing the weights to have near zero values and removing small weights. Therefore, we only need to consider to control the hyperparameter for sparsity, whereas the existing methods are required to adjust the combination of the sparsity hyperparameter and pruning thresholds.

Our experiments show that the proposed method achieves competitive results compared to the previous methods while reducing the burden of trial-and-error. As a representative result, Figure 1 compares the learning processes of LeNet-300-100 in terms of the test accuracy with respect to the training iteration for the existing sparse variational dropout (VD) method [18] and the proposed method for the MNIST classification problem (see Section 4 for more details). The accuracy of a trained network is shown as a line, and the accuracy of a pruned network is shown with a marker. In the case of sparse VD, adjusting the threshold produces pruned networks with different compression ratios from a single trained network, which needs to be determined manually. In addition, a significant accuracy loss occurs after pruning, and the amount of the loss is observable

*Yonsei University, Republic of Korea, email: {jin-woo.park, jongseok.lee}@yonsei.ac.kr

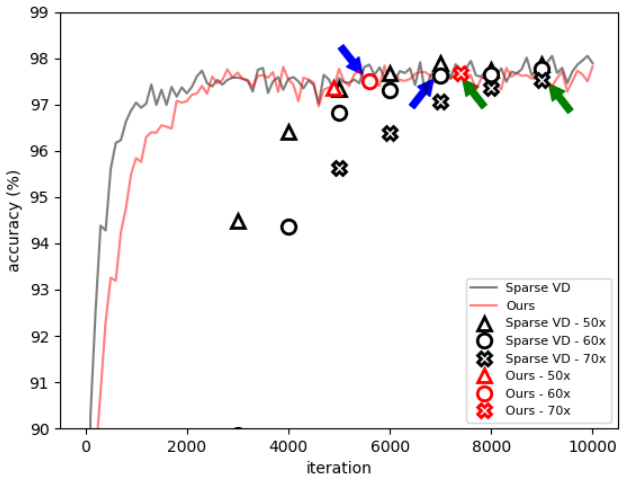


Figure 1: Comparison of the learning processes of LeNet-300-100 for the MNIST dataset between the sparse variational dropout (VD) method [18] (black color) and our proposed dynamic thresholding method (red color). Each line indicates the test accuracy of the trained network. Each marker denotes the test accuracy of the pruned network with a compression ratio of 50x, 60x, or 70x. Note that in the case of sparse VD, multiple pruned networks having different compression ratios are obtained from the trained network at a certain training iteration using different threshold values determined manually. On the other hand, our method automatically adjusts the threshold values during training. In addition, an accuracy loss occurs due to pruning in the case of sparse VD, whereas our method does not have such an issue. As a result, our method produces pruned networks faster than sparse VD for the same compression ratios and similar accuracy (marked with arrows of the same colors).

only after performing pruning. On the other hand, our method allows us to monitor the compression ratio and the accuracy of the pruned network immediately during training, which alleviates the necessity of the trial-and-error procedure. Furthermore, any accuracy loss is not involved due to pruning, which accelerates the overall process. As a result, our method requires smaller numbers of training iterations than sparse VD (e.g., 5600 vs. 6700 iterations for 60x (marked with blue arrows), and 7400 vs. 9100 iterations for 70x (marked with green arrows)), which results in faster training time of our method (i.e., 341.6 vs. 448.9 seconds for 60x, and 451.4 vs. 609.1 seconds for 70x).

The remainder of the paper is organized as follows. Section 2 reviews the related work. Section 3 presents the proposed method in detail. The experimental validation is shown in Section 4. Finally, conclusion is given in Section 5.

2 Related Work

2.1 Neural Network Pruning

Neural network pruning typically repeats the train-prune-retrain process to reduce the size of networks. Optimal brain damage [11] and optimal brain surgeon [9] are the representative early attempts to prune redundant parameters in neural networks. As deep neural networks became deeper and larger, pruning has been considered important as a way to reduce the size of networks and find efficient network structures. Deep compression [7, 8] proposes to prune the weights having small magnitudes, together with further compression schemes including quantization and Huffman coding. Dynamic network surgery (DNS) [6] also prunes weights based on their magnitudes, but it includes a step to recover pruned weight parameters during retraining to achieve better sparsity. The method proposed in [5] re-initializes the remaining weights after pruning to improve the classification performance of the pruned network.

To obtain simpler network structures after pruning, structured pruning methods also have been proposed, including filter pruning [12] and channel pruning [17]. They can produce more structured networks compared to weight pruning methods, but the compression performance tends to be lower than that of weight pruning.

There have been also studies on investigating the domains or criteria to choose redundant weight parameters. For instance, the work in [14] introduces a method to perform pruning in the frequency domain. The method in [23] uses a channel selection criterion to identify the channels' contribution to the discriminative power. The method in [10] uses reinforcement learning to determine the optimal set of filters to be pruned.

These pruning techniques need to start from pretrained models, and require a retraining process. Therefore, it is difficult to find a satisfactory compression ratio without trial-and-error of selecting an appropriate pruning threshold and retraining required to confirm the sparsity of the pruned network.

2.2 Training Sparse Neural Networks

Another way to obtain economized network structures is to impose sparsity during training from scratch. Many of them rely on regularization techniques to induce weight values to shrink to zero. The method proposed in [19] sets an additional binary gate variable to control each weight connection, which is trained using the straight-through-estimator technique. The method in [16] approximates and minimizes the L0-norm of weights. Deep rewiring [2] uses a L1 regularizer, and randomly awakens dormant weights to retain the initially set sparsity level at every training step. Sensitivity-driven training [20] defines a term to quantify the effect of weight value change to the final output of the network, which is used in the regularization term. Similar to the case of pruning, some studies attempt to train structured sparse networks. Structured sparsity learning [21] imposes a regularizer on various sub-structures of the network (e.g., filters, channels, etc.) to remove them. Network slimming [15] uses a scaling factor for each channel of convolution layers and regularizes it to promote channel sparsity. The work in [13] defines synaptic strength to represent the strength of connection between the layer input and output, which is regularized during training.

There also exist other types of sparse neural network training. Sparse variational dropout (sparse VD) [18] uses a variational dropout technique to induce weights to be extremely small. The learning-compression algorithm proposed in [3] iteratively alternates a learning step and a compression step to solve the pruning task as an optimization problem. Centripetal stochastic gradient descent [4] induces multiple convolutional filters to become identical and merges them.

Most of the aforementioned methods commonly have a limitation in that they are only able to induce weights to small values instead of making them have exactly zero values, thus additional thresholds to remove weights need to be applied. This leads to the necessity of a trial-and-error process to determine optimal hyperparameters.

3 Proposed Method

Consider the structure of a neural network having total N weight parameters, $\{w_i\}_{i=1}^N$. For each weight w_i, a non-negative threshold parameter is assigned, t_i, to determine if the weight is to be pruned, i.e., it is pruned if $|w_i| < t_i$.

The goal of our proposed dynamic thresholding is to directly train the threshold parameters used for pruning as well as the weight parameters by gradient descent learning. To achieve this, the thresholds are included in the loss function used for training through a pruning function whose approximated version is differentiable. Let $\Phi(w,t)$ denote the pruning function:

$$\begin{aligned}\Phi(w,t) &= \frac{w}{2}\{\mathrm{sign}(w-t) - \mathrm{sign}(w+t) + 2\} \\ &= \begin{cases} 0\,, & |w| < t \\ w\,, & \text{otherwise} \end{cases}\end{aligned} \tag{1}$$

Let L_c be the primary loss accounting for the classification performance (e.g., cross-entropy). Then, the optimization problem that we want to solve can be formulated as

$$\{\hat{w_i}\}, \{\hat{t_i}\} = \underset{\{w_i\},\{t_i\}}{\arg\min} \left[L_c\left(\{\Phi\left(w_i, t_i\right)\}\right) + L_p\right] \tag{2}$$

Here, the second term L_p is the loss promoting sparsity, which will be explained later. The sparsified network is obtained by applying the optimized thresholds $\{\hat{t}\}$ to the obtained weights $\{\hat{w}\}$ using the pruning function.

3.1 Promoting Sparsity

The sparsity of the network can be promoted by maximizing the values of the thresholds. Therefore, we use the following loss term[†].

$$L_p = -\lambda \sum_{i=1}^{N} \log t_i \tag{3}$$

where λ is a positive hyperparameter to control the relative importance of the sparsity promoting loss term. If λ is large, the contribution of the L_p term becomes large and the threshold are forced strongly to be large. Here, we use the logarithm to control the thresholds easily, especially at the early stage of training where most of the threshold values are very small.

Finally, the overall optimization problem is written as

$$\{\hat{w_i}\}, \{\hat{t_i}\} = \underset{\{w_i\},\{t_i\}}{\arg\min} \left[L_c\left(\{\Phi\left(w_i, t_i\right)\}\right) - \lambda \sum_{i=1}^{N} \log t_i\right] \tag{4}$$

3.2 Global vs. Local Thresholds

The pruning threshold can be set at different scales. While we use a separate threshold for each weight parameter in Equations 3 and 4, it is also possible to use a global threshold applied to all weights or a threshold assigned to each layer. Using a global threshold or layer-wise thresholds reduces the number of parameters to be trained. On the other hand, individualized thresholds can consider the distinguished role of each weight at the cost of an increased number of parameters. We compare these three cases experimentally in Section 4.

[†]Another possible choice of L_p is the L1-norm of the pruned weights, i.e., $L_p = \lambda \sum_{i=1}^{N} |\Phi\left(w_i, t_i\right)|$, so that the magnitudes of the pruned weights become small. However, we observed that Equation 3 promotes sparsity better than this version in our preliminary experiments.

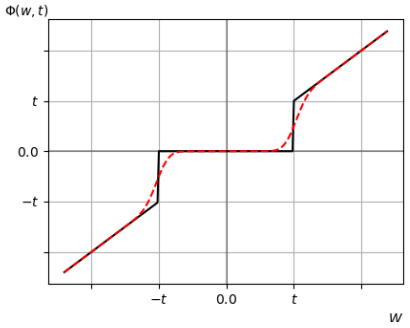
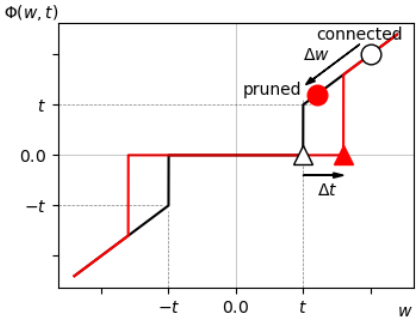
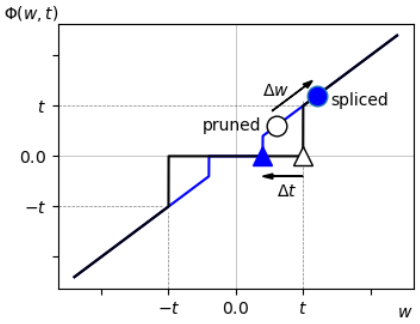


Figure 2: Graphical representation of the pruning function $\Phi(w,t)$ (black line) and its approximation using the Gauss error function with a temperature of 0.1 (red dashed line).



(a) Pruning a weight



(b) Splicing a pruned weight

Figure 3: Pruning and splicing mechanisms during training in the proposed method. The black (or colored) lines indicate the pruning function before (or after) the update of the parameters.

3.3 Approximation of Pruning Function

The pruning function given in Equation 1 is not differentiable and thus we cannot train w and t with the gradient descent optimization technique. To resolve this issue, we propose to approximate the discontinuous pruning function with a differentiable one by substituting the sign function with the Gauss error function having a temperature parameter, i.e.,

$$\tilde{\Phi}\left(w,t\right) = \frac{w}{2}\left\{\mathrm{erf}\left(\frac{w-t}{\tau}\right) - \mathrm{erf}\left(\frac{w+t}{\tau}\right) + 2\right\} \tag{5}$$

As temperature τ goes to 0, the Gauss error function becomes sharper, and is mathematically identical to the sign function when the temperature is equal to 0. Figure 2 compares the exact and approximated pruning functions. For a weight smaller than the associated threshold, our approximated pruning function reduces the gradient flow to the weight, while it affects little on the gradient for a weight larger than its threshold. In other words, the update of a weight depends on whether it is pruned or not at each training iteration.

In each forward step, the pruning function $\Phi(\cdot)$ is used to obtain the output of the pruned network. In each backward step, we update the weights and thresholds with their gradients computed by the derivatives of the approximated pruning function $\tilde{\Phi}(\cdot)$ instead of $\Phi(\cdot)$.

3.4 Effect of Pruning Function

We analyze how the weight update and threshold update work together during training. The update rule of weight w at training iteration k is expressed as

$$\begin{aligned} w^{k+1} =& w^k + \Delta w^k \\ =& w^k - \alpha \frac{\partial L}{\partial w^k} \\ =& w^k - \alpha \frac{\partial L_c}{\partial p^k} \frac{\partial p^k}{\partial w^k} \\ \approx& w^k - \alpha \frac{\partial L_c}{\partial p^k} \frac{\partial}{\partial w^k} \tilde{\Phi}(w^k, t^k) \end{aligned} \tag{6}$$

and the update rule of the associated threshold t is written as

$$\begin{aligned} t^{k+1} =& t^k + \Delta t^k \\ =& t^k - \alpha \frac{\partial L}{\partial t^k} \\ =& t^k - \alpha \frac{\partial L_c}{\partial p^k} \frac{\partial p^k}{\partial t^k} - \alpha \frac{\partial L_p}{\partial t^k} \\ \approx& t^k - \alpha \frac{\partial L_c}{\partial p^k} \frac{\partial}{\partial t^k} \tilde{\Phi}(w^k, t^k) + \alpha \lambda \frac{1}{t^k} \end{aligned} \tag{7}$$

where $L = L_c + L_p$ denotes the total loss function (Equation 4), α is the learning rate, and $p^k = \Phi(w^k, t^k)$. Note that the last term of Equation 7 is always positive, trying to increase the threshold value. In certain conditions, the sign of Δw^k becomes the opposite to that of Δt^k, and pruning of a weight remaining at the previous iteration or splicing of a previously pruned weight can occur. To see this, the gradients of the approximated pruning function in Equations 6 and 7 can be further expressed as

$$\begin{aligned} \frac{\partial}{\partial w} \tilde{\Phi}(w, t) =& \frac{1}{2} \{ \mathrm{erf}(w - t) - \mathrm{erf}(w + t) + 2 \} \\ &+ \frac{1}{\sqrt{\pi}} w \left(e^{-(w-t)^2} - e^{-(w+t)^2} \right) \end{aligned} \tag{8}$$

and

$$\frac{\partial}{\partial t} \tilde{\Phi}(w, t) = -\frac{1}{\sqrt{\pi}} w \left(e^{-(w-t)^2} + e^{-(w+t)^2} \right) \tag{9}$$

where we omit k and τ for simplicity. In Equation 8, both the first and second terms are positive, thus the total gradient is always positive. On the other hand, the sign of Equation 9 is opposite to the sign of w. When $w > 0$, Equation 9 becomes negative, meaning that Δw and Δt are in the opposite directions if the last term in Equation 7 is relatively small. Figure 3 illustrates the effect of this mechanism, showing the case where previously connected weight w is pruned due to decreasing w and increasing t (Figure 3a), and the case where previously pruned w becomes spliced due to increasing w and decreasing t (Figure 3b) (note that there are two other cases where the status of w remains the same). For $w < 0$, the same mechanism can happen in the left half-plane. Therefore, by training both the weights and thresholds, the weight connections in the network are adjusted via flexible pruning and splicing in our method.

4 Experiments

4.1 Experiment on Exclusive-OR Problem

First, we conduct an experiment for better understanding of how the proposed dynamic thresholding method works. To monitor the dynamic change of weight and threshold values, we consider an exclusive-OR (XOR) problem that can be solved by a small-sized network. The problem is to classify the input binary pairs (0, 0), (0, 1), (1, 0) and (1, 1) to their results of XOR calculation.

We train a two-layered fully connected network used in [6]. It has two input nodes, five hidden neurons, and one output neuron. Thus, the first and second layers of the network have 10 and 5 weight parameters, respectively. The sigmoid function is used as the activation function in both layers. We randomly generate 20,000 samples of input data by adding Gaussian noise. A half of them are used for training and the rest for test. The network is trained using the Adam optimizer for 10,000 iterations.

Figure 4 shows how the magnitudes of each weight (blue) and its corresponding threshold (red) evolve during training. The panels in the first and second rows are for the weights from the first and second input nodes to the hidden neurons, respectively. The last row shows the results for the weights from the hidden neurons to the output neuron. The state where a weight is pruned (i.e., the weight magnitude is smaller than its threshold) is indicated by gray color. Four weights (weights 1, 2, 6 in layer 1, and weight 1 in layer 2) are pruned at the end of the training procedure. At the beginning, most of these weights have larger magnitudes than their thresholds, but the thresholds are automatically adjusted to become larger than the weight magnitudes. In the case of weight 4 in layer 1, the weight is pruned at the beginning due to its threshold larger than the magnitude of the weight, but the weight magnitude exceeds the threshold and thus the weight becomes spliced at about iteration 1000. It is particularly interesting to observe that for weight 7 in layer 1, the weight is connected at the early stage of training, then pruned at about iteration 1300, but spliced again at about iteration 3000. These demonstrate that our method has a capability of dynamically determine the network structure during training.

4.2 Experiments on Image Classification

In order to evaluate the effectiveness of the proposed method, we perform image classification experiments. In particular, we follow the test protocols (datasets and network architectures) commonly used in previous studies [6, 8, 16, 18–20] in order to facilitate performance comparison with them. We use LeNet-300-100 and LeNet5 on the MNIST dataset, VGG-like on the CIFAR10 dataset, and AlexNet on the ILSVRC-2012 (ImageNet) dataset. The performance measures are the test classification accuracy loss (i.e., the absolute accuracy difference of the original and sparse networks) and the compression ratio (i.e., the ratio of the number of weights in the original network

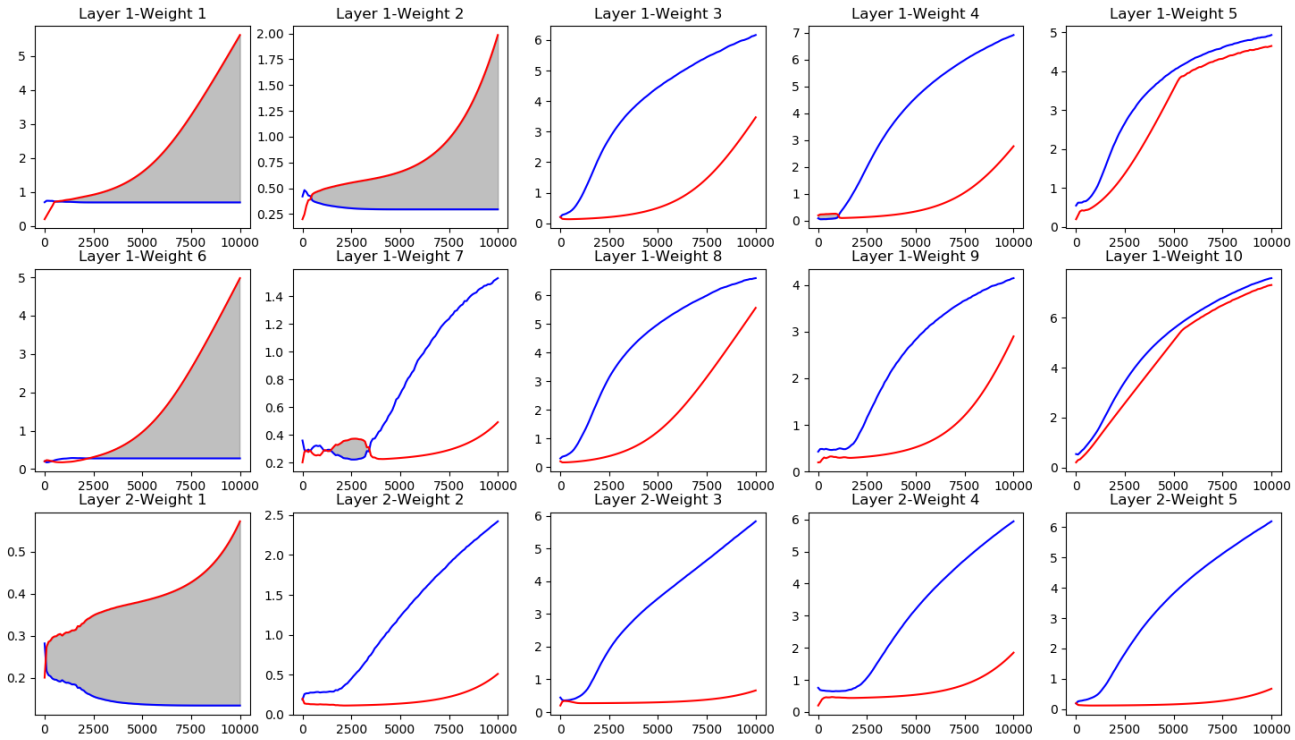


Figure 4: Evolution of each weight magnitude (blue) and the associated threshold (red) during training for the XOR problem. The x-axis is the number of iterations and the y-axis is the value. The gray area indicates the state where the threshold is larger than the magnitude of the corresponding weight and thus pruning of the weight occurs.

and the number of remaining weights in the pruned network). Our implementation is based on Tensorflow [1].

4.2.1 *Experimental Setup*

We perform the image classification experiments with following specifications. For the MNIST dataset, we train the fully connected network, LeNet-300-100, and the convolutional neural network, LeNet5. The weights of both networks are randomly initialized with a standard deviation of 0.01, and no dropout technique is used. We train them using the Adam optimizer for 10,000 iterations with a learning rate of 0.001. The batch size is set to 100. In addition, we test different scales of thresholds, i.e., global, layer-wise, and individual thresholds.

For the CIFAR10 dataset, we test our method using individual thresholds for the VGG-like architecture [22] having 13 convolutional and two fully connected layers. We train the network using the Adam optimizer for 100 epochs. The learning rate is initially set to 0.1 and is reduced by 10 times at every 25 epochs. The batch size is set to 128. The weight decay is used with a parameter of 5×10^{-4}. We set $\lambda = 1.8 \times 10^{-4}$.

Finally, we apply our method using individual thresholds to AlexNet with the ImageNet dataset to examine if our method successfully works for a large network structure and dataset. The momentum optimizer is used for training for 150 epochs with a batch size of 128. The learning rate is initially set to 0.01 and reduced by 10 times at epochs 70, 115, and 140. We set the momentum and weight decay parameters to 0.9 and 10^{-4}, respectively. The value of λ is set to 2×10^{-5}.

For these experiments, we observed that if we use threshold t directly in the pruning function in our method, too many weights are removed in the very early stage of training because the threshold increases fast via training. Therefore, we restrict t to be bounded up to 1 using a sigmoid function, which is sufficient considering the scales of weight values in our experiments. Using a sigmoid function also keeps the threshold to be always positive. The initial value of t is set to 0.0067 ($= sigmoid(-5)$) in all experiments. We observed that the initial value of the threshold is not critical thanks to flexible pruning and splicing in our method.

4.2.2 *Results*

First, the case using a separate threshold for each weight is compared to the cases with a global threshold and layer-wise thresholds on the MNIST dataset. Table 1 shows the results of training LeNet-300-100. As expected, learning individual thresholds yields significantly improved compression performance. Therefore, all the remaining experiments are conducted with individual thresholds.

Table 1: Performance of different scales of the pruning thresholds for LeNet-300-100 on MNIST

Scale	Top-1 acc. loss	Compression ratio
Global	$0.25\%p$	17x
Layer-wise	$0.33\%p$	23x
Individual	$0.36\%p$	**81x**

Table 2: Accuracy losses and compression ratios of different methods for image classification problems

Dataset	MNIST				CIFAR10		ImageNet	
Architecture	LeNet-300-100		LeNet5		VGG-like		AlexNet	
Method	Top-1 acc. loss	Compression ratio	Top-1 acc. loss	Compression ratio	Top-1 acc. loss	Compression ratio	Top-1 acc. loss	Compression ratio
Iterative pruning [8]	$-0.05\%p$	12.2x	$-0.03\%p$	11.9x	-	-	$-0.01\%p$	9.1x
DNS [6]	$-0.29\%p$	56x	$0.00\%p$	108x	-	-	$0.31\%p$	17.7x
Srinivas et al. [19]	-	-	$0.01\%p$	24x	-	-	$0.31\%p$	10.3x
Sparse VD [18]	$0.28\%p$	68x	$-0.05\%p$	280x	$-0.0\%p$	65x	-	-
Sensitivity-driven [20]	$0.31\%p$	103x	$-0.02\%p$	51x	-	-	-	-
L0 regularization [16]*	$0.28\%p$	12.2x	$0.20\%p$	70x	-	-	-	-
Dynamic Thresholding	$0.36\%p$	81x	$-0.03\%p$	151x	$-1.2\%p$	102x	$0.82\%p$	11.7x

* Results reported in [20]

Table 2 summarizes the results of the proposed dynamic thresholding method and the previous methods. The best and second best methods in terms of the compression ratio are marked with red and blue colors, respectively. Overall, our method achieves competitive performance, showing the best or second best compression ratios for all cases. The sparse VD method [18] shows the best compression ratio for LeNet5, but is inferior to ours for LeNet-300-100 and VGG-like. The sensitivity-driven method [20] records the best compression ratio for LeNet-300-100, but is far worse for LeNet5 than our method. In contrast, our method consistently shows good performance over various problems and network architectures. In particular, the result on the ImageNet dataset shows that the proposed method can successfully train a relatively large network structure with a large dataset even from scratch without using a pretrained model. The DNS method [6] shows better compression performance than ours, but is outperformed by our method for both LeNet-300-100 and LeNet5. Furthermore, DNS takes longer training time, i.e., 230 training epochs (90 epochs for pretraining and 140 epochs during pruning) for training AlexNet ImageNet, whereas our method takes only 150 epochs for training from scratch. The iterative pruning method, showing a lower compression ratio than our method for AlexNet, also requires training for over 960 epochs [8].

Together with the result shown in Figure 1, these results confirm that the proposed dynamic thresholding method can train sparse neural networks from scratch with good compression ratios without the necessity of a trial-and-error procedure to determine the thresholds manually, and furthermore, it reduces the training time.

4.2.3 Analysis

We examine further about the change of threshold and weight values during the training process. Figure 5 shows how the threshold values change over the training iterations for the VGG-like network trained on the CIFAR10 dataset. It is observed that the threshold values tend to increase, which promotes sparsity more and more. Furthermore, they become more and more different depending on the associated weights, indicating that they individually adapt to the associated weights.

Figure 6 shows the numbers of remaining weights in percentage with respect to the training epoch for AlexNet trained on the ImageNet dataset. It is seen that our method successfully induces the network to become more and more compact during training. Moreover, the flexibility is also observed, i.e., the level of sparsity does not monotonically increases but adaptively fluctuates, showing that

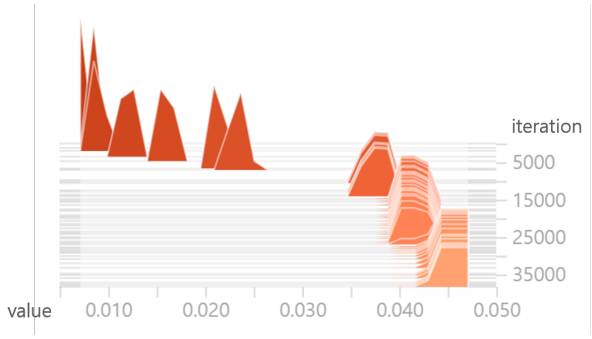


Figure 5: Distribution of the thresholds during training by the proposed dynamic thresholding method for VGG-like trained on the CIFAR10 dataset.

the dynamic pruning-splicing mechanism works effectively. Despite such a dynamic process, the classification accuracy, which is the primary objective of training, desirably increases monotonically (Figure 7).

5 Conclusion

In this paper, we proposed the dynamic threshold technique that can obtain pruned neural networks without the necessity of trial-and-errors of pruning threshold selection via simultaneous learning of the weights and thresholds. The dynamic mechanism of pruning-splicing during training was analyzed theoretically and experimentally. The experimental results showed that the proposed method successfully produces good performance in terms of accuracy and compression ratio reliably across different network architectures and datasets. It was also shown that our method converges fast in comparison to the existing methods.

ACKNOWLEDGEMENTS

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT) (NRF2016R1E1A1A01943283). In addition, this work was also supported by the Institute of Information & communication Technology Promotion (IITP) grant funded by MSIT (R7124-16-0004, Development of Intelligent Interaction Technology Based on Context Awareness and Human Intention Understanding).

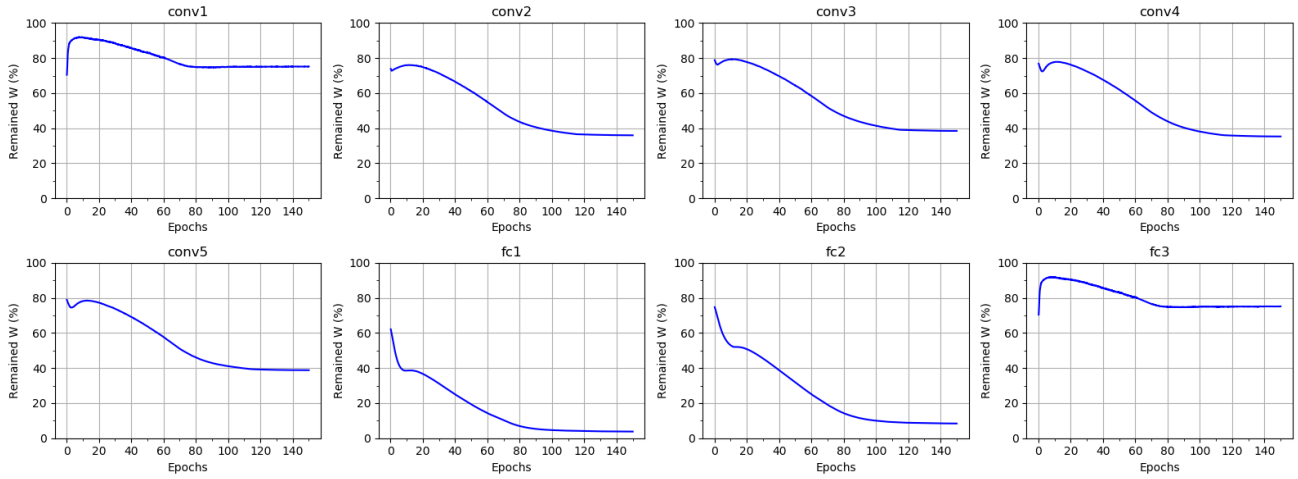


Figure 6: Percentage of the remaining weights in each layer with respect to the training epoch for AlexNet trained by the proposed method on the ImageNet dataset.

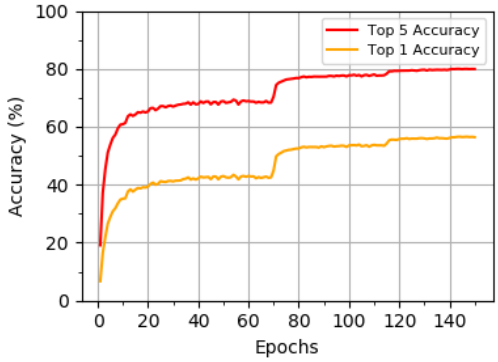


Figure 7: Top-1 and top-5 accuracy of AlexNet trained on the ImageNet dataset during training by the proposed method.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., 'Tensorflow: A system for large-scale machine learning', in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pp. 265–283, (2016).

[2] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein, 'Deep rewiring: Training very sparse deep networks', in *Proceedings of the International Conference on Learning Representations*, (2018).

[3] Miguel A. Carreira-Perpiñán and Yerlan Idelbayev, "'Learning-Compression" algorithms for neural net pruning', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8532–8541, (2018).

[4] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han, 'Centripetal SGD for pruning very deep convolutional networks with complicated structure', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4943–4953, (2019).

[5] Jonathan Frankle and Michael Carbin, 'The lottery ticket hypothesis: Finding sparse, trainable neural networks', in *Proceedings of the International Conference on Learning Representations*, (2019).

[6] Yiwen Guo, Anbang Yao, and Yurong Chen, 'Dynamic network surgery for efficient DNNs', in *Advances In Neural Information Processing Systems*, pp. 1379–1387, (2016).

[7] Song Han, Huizi Mao, and William J. Dally, 'Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding', in *Proceedings of the International Conference on Learning Representations*, (2016).

[8] Song Han, Jeff Pool, John Tran, and William Dally, 'Learning both weights and connections for efficient neural network', in *Advances in Neural Information Processing Systems*, pp. 1135–1143, (2015).

[9] Babak Hassibi and David G. Stork, 'Second order derivatives for network pruning: Optimal brain surgeon', in *Advances in Neural Information Processing Systems*, pp. 164–171, (1993).

[10] Qiangui Huang, Shaohua Kevin Zhou, Suya You, and Ulrich Neumann, 'Learning to prune filters in convolutional neural networks', in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pp. 709–718, (2018).

[11] Yann LeCun, John S. Denker, and Sara A. Solla, 'Optimal brain damage', in *Advances in Neural Information Processing Systems*, pp. 598–605, (1989).

[12] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf, 'Pruning filters for efficient convnets', in *Proceedings of the International Conference on Learning Representations*, (2017).

[13] Chen Lin, Zhao Zhong, Wu Wei, and Junjie Yan, 'Synaptic strength for convolutional neural network', in *Advances in Neural Information Processing Systems*, pp. 10170–10179, (2018).

[14] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong, 'Frequency-domain dynamic pruning for convolutional neural networks', in *Advances in Neural Information Processing Systems*, pp. 1051–1061, (2018).

[15] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang, 'Learning efficient convolutional networks through network slimming', in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2755–2763, (2017).

[16] Christos Louizos, Max Welling, and Diederik P. Kingma, 'Learning sparse neural networks through l_0 regularization', in *Proceedings of the International Conference on Learning Representations*, (2018).

[17] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin, 'Thinet: A filter level pruning method for deep neural network compression', in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5068–5076, (2017).

[18] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov, 'Variational dropout sparsifies deep neural networks', in *Proceedings of the International Conference on Machine Learning*, pp. 2498–2507, (2017).

[19] Suraj Srinivas, Akshayvarun Subramanya, and R. Venkatesh Babu, 'Training sparse neural networks', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 138–145, (2017).

[20] Enzo Tartaglione, Skjalg Lepsøy, Attilio Fiandrotti, and Gianluca Francini, 'Learning sparse neural networks via sensitivity-driven regularization', in *Advances in Neural Information Processing Systems*, pp. 3882–3892, (2018).

[21] 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*, pp. 2074–2082, (2016).
[22] Sergey Zagoruyko. 92.45% on CIFAR10 in Torch, 2015.
[23] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jin-Hui Zhu, 'Discrimination-aware channel pruning for deep neural networks', in *Advances in Neural Information Processing Systems*, pp. 883–894, (2018).