Pruning Method Using Correlation of Weight Changes and Weight Magnitudes in CNN

Azzaya Nomuunbayer and Sanggil Kang

Department of Computer Engineering, Inha University, Incheon, Korea

Abstract

Very complex deep learning models need to be compressed to be memory and cost effective, especially for applications on a mobile platform. We propose a new method of selecting weights to prune to compress convolutional neural networks. To select unimportant weights and get the best result, we combine typical weight magnitude pruning method with our method, which evaluates correlation coefficients of weights to measure the strength of a relationship between weight magnitudes and weight changes through the iterations. In the experimental section, we show our result of pruning 94% of weights in LeNet-5 without significant accuracy loss.

Keywords: Convolutional neural networks, Pruning weights, Weight correlation, Weight change

1. Introduction

Deep learning has become so popular in last few years, and the areas that use it have been expanding more and more. There are so many new improvements that are made each day to make deep learning as efficient as possible.

From deep learning architectures, convolutional neural networks (CNNs) are favored in image classification and video recognition and in natural language processing fields [1]. Some of the newest applications that use CNNs are automatic colorization [2] and object detection [3] of automatic image caption generation [4]. Because of the complexity of the architecture, CNNs have a huge number of weights. For example, well-known model VGGNet [5] has 138 M, and AlexNet [6] has 61 M weight parameters. The need of big memory and computational cost of large data are quite an issue for a mobile platform. This problem can be solved with the cloud computing, but it doesn't work for application usage off the Internet or in poor connections. Also, extra attention should be paid to the privacy. A reasonable solution for these issues is reducing memory by compressing CNNs.

Weight sharing and creating simpler networks are practices of neural network compression. In weight sharing technique, weight bits are diminished, and it is shared across multiple connections. In the study of Han et al. [7], each weight bits are reduced from 32 to 5. Despite the complex architecture of well-known networks, some researchers create simpler architecture, which derives fewer weights. For example, SimpleNet [8] with 13-layer architecture compresses weights by 2 to 25 times comparing to complex architectures.

Pruning is one of the popular techniques for compression. It removes insignificant connections or neurons from the network. In [9], they remove neurons based on neuron similarities.

Received: Sep. 12, 2018
 Revised : Dec. 1, 2018
 Accepted: Dec. 21, 2018

Correspondence to: Sanggil Kang
 (sgkang@inha.ac.kr)
 ©The Korean Institute of Intelligent Systems

©This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

The most familiar approach of pruning is based on weight magnitude. In [10], they assign a threshold and prune the weights with the magnitude smaller than the threshold. This work and others remove weights in all layers at the same time and don't consider how weights update, but it could prune smaller but important weights.

To solve this problem, we propose a pruning technique that compresses more than others without accuracy loss. We consider backpropagation algorithm of weight updates and prune unimportant weights layer by layer from the last layer. Unimportant weights are selected based on two criteria: 1) small magnitude and 2) small weight correlation of weight magnitude and weight change. In the experimental section, we present our result, which outperforms [11], on MNIST dataset.

Remainder of this paper is organized as follows: in Section 2, we study works that are related with neural network compression. Next, we describe our superior pruning method in detail in Section 3. Then in Section 4, our pruning method's success is proved with the experiments. Finally, we conclude our work in Section 5.

2. Related Work

There are many works that compressed deep neural networks (DNNs) with various methods. All of them share the same goal of compressing the network for quicker training and smaller storage and memory without affecting the performance.

Some introduced new networks similar to well-known networks but less complex. Simple networks are typically known with their low accuracy. In [8], they considered the accuracy and the memory efficiency and created SimpleNet. They focus only on the crucial parts of the DNN architecture. SimpleNet performs as good as complex models such as ResNet [12], VGGNet, and GoogleNet [13]; furthermore, it has a simple 13-layer architecture, which has fewer layers and fewer connections. In 2016, Iandola et al. [14] created a new architecture, named SqueezeNet. Having 50 times fewer weights, it successfully performs same as AlexNet on ImageNet dataset by replacing filters, decreasing input channel number, and downsampling.

There are researches of pruning by removing neurons from the network. In [15], they added noise outputs to the network, which increase the correlation of neuron activations in the hidden layers, and remove neurons with a high correlation during the training. This algorithm is called NoiseOut. In the result, NoiseOut method removed 85% of the parameters on SVHN dataset without loss of accuracy. Han et al. [11] proposed an-

Figure 1. Process of our pruning method.

other method of removing neurons for any model with fully connected layers. They proved that similar neurons are inessential and showed the way to prune them. In the end, they pruned 85% of the weights for MNIST and about 35% for AlexNet.

Another pruning technique is removing. Li et al. [16] introduced a method to remove filters with lower weight magnitudes across multiple layers. They attained 30% FLOP trimming for VGG-16 and ResNets without significant accuracy loss. Another method of pruning filters was proposed in [17]. They come up with a new framework, ThiNet, which selects inconsequential filters to prune by examining the filters' next layer. ThiNet reached 3.31 times FLOPs downsizing and 16.63 times compression on ResNet-50.

Another pruning method, which our method can relate the most, is pruning parameters/weights of the network. In most weight pruning methods, the magnitude/size of the weights is the main attribute for pruning decision. In [11], they prune the unneeded connections with a three-step procedure: training, pruning, and retraining. They achieved a good result by reducing the number of weights from 61 million to 6.7 million in AlexNet and from 138 million to 10.3 million in VGG-16 not affecting their accuracy.

The methods mentioned above did not consider weight change. Yet, we consider that weights are dependent with how they change.

3. Pruning Method Using Correlation of Weight Changes and Weight Magnitudes

Figure 1 is the schematic diagram of our pruning method process with three steps: training of CNN, analyzing of weights, and pruning of weights. As shown in Figure 1, firstly we fully train the network to study its weights. The important part of this step is to get the values of all the weights in every iteration. Then we analyze the importance of weights in contribution to the outputs and choose less important weights to prune using the weight amplitudes and the weight changes during the training. Lastly, we prune the selected weights and repeat the process.

Like in many related pruning methods, pruning the weights

Figure 2. Pruning order in layers.

in all layers at the same time can reduce the performance of the training significantly. To avoid this issue, we learn from the weight update technique. Weights update with the formula:

$$\omega_{new} = \omega_i - \eta^* \frac{\partial E}{\partial \omega_i}, \quad (1)$$

where ω is the first weight, η is the learning rate, and E is the squared error. Here we use the gradient descent method calculating the derivative of the squared error with respect to the weights, and it is done by the backpropagation algorithm. Therefore, starting to prune from the beginning can exaggerate the error and extend the computation time. To solve this problem, we gradually prune from the last layer, which is a lot more effective. It means weights will be pruned layer by layer from the last layer. For example, if a network consists of 2 convolutional and 2 fully connected layers as illustrated in Figure 2, pruning starts from the last layer, which is fully connected layer II. Next, we train the network because we need some regularization after each pruning to keep the performance. Then we prune weights in fully connected layer I, then in convolutional layer II, and then in convolutional layer I. After pruning once in all layers, pruning starts again from the last layer and continues so on until the accuracy starts decreasing.

To choose the weights to prune, we analyze the weights based on the statistical information of the weights in each iteration, which is the core part of our method. Weights that fit in two conditions will be selected from the choosing process. Our first condition is based on amplitudes of the weights; however, for the second condition we look at both weight amplitudes and weight changes together.

To analyze the weight values for our first condition, we assume the distribution of weights is normal (Gaussian distribution), with the probability density function shown in Eq. (2) according to the central limit theorem [18] because CNNs usually have millions of weights in image classification applications.

$$f(\omega|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\omega-\mu)^2}{2\sigma^2}}, \quad (2)$$

where μ is the mean, σ is the standard deviation, and σ^2 is a variance of weights in each layer.

To distinguish less important weights from the important ones, we need to determine an optimal threshold value. As in typical pruning method, all weights with the magnitude below the threshold fit in this condition. The threshold for this magnitude-based pruning is calculated by multiplication of a quality parameter and standard deviation of the weight [7]. Pruning based on amplitude only can prune some important weights of the network due to lack of information, so we add another condition to narrow the criterion and choose the least important weights. As the second condition, we consider both weights and weight changes together for pruning. Weight changes mean how much the weight has changed from one iteration to the next after each update. Our initial weights are selected randomly as a cold start; thus, at first the weight changes dramatically to minimize the cost function. So only for the first time, we don't calculate the weight changes from the first iteration. To decide from which exact iteration, we start to analyze the weight amplitudes and weight changes, we do the experiments. For the notation, we use n for the total number of weights, m for the total number of iterations, i for the weight number, ω for weights, and $\Delta\omega$ for weight changes. Therefore, $\Delta\omega_i^{(m)}$ is i -th weight's update from iteration $m-1$ to m , $|\omega_i^{(m)} - \omega_i^{(m-1)}|$. Here, we take the absolute value of the difference because the sign of the changes distorts the weight change information.

Having two variables, weight amplitudes and weight changes, we do bivariate analysis because the two variables can be considered dependent during training, and our bivariate simple random sample for the first weight would be:

$$(\omega_i^{(1)}, \Delta\omega_i^{(1)}), (\omega_i^{(2)}, \Delta\omega_i^{(2)}), \dots, (\omega_i^{(m)}, \Delta\omega_i^{(m)}), \quad (3)$$

where number in parentheses is the number of iterations, and m is the total number of iterations. If we have n number of weights, there will be n of these bivariate simple random samples. Because there is no iteration 0, weight changes for the first iteration $(\Delta\omega_1^{(1)}, \Delta\omega_2^{(1)}, \dots, \Delta\omega_i^{(1)}, \dots, \Delta\omega_n^{(1)})$ would be 0. Since the variables are dependent, we can calculate the correlation coefficient to measure the degree of the relationship of our two variables using the bivariate simple random sample. We use Pearson correlation [19] with the formula:

$$r(\omega_i, \Delta\omega_i) = \frac{m(\sum\omega_i\Delta\omega_i) - (\sum\omega_i)(\sum\Delta\omega_i)}{\sqrt{[m\sum\omega_i^2 - (\sum\omega_i)^2][m\sum\Delta\omega_i^2 - (\sum\Delta\omega_i)^2]}}, \quad (4)$$

where $r(\omega_i, \Delta\omega_i)$ is the correlation coefficient of the weight

amplitude and the weight change for the first weight. After calculating correlation coefficient for each weight, we will have the result in the matrix form as below:

$$\begin{matrix} & \Delta\omega_1 & \Delta\omega_2 & \cdots & \Delta\omega_n \\ \omega_1 & \left[\begin{array}{cccc} r(\omega_1, \Delta\omega_1) & \cdots & \cdots & \cdots \\ \cdots & r(\omega_2, \Delta\omega_2) & \cdots & \cdots \\ \vdots & \cdots & \ddots & \cdots \\ \omega_n & \cdots & \cdots & \cdots & r(\omega_n, \Delta\omega_n) \end{array} \right] \end{matrix} \quad (5)$$

The value of correlation coefficient is between -1 and 1 . If the weight amplitude and the weight change relationship of a weight is unimportant, then the corresponding correlation coefficient would be closer to 0 , and we consider them not so valuable in our network. Pruning those weights won't affect the performance of the network. With the experiment, we can decide the percentage of weights to prune at once without losing accuracy.

As a result, for our last step we prune weights that fit in both of our conditions, small magnitude and small correlation coefficient. After each pruning, there will be more and more weights with zero values, so to reduce the computational cost we don't include the pruned weights for the next calculation.

4. Experiment

We train CNN models on TensorFlow and use two different datasets, MNIST and CIFAR-10. The CIFAR-10 dataset contains 50,000 training and 10,000 test color images in 10 different classes. The MNIST dataset consists of 60,000 training and 10,000 test images of handwritten digits. We train widely known simple model LeNet-5 [20] on MNIST and our model on CIFAR-10 dataset.

CNNs need thousands of iterations to be fully trained, for example we have 11,000 iterations in our training of LeNet-5. As we explained, we don't calculate weights from the beginning to choose the important ones because of the random selection and this huge number of iterations. We tried to analyze weights of the last 10% to 90% of all iterations. As shown in Figure 3, analyzing the weights in the last 80% and 90% of all iterations, which is almost in all iterations, results lower accuracy as we expected. In other hand, studying weights in the last 10% of all iterations gives us the highest accuracy, which is 99.07. As training reaches the end, weight values get closer to the best condition minimizing the loss function. Therefore, learning the

Figure 3. Accuracy of the network and iteration percentage relationship.

Figure 4. Correlation coefficient percentage and accuracy.

weights in the last 10% of all iterations and removing selected ones in our first pruning results the best effect, the highest accuracy.

After calculating correlation coefficients in our second condition, weights with the smallest ones fit in this condition. We trained for several times with the smallest 10%, 20% and until 90% of the correlation coefficients. Smaller the percentage fewer the total number of weights to be pruned. As illustrated in Figure 4, as the percentage increases the accuracy drops. However, pruning fewer weights means smaller compression, so it takes much more time to reach the wanted compression rate result. Consequently, 40% is the best for both accuracy and compression rate.

With CIFAR-10 dataset, our method pruned 87.3% of all weights, while related work [16] pruned 64%.

Table 1. Comparison of the pruned results on LeNet-5

Layer	Weights' %	
	[11]	[Our method]
Convolution 1	66	4
Convolution 2	12	9
Fully connected 1	8	6
Fully connected 2	19	14
Total (left)	8	6
Total (pruned)	92	94

Han et al. [11] introduced a method and experimented with LeNet-5 on MNIST dataset. As shown in Table 1, by pruning redundant connections their method removed 92% of the weights. However, our method outperformed their result on same model and dataset. We pruned 94% of all the weights without significant accuracy loss.

5. Conclusion

In this work, we introduce a superior method to find insignificant weights of the network to prune for quicker and memory-efficient convergence. We remove unimportant weights based on weight magnitude as well as correlation, which computes the relationship degree of weight change and weight magnitude. Moreover, pruning gradually from the last layer of the network and retraining after each pruning result better accuracy and compression rate. As a result, our compression rate exceeds the existing method without losing accuracy. A compressed network makes implementation on a mobile platform a lot more possible.

For our future work, we would like to implement more accurate and automatic way of our manual calculations, such as how many percentages of weights with the smallest correlation coefficients to prune and from which iteration to start analyzing weights to prune for the first time. By improving our hardware, we also would like to experiment our method on deeper and more complex networks and datasets. So we can support generalization and make our algorithm more applicable for many others.

Conflict of Interest

No potential conflict of interest relevant to this article was reported.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. NRF-2016R1D1A1B03932447) and Institute for Information & communications Technology Promotion grant funded by the Korea government (MSIT) (No. 2017-0-00142, Development of Acceleration SW Platform Technology for On-device Intelligent Information Processing in Smart Devices).

References

- [1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015. <https://doi.org/10.1038/nature14539>
- [2] J. Hwang and Y. Zhou, "Image colorization with deep convolutional neural networks," 2016. Available http://cs231n.stanford.edu/reports/2016/pdfs/219_Report.pdf
- [3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems*, vol. 28, pp. 91-99, 2015.
- [4] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, 2015, pp. 3128-3137. <https://doi.org/10.1109/CVPR.2015.7298932>
- [5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. Available <https://arxiv.org/abs/1409.1556>
- [6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097-1105, 2012.
- [7] S. Han, H. Mao, and W. J. Dally, "Deep compression: compression deep neural network with pruning, trained quantization and Huffman coding," 2015. Available <https://arxiv.org/abs/1510.00149>
- [8] S. H. HasanPour, M. Rouhani, M. Fayyaz, and M. Sabokrou, "Lets keep it simple, using simple architectures to outperform deeper and more complex architecture," 2016. Available <https://arxiv.org/abs/1608.06037>

- [9] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," 2015. Available <https://arxiv.org/abs/1507.06149>
- [10] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," *Advances in Neural Information Processing Systems*, vol. 29, pp. 1379-1387, 2016.
- [11] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural networks," *Advances in Neural Information Processing Systems*, vol. 28, pp. 1135-1143, 2015.
- [12] 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*, Las Vegas, NV, 2016, pp. 770-778. <https://doi.org/10.1109/CVPR.2016.90>
- [13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, 2015, pp. 1-9. <https://doi.org/10.1109/CVPR.2015.7298594>
- [14] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5MB model size," 2016. Available <https://arxiv.org/abs/1602.07360>
- [15] M. Babaeizadeh, P. Smaragdis, and R. H. Campbell, "NoiseOut: a simple way to prune neural networks," 2016. Available <https://arxiv.org/abs/1611.06211>
- [16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2016. Available <https://arxiv.org/abs/1608.08710>
- [17] J. H. Luo, J. Wu, and W. Lin, "ThiNet: a filter level pruning method for deep neural network compression," in *Proceedings of 2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, 2017, pp. 5068-5076. <https://doi.org/10.1109/ICCV.2017.541>
- [18] W. Hoeffding and H. Robbins, "The central limit theorem for dependent random variables," *Duke Mathematical Journal*, vol. 15, no. 3, pp. 773-780, 1948. <https://doi.org/10.1215/S0012-7094-48-01568-3>
- [19] K. Pearson, "Notes on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, pp. 240-242, 1895.
- [20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998. <https://doi.org/10.1109/5.726791>

Azzaya Nomuunbayar received her B.S. degree in Mongolian University of Science and Technology, Mongolia in 2015. Currently, she is a M.S. candidate of the Department of Computer Science and Information Engineering at Inha University, Korea. Her research interest includes machine learning, computer vision, and deep learning.

E-mail: azzaya.nb@gmail.com

Sanggil Kang received the M.S. and Ph.D. degrees in Electrical Engineering from Columbia University and Syracuse University, USA in 1995 and 2002, respectively. He is currently an associate Professor in the Department of Computer Science and Information Engineering at Inha University, Korea. His research interests include semantic web, artificial intelligence, multimedia systems, inference systems, etc.

E-mail: sgkang@inha.ac.kr