Neural Network Optimization with Weight Evolution

Anonymous Authors[1]

Abstract

In contrast to magnitude pruning, which only checks the parameter values at the end of training and removes the insignificant ones, this paper introduces a new approach that estimates the importance of each parameter in a holistic way. The proposed method keeps track of the parameter values from the beginning until the last epoch and calculates a weighted average across the training, giving more weight to the parameter values closer to the completion of training. We have tested this method on popular deep neural networks like AlexNet, VGGNet, ResNet and DenseNet on benchmark datasets like CIFAR10 and Tiny ImageNet. The results show that our approach can achieve higher compression with less loss of accuracy compared to magnitude pruning.

1. Introduction

Over time, Neural networks have evolved from a single neuron(perceptron (Rosenblatt, 1958)) to Artificial Neural Networks(multi layer perceptron) (Guo et al., 2018; Sharma & Singh, 2017), followed by Convolution Neural Networks(CNN) (Fukushima, 1980; LeCun et al., 1989; Weng et al., 1993; Salman et al., 2018) and then moving on to Recurrent Neural Networks (Robinson & Fallside, 1987). In a neural network, the basic component is a neuron (which is called a filter in the case of a convolutional neural network), and every neuron has weights. As the neural network increases in depth and density, the number of neurons (and consequently, the number of weights in the network) also increases. During the training process, the weights are assigned values to create a reliable classifier that can differentiate between data points of different categories. After training, a standard way of reducing the size of the network is by pruning weights that have low values, setting them to 0. (Han et al., 2016). The standard approach for pruning neural networks involves multiple rounds of pruning followed by retraining to maintain the accuracy of the model. This technique is highly regarded and has led to significant reductions in the size of neural networks while often outperforming un-pruned networks in terms of accuracy. Many weight-based pruning algorithms are based on this method. However, this paper presents a critical analysis of commonly used pruning methods and highlights key concerns. A revised approach is also proposed for more effective pruning of neural networks.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Research Question

Currently, the commonly accepted pruning method involves examining the weights in a network after training, producing a threshold-based mask using the absolute values of the weights, and preserving those with a higher magnitude. This paper challenges this approach, instead presenting a more comprehensive perspective on how parameter magnitudes change throughout training.

Weight Evolution: This paper proposes that it is not enough to evaluate the importance of weights based solely on their values at the end of the training process. The article questions whether a weight that has a high magnitude initially could lose its importance over time and vice versa. Instead, the article suggests that monitoring the trend of weight value changes throughout training could be a more informative metric for assessing weight importance. As a result, the article advocates for a holistic approach for observing the progress of weight values during training and proposes a new metric called **weight evolution**, which is the main focus of the paper.

Section 3 delves deeper into the process of recording the evolution of weights. It also discusses the method of forward and fine pruning that constitute the complete pruning operation. We perform our experiments on the tiny-Imagenet and CIFAR-10 (Krizhevsky, 2009) datasets. The models used are AlexNet, DenseNet, ResNet and VGGNet. Details about the models and data are presented in the section 4. Finally, section 5 shows the results and comparison with other standardised pruning algorithms.

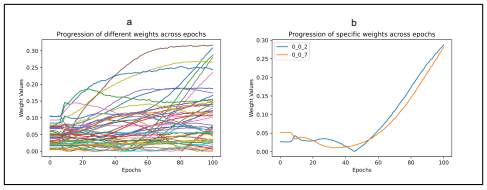

Figure 1. a. The progression of randomly chosen weights across 100 epochs of training.
b. The progression of specific weights across 100 epochs of training.

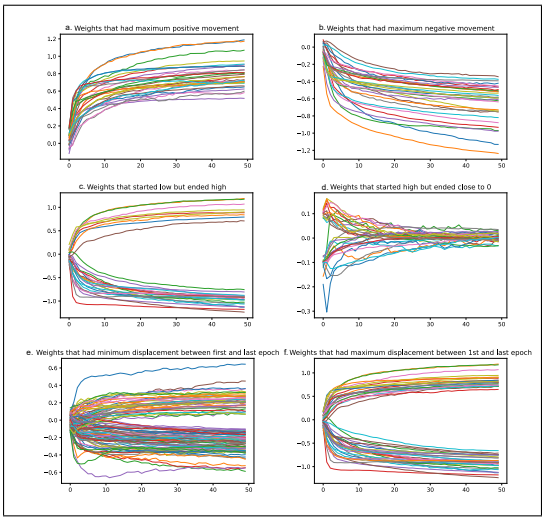

Figure 2. Different patterns of magnitude perturbation across training epochs

3. Proposed Method

The pruning method proposed in this work consists of the following components.

3.1. Evolution Of Weights

Before training a neural network, the weights are assigned arbitrary values. As training progresses, these values change and may increase or decrease at different rates. Figure 1a shows the changes in weight magnitudes for a set of weights in a neural network trained with the CIFAR-10 dataset. We observe that some weights initially increase in magnitude, then decrease, while others continue to increase throughout training. In Figure 1b, two weights have a similar magnitude at the end of training, but their trajectories are different, with one weight increasing and then decreasing, while the other weight consistently increases. We can use these trajectories to compare and prefer one weight over the other.

To further illustrate our findings, we present Figure 2, which organizes weights into different groups based on their trend of magnitude. Figure 2a shows the weights that had the greatest movement in the positive direction, while Figure 2b shows the weights that moved towards negative values. Interestingly, Figure 2d depicts weights that ended up close to 0 despite having larger initial magnitudes. Figure 2e shows weights that remained almost stationary and had final values close to their initial values. Figure 2f highlights weights that had the greatest difference between their initial and final magnitudes. These results illustrate the importance of monitoring weight evolution throughout epochs, rather than solely examining their final values. In the next sub-section, we explain how an importance vector can be generated to represent the progression of weights.

3.1.1. Weight Importance from Evolution

In order to accurately determine the importance of network parameters during each training epoch, we have developed a calculation process that involves aggregating the weighted magnitudes of the parameters. Simply analyzing the final values of the parameters may not provide an accurate assessment of their significance throughout the training process. Instead, we assign weights to the magnitudes of the parameters at each epoch based on their position in relation to the final epoch. This weighting approach places greater importance on parameters closer to the final epoch, while still incorporating values from previous epochs. After multiplying parameter magnitudes by their respective weights, we average the resulting products to generate an importance score for each parameter. Utilizing this method, we can create an importance vector that displays parameter progression throughout the training process.

For example, we can calculate the weighted importance of a weight or filter by retaining an array that logs their magnitudes at every epoch during the training process. We can then use this array to determine the weighted importance of the weight or filter based on the equation provided, with the final result reflecting the weight's importance in relation to its magnitude and progression across all epochs. Table 1 provides an example of recorded weight magnitudes across epochs. Using our previously described process, we determined that the three most important weights are Weight 1 (with an importance score of 14), Weight 2 (with an importance score of 13), and Weight 3 (with an importance score of 6.66).

To generalize the calculation process, a vector is created for each weight or filter ($val_i = [val_{i1}, val_{i2}, val_{i3}, ...val_{in}]$), where each element of the vector represents the magnitude of the weight or filter i at each epoch during the n epochs of training. This vector is then used to calculate the weighted importance, with the equation provided determining the importance value considering the last k epochs.

$$Imp_i = \frac{\sum_{L=0}^{k} val_{i(n-L)} * (n-L)}{\sum_{L=0}^{k} (n-L)} \tag{1}$$

where the value of L ranges from 0 to k. This means that 0 represents the immediate final epoch of training, while k represents the k^{th} epoch going backwards from the end of training. The resulting importance matrix serves as a primary reference to understand the importance of weights and guide the pruning process

Table 1. To calculate the importance of parameters across epochs, we take the weighted average of each parameter by recording its value at the end of each epoch in columns. We then calculate the aggregated importance by taking the weighted sum of the magnitudes of each weight, with the value of the multipliers shown in the last row indicating how much each magnitude is multiplied.

Weight Number	Ep 1	Ep 2	Ep ...	Ep k	Aggregated Importance
1	8	8	...	6	14
2	6	3	...	9	13
3	7	5	...	1	6.66
4	1	2	...	3	4.66
Multiplier	$*1$	$*2$	...	$*k$	

The importance matrix generated in this way is used in the next step to perform the pruning of the neural network.

3.1.2. Optimized Weighted Average Calculation

During training, the intermediate weights are not saved. Instead, at each epoch, the importance value of the weights is calculated using weighted average calculation. The generalized approach is described as follows

- if epoch == 1:
  - copy $current\_weights$ to $imp\_weights$
- else:
  - $sum = (1 + 2 + ... + (epoch - 1))$
  - $imp\_weight = imp\_weight * sum$
  - $imp\_weight+ = current\_weights * epoch$
  - $imp\_weight = imp\_weight/(1+2+...+epoch)$

3.2. Forward Propagation

3.2.1. Mask creation

In this particular step, we opted not to use the final magnitudes of parameters as the measure of their importance. Instead, the weighted averages of the parameters calculated in the preceding section serve as the importance metric. Furthermore, the importance values demonstrate a uniform distribution with the majority of values lying closer to 0. Depending on the desired level of compression, we select a threshold importance value and create a layer-specific mask tensor. This mask has an identical shape to the parameter

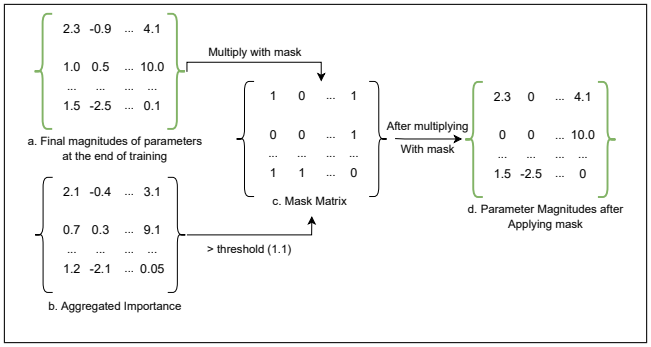

Figure 3. Pruning Of Network using the importance tensor

tensor in the training framework, with 0s for low-importance corresponding parameters and 1s when the importance value surpasses the threshold. Please refer to Figure 3 for a visual representation. The parameter values post-training are depicted in Figure 3a, while Figure 3b shows the aggregated importance tensor throughout training. In our example, we chose a threshold of 1.1 based on the level of compression we aimed to achieve. Thus, Figure 3c displays the mask tensor with 1s for parameters with importance values greater than the threshold and 0s for those that aren't. It's crucial to highlight that the parameter, importance, and mask tensors share the same dimensions, simplifying the network pruning process. By applying the mask on the parameters tensor, we generate a pruned network, as depicted in Figure 3d.

3.2.2. Pruning And Re-training

The second stage of the forward propagation process involves layer-by-layer application of the pruning mask. We begin by applying the mask to the parameters of the first layer and then retrain the network while ensuring that the first layer is constantly multiplied with the mask whenever the parameters are updated. Doing so prevents the previously nullified weights in the first layer from returning to non-zero values and maintains the level of network compression. Figure 4a showcases a trained neural network, while Figure 4b displays the pruned first layer and the retrained network. However, to preserve compression, the first layer is combined with the same mask during each batch. Figure 4c illustrates the inclusion of the mask for layer 2, thus expanding the scope of pruning. We retrain the network, combining the parameters of layer 1 and 2 with the mask until all layers are covered, completing the forward propagation process.

3.3. Fine-Pruning

Fine-pruning, a form of structured pruning, involves nullifying entire filters or neurons. After conducting forward propagation pruning, we observed that several network com-

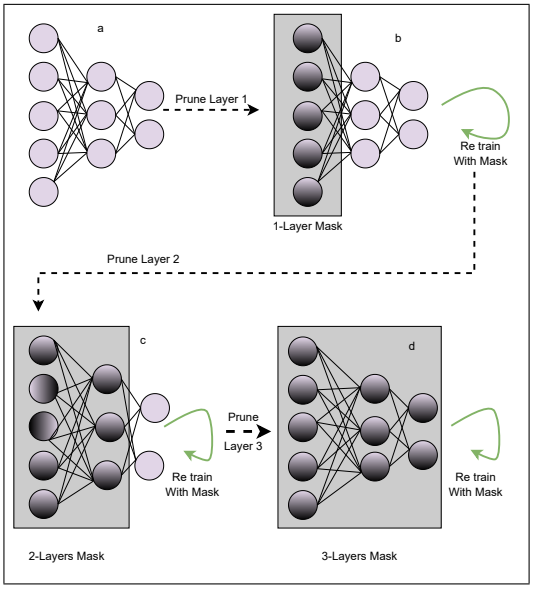

Figure 4. Pruning Of Network using the importance tensor

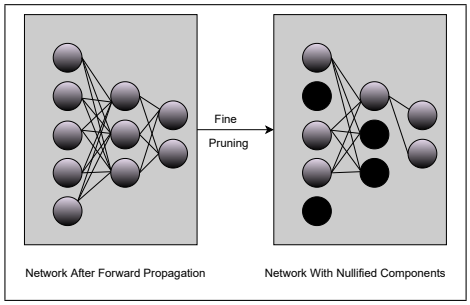

Figure 5. Fine-Pruning Network. Neurons in black represent network components with all parameters set to 0

ponents already underwent maximum parameter pruning. To execute fine-pruning, we nullify all parameters indiscriminately for such filters or neurons. If the threshold for the number of zero parameters in a given component is exceeded, we force all parameters in that same component to be set to zero. Fine-pruning can result in further model compression with little or no accuracy loss, depending on the desired accuracy. Figure 5 illustrates the process after forward propagation pruning, with the network on the left containing many neurons and most parameters set to zero, while the effect of fine-pruning is visible on the right, cancelling entire neurons and pruning connections to the next layer. The steps are depicted in Figure 6.

4. Experiment

Table 2 shows the different datasets used and the models that were trained on them. While training the models on the given datasets, the evolution of each individual weight

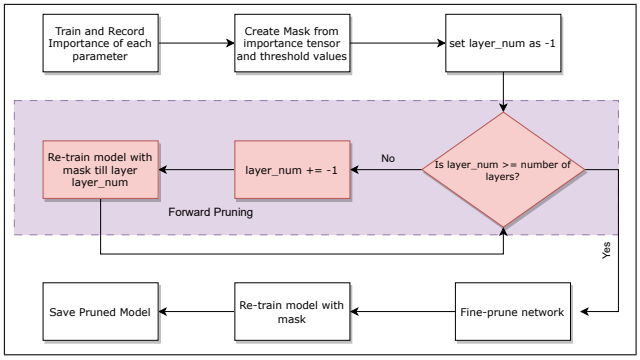

Figure 6. Complete Flow of operations consisting of Weight evolution , Forward Propagation Pruning and Fine-Pruning

was measured. At the end of training, the importance of the weights was calculated by the method mentioned in the Table 1. This importance was then used to apply Forward propagation followed by fine pruning on the models. The model was originally trained for 30 epochs and for 10 epochs after each process of pruning. The pruning method was compared with the magnitude based unstructured pruning method proposed by (Han et al., 2015).

Table 2. Combination Of Dataset and Model used to evaluate the proposed Algorithm

Model	Dataset
AlexNet	TinyImageNet
DenseNet	CIFAR10
DenseNet	TinyImageNet
ResNet18	CIFAR10
VGGNet	CIFAR10

5. Results And Comparisons

Figures 7, 8, 9, 10, and 11 show the result of fine pruning on the respective neural networks and datasets mentioned in table 2. The horizontal axis represents the compression level of the network and is calculated by the number of weights with a zero magnitude, divided by the total number of weights. The vertical axis represents the classification accuracy of the models. The orange line depicting the fine pruning method shows a higher tolerance towards compression for all cases against the blue line representing the general unstructured magnitude based pruning (Han et al., 2015). Our approach is able to maintain higher accuracy before dropping in accuracy, proving better identification of important neural network weights.

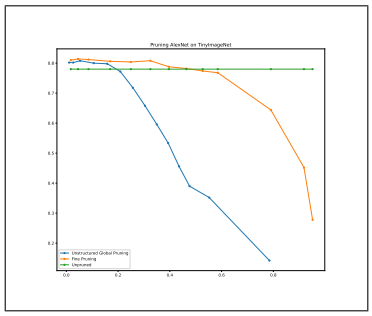

Figure 7. Pruning of AlexNet on the TinyImageNet data. The horizontal axis represents compression while the vertical axis represents accuracy. The blue line indicates state-of-art unstructured global pruning while the orange line indicates Fine Pruning method. The accuracy of the un-pruned network is depicted in green

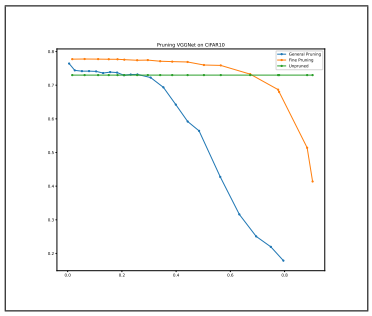

Figure 8. Pruning of VGGNet on the CIFAR10 data. Legends are same as Figure 7

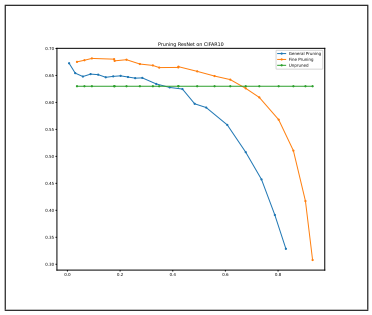

Figure 9. Pruning of ResNet on the CIFAR10 data. Legends are same as Figure 7

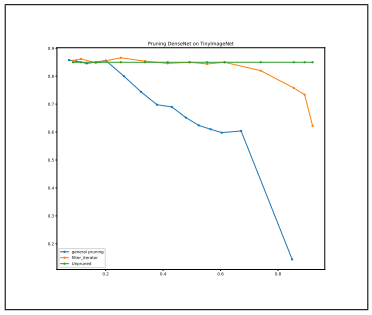

Figure 10. Pruning of DenseNet on the TinyImageNet data. Legends are same as Figure 7

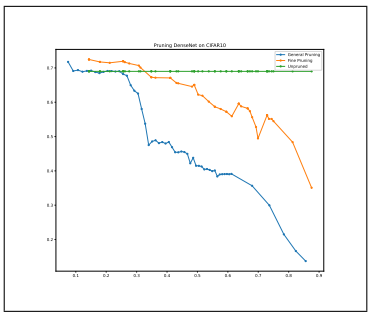

Figure 11. Pruning of DenseNet on the CIFAR10 data. Legends are same as Figure 7

References

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. ISSN 03401200. doi: 10.1007/BF00344251.

Guo, Y., Liu, Y., Georgiou, T., and Lew, M. S. A review of semantic segmentation using deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(2):87–93, 2018. ISSN 2192662X. doi: 10.1007/s13735-017-0141-z. URL https://doi.org/10.1007/s13735-017-0141-z.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–14, 2016.

Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to digit recognition, 1989. URL https://www.ics.uci.edu/$\sim$welling/teaching/273ASpring09/lecun-89e.pdf.

Robinson, A. and Fallside, F. The utility driven dynamic error propagation network. *Ieee*, 1, 1987.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Salman, H., Grover, J., and Shankar, T. Hierarchical Reinforcement Learning for Sequencing Behaviors. 2733 (March):2709–2733, 2018. doi: 10.1162/NECO. URL http://arxiv.org/abs/1803.01446.

Sharma, P. and Singh, A. Era of deep neural networks: A review. *8th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2017*, 2017. doi: 10.1109/ICCCNT.2017.8203938.

Weng, J. J., Ahuja, N., and Huang, T. S. Learning recognition and segmentation of 3-D objects from 2-D images. *1993 IEEE 4th International Conference on Computer Vision*, pp. 121–127, 1993. doi: 10.1109/iccv.1993.378228.