Dynamically Adding and Removing Neurons from Neural Networks

Aasta Gandhi [*1] Rohit Bandaru [*1]

Abstract

Neural networks are flexible and powerful models that can excel in many learning tasks. However, the size of the hidden layers is difficult to optimize causing under or over parameterization. A common approach to learning this complexity is initially training an overly large network and pruning parts of it for efficient inference. However, this method is computationally inefficient in training and susceptible to under-parameterization. To address these limitations, we introduce dynamic neuron creation and deletion, in which neurons of a model are dynamically pruned or added while maintaining accuracy close to a baseline. We first prune based on various activation statistics of neurons such as variance and correlation. We also propose methods to ”add” neurons by duplicating neurons based on these same statistics, but have not achieved experimental results. We compare different pruning methods on benchmark image classification datasets such as MNIST and CIFAR-10 and demonstrate that efficient fully connected and CNN architectures can be learned with a poor choice of initial model complexity.

1. Introduction

Deep learning has achieved incredible performance on many tasks in a variety of domains. However, most of these deep neural networks are computationally expensive. They can require hours or days on GPUs to train and occupy significant memory space. They can also be slow in inference and cause excessive energy consumption. More efficient neural networks would allow for more powerful applications and make this technology more accessible.

The architecture of a neural network must be carefully designed to succeed in its task. The number of layers and neurons or filters in each layer must be set before training. A model of low complexity would be computationally inexpensive, but can fail to capture the complexity of the data and achieve desired performance. A model of high complexity can be high performing, but computationally expensive.

The architecture of neural networks is often fixed manually, requiring trial and error to find the right setting. Even then, it is unlikely that a human can achieve the best setting through vague intuitions. Methods to learn this complexity can help find more efficient neural network architectures.

Current research focuses on taking large effective networks and making them more computationally efficient for inference. One approach to achieve this is pruning, which selectively removes some parts of the network. Pruning can be done on weights, or directly on neurons of a network. Pruning on weights involves removing weights that are not being used and results in sparse matrices. The problem with this method is that certain degree of matrix sparsity is necessary to observe computational gains, and also many hardware platforms do not support sparse matrix computation well. It is more efficient to have dense matrices, but of smaller sizes, which can be achieved by pruning on neurons (Hu et al., 2018). However, pruning the neurons themselves is not as popular because it tends to have a larger effect on the performance of the network.

However, we also want to be able to take under-parametrized networks, and automatically make them more complex to achieve better performance. One approach is adding neurons to layers of neural networks by evaluating the statistics and interactions between different neurons. By adding or replicating neurons at points of high activity, features can be separated and classified with greater resolution.

2. Related Work

Early work in this area focused primarily on compression by weight based pruning to reduce network complexity, while maintaining training error. Optimal Brain Damage (OBD) (LeCun et al., 1990) is a technique which predicts and prunes on low saliency weight parameters using the Hessian of the loss function. OBD demonstrated both speed up and a four times size reduction in a sparsely connected

[*] Equal contribution [1] Computer Science, Cornell University, Ithaca, New York. Correspondence to: Aasta Gandhi <apg67@cornell.edu>, Rohit Bandaru <rb696@cornell.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

network.

In denser, less constrained networks, weights can be pruned using thresholding methods. (Han et al., 2015b;a) demonstrated thresholding low-weight connections in Convolutional Neural Networks (CNNs) using iterative pruning, which retrains the pruned network to learn the weights of the new, sparse network. However, thresholding on small-magnitude weights primarily compresses fully connected layers. Convolutional layers can be better compressed by iteratively removing filters with low L1-norm of weights (Li et al., 2016). Magnitude-based pruning has effectively reduced model complexity without loss of accuracy, but does not necessarily reduce redundancy within layers. Pruning neurons based on weights has been explored.

There are also methods to encourage training of networks to be more prunable. A common approach demonstrated by (Han et al., 2015b) is to add L1 regularization to the weights, to encourage training a sparse network. (Babaeizadeh et al., 2016) encourages correlations between neurons by duplicating the target vector with noise. (Alvarez & Salzmann, 2017) introduces a loss term to encourage low rank weight matrices, which can be compressed more strongly.

A scarce area of research has been increasing the density of neural networks by adding neurons. Adding neurons is explored in the field of lifelong learning in which a network is given more tasks over time in an online setting. More tasks can require increasing neural network capacity. (Lee et al., 2017) propose a Dynamically Expandable Network (DEN) where the network expands by adding neurons and selectively retraining new neurons and old neurons that are not well utilized. However, expandable neural networks have not been applied on offline learning problems with fixed tasks. These expandable neural network methods can be used to efficiently find an effective neural network architecture.

3. Metrics of Neurons

Neurons are characterized by their weight, bias and activations. The mean and variance of these parameters is simplest method to understand the significance of each neuron. As described above, pruning on complex statistics, or even simpler methods such as thresholding have been attempted primarily on the weights of the neurons. However, the fixed architecture and high dependency between neurons and layers in fully-connected networks makes them more susceptible to over parametrization. To better understand the relation between neurons within layers themselves, we can observe simple heuristics such as variance and correlation between the activations of neurons.

Activations are the outputs of a neuron that are used as the input of a neuron in the next layer. This activation describes the ”activity” of each neuron. We can observe the variance between these activations and delete neurons with ”low activity” or low variance. Similarly, the activity of neurons in relation to other neurons can be measured via correlation. Higher correlation can mean similar activity between neurons, which can be combined into one neuron. These metrics can describe how significant or ”active” different areas of a network are.

In convolutional neural networks, the activity of a neuron, or kernel can be determined by measuring the average of the correlation between convolved features. Then, to prune kernels, we can take the correlation of each kernel activity as described above.

In this paper, we describe two methods two pruning. We first attempt magnitude-based pruning using the L1 norm on activations of each kernel. In each layer, we remove the kernel with the least individual activity. Second, we attempt to correlate the activity of neurons, or kernels themselves. By making this comparison, we can determine which method reduces model complexity effectively, as well as the inter-dependence of kernel activations.

4. Pruning Neurons

The goal of pruning neural networks is to remove as many of the parameters while not hurting performance. This can be formalized as such:

$$\min_{\mathcal{W}'} |\mathcal{W}'| \quad s.t. \quad \mathcal{C}(\mathcal{D}|\mathcal{W}') - \mathcal{C}(\mathcal{D}|\mathcal{W}) \leq B \qquad (1)$$

We want to minimize the cardinality of the weights or number of parameters with a bound on the difference in loss. This bound is expressed in Algorithm 1 as a sacrifice in accuracy. We apply various metrics to determine which neurons can be removed with minimal detriment to the performance.

5. Adding Neurons

Adding a neuron to a fully connected layer means incrementing the output dimension and adding another row to the weight matrix. Adding to a convolutional layer means adding an additional filter and incrementing the number of output channels.

$$\min_{\mathcal{W}'} |\mathcal{C}(\mathcal{D}|\mathcal{W}')| \quad s.t. \quad |\mathcal{W}'| - |\mathcal{W}| \leq B \qquad (2)$$

We want to increase the accuracy of the network with a bound on the difference in cardinalities of the weights or number of parameters added. Our goal in designing algorithms to add neurons is to best minimize the loss of the network while also having a tighter bound B, which would result in more efficient architectures.

The methods for pruning neurons have analogous adding methods. We start by using variance. Neurons with high variance activations are selected to be used to add additional neurons. From this we can determine the number of neurons to add and which neurons they are splitting from. These new neurons are meant to take some responsibility from the neurons they are splitting from. There are multiple methods of initializing these new neurons. We can either initialize it with random noise as done when training a neural network from scratch. We also tried copying the weights from the neurons they are splitting from and adding noise to both instances.

Other pruning methods can be applied in the expanding setting. We can take two neurons with very low activation correlation, and add a third neuron with an average of the weights. High L1 norm can also be used as a basis to split a neuron.

6. Method

We apply pruning iteratively. The network is first trained to achieve desired performance, and then neurons are pruned based on the metrics described above. The network is then fine-tuned to accommodate for the changes in structure. This is repeated for a set number of iterations, or until there are no neurons that can be pruned.

Algorithm 1 outlines our approach for pruning. We set thresholds T on a logarithmic scale to prune the network in an iteratively increasing aggression. This is because the network is more sensitive to changes the more pruned it is. The differences between thresholds or number of thresholds is analogous to a learning rate, but in our case we can view it as a pruning rate.

7. Results

We designed our method to be invariant to neural network architecture by using the same pruning algorithm for all experiments. We tested fully connected and convolutional neural networks on a NVIDIA K80 run on Google Colaboratory. For all experiments, we measured testing accuracy, number of operations in terms of floating point operations per second (FLOPS), wall clock time and the total number of parameters for each model.

We chose FLOPS and number of parameters as our primary metric because they are often used to understand complexities in CNNs. Both measures can be statically computed from model as they rely on the network's dimensionality and matrix operations alone. We used wall clock time to measure how long inference takes for a set of testing data.

Overall, our results show that the number of parameters and operations can be reduced. Our figures show the number of parameters and FLOPS plotted against the number of times the model has been trained and pruned ("training iteration"). Parameters (orange) are marked with the corresponding threshold used for pruning as well as the corresponding accuracy at that threshold (blue). The un-pruned model has a threshold of 0.

Algorithm 1 Dynamic Neuron Removal

```
Input: Neural Network Model $M$, Sacrifice $s$
Initial Accuracy $A_i$, $M = Pretrain(M)$
Thresholds $T = \{t_i\}$
for $i = 1$ to $n_t$ do
  Pruned Model $M' = Prune(N, t_i)$
  Number of Retraining Iterations $n_{rt} = 0$
  while True do
    $M' = Retrain(M')$, $A' = accuracy(M')$
    if $A' >= A_i - s$ then
      Accept pruned model $M = M'$
      continue
    end if
    if $n_{rt} = n_{rt-max}$ then
      Return $M$
    else
      $n_{rt} = n_{rt} + 1$
    end if
  end while
end for
Return $M$
```

7.1. Datasets

The MNIST dataset contains 62000 28 x 28, black and white handwritten digits from 0 to 9.

The CIFAR-10 dataset contains 60000 32x32 mutually exclusive, color images of animals and vehicles.

7.2. Fully Connected Network

A neural network with two convolutional layers with max pooling, 5 linear layers with ReLU activation were used. Pruning was done with correlation methods on the first 4 linear layers. The network was trained on 3 epochs on the MNIST dataset.

In this experiment, approximately 17% of the model parameters were pruned with a loss of accuracy of 1.9% (Figure 1) after the very first threshold (0.98).

7.3. Convolutional Neural Network

We designed a 4 layer CNN with a maximum with ReLU activation, max pooling and batch normalization and a single linear layer before the output. Pruning was done with the minimum L1 and correlation methods on the first 3 layers of

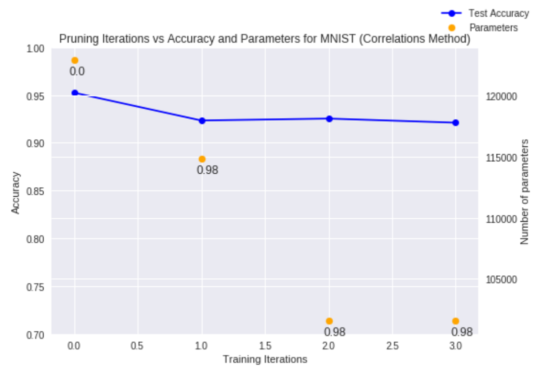

Figure 1. Pruning on highly correlated neurons in fully-connected network on MNIST dataset.

the net. The network was trained on 3 epochs for MNIST.

When pruning on minimum L1 norms of kernels, approximately 25% of the model parameters were pruned and the number of floating point operations reduced by 34% (Figure 2) with a loss of 1.1% accuracy over 3 iterations of training on the MNIST dataset. The average wall clock time through each iteration not including the baseline is 13.11 seconds.

When pruning correlations of kernels, approximately 29% of the model parameters were pruned and the number of floating point operations reduced by 71% (Figure 3) with a loss of 1.3% accuracy over 10 thresholds (10 distinct iterations). The average wall clock time for testing through each iteration not including the baseline is 12.97 seconds.

Pruning on the correlation of kernel activity decreases the number of operations 2 times more than using the minimum L1 norm of each layer, suggesting that understanding the activity within layers of a network can result in more effective compression while maintaining accuracy. Though, the number of iterations for correlation is more than 3 times of the L1 method, pruning with correlations achieves finer granularity and a lower loss of accuracy. Additionally, the average wall clock time for each iteration of testing is approximately the same indicating there is no significant difference in software inference time. However, since the number of operations and parameters decreases, hardware memory and resource utilization would presumably decrease as well.

7.3.1. CIFAR-10

We also tested our pruning methods on a VGG-11 based neural network on the CIFAR-10 image classification task.

When pruning on minimum L1 norms of kernels, approximately 3.3% of the model parameters were pruned and the number of floating point operations reduced by 6.6% (Figure 5) with a gain in accuracy over 9 iterations of training. The average wall clock time through each iteration not

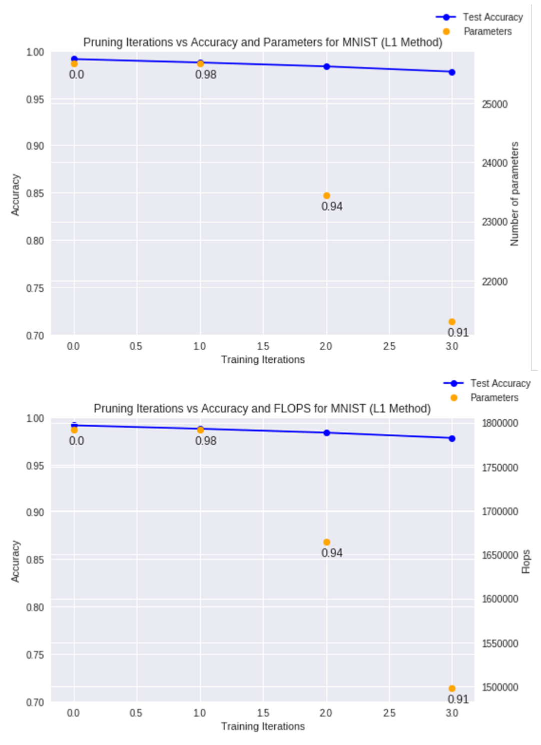

Figure 2. Pruning on CNN kernels with minimum L1 norm on MNIST dataset.

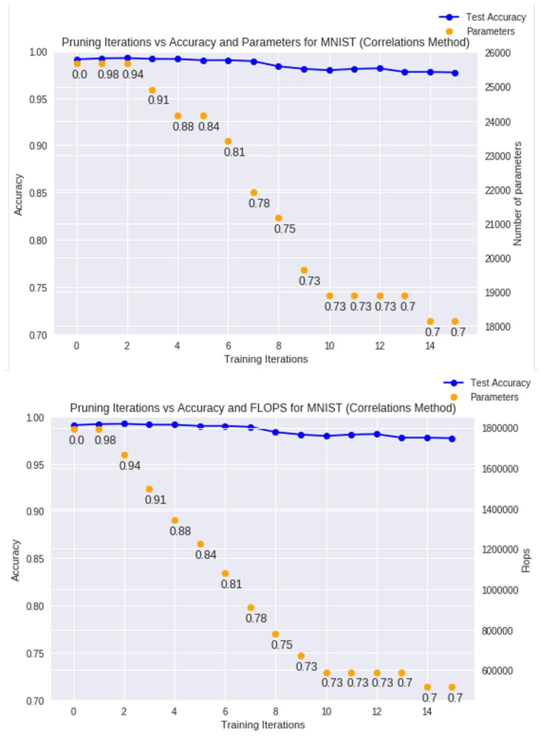

Figure 3. Pruning on highly correlated CNN kernels on MNIST dataset.

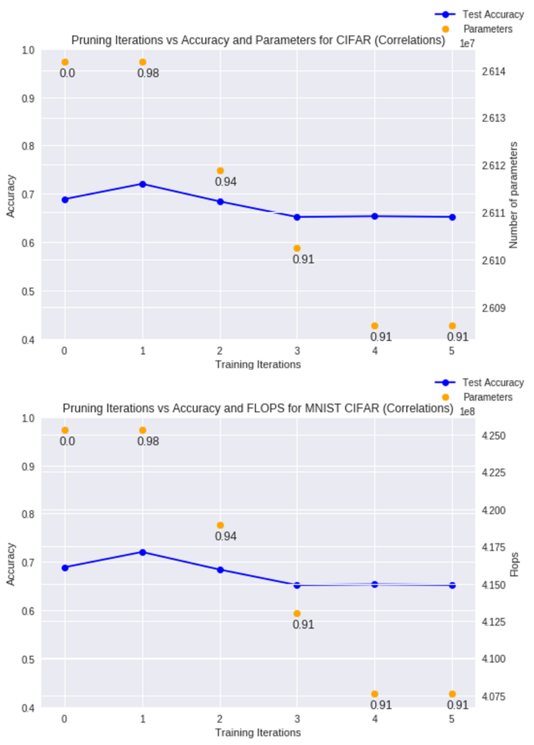

Figure 4. Pruning on CNN kernels with minimum L1 norm on CIFAR.

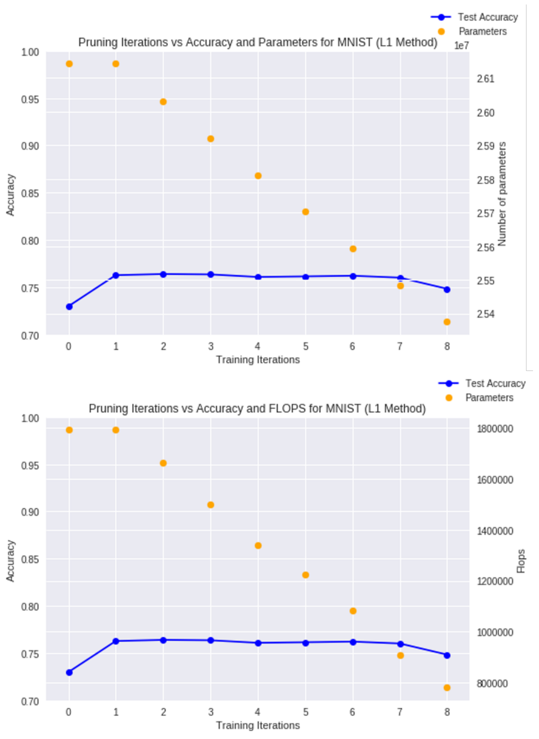

Figure 5. Pruning on highly correlated CNN kernels on CIFAR.

including the baseline is 12.49 seconds.

When pruning correlations of kernels, approximately 0.2% of the model parameters were pruned and the number of floating point operations reduced by 4% (Figure 4) with a gain in accuracy of 6.7% accuracy over 3 thresholds. The average wall clock time for testing through each iteration not including the baseline is 91.37 seconds.

The L1 norm method produced a significantly larger reduction in number of parameters over the correlation method over a greater number of iterations than correlation. Because the correlation process takes longer in terms of average wall clock time and has larger computations, the improvements observed take longer. However, the accuracy gain for correlation was higher again suggesting that the inter-dependent activity effects how effectively layers are pruned.

Additionally, the effect of pruning on the L1 norm is more visible when training with a more complex dataset such as CIFAR. There is an almost linear decrease in both the number of parameters and FLOPS as the number of training iterations increases, which suggests that magnitude based pruning effects larger datasets more adversely, while correlation methods can provide more stable compression.

7.4. Adding Neurons

We implemented the addition of neurons into a modified version of Algorithm 1. However, we were unable to exceed the initial accuracy of the network. The network appears to be stuck in a local minimum from the pretraining and the additional neurons cannot effectively improve the network.

We took a VGG-11 architecture on the CIFAR-10 dataset and drastically reduced the number of filters in each convolutional layer and tried to dynamically add neurons to increase accuracy after pretraining. We were unable to exceed initial accuracy and often could not recover initial accuracy.

8. Discussion

In our work, we explore multiple methods of pruning based on activation statistics. There are more advance metrics for pruning, that we may want to use. We can also use a combination of multiple metrics, so we can incorporate information from the activations and weights. There are also methods to encourage the network to be more prunable such as using a regularization term in the loss function.

We have a promising framework for the addition of neurons. Our goal is to develop an efficient algorithm for simultaneously adding and removing neurons from a network to efficiently train a compact neural network. More work needs to be done in order to effectively retrain networks after the addition of neurons. Some possible directions are adding stochastic noise to all of the weights, selectively retraining certain neurons, and using different initializations of new neurons. The development of effective expanding neurons can lead to a robust dynamic neuron creation and removal algorithm that unifies pruning and neuron expansion. We aim to expand our dynamic neuron deletion algorithm to Dynamic Adding and Removing of Neurons (DARN).

9. Conclusion

We were able to demonstrate significant results in pruning both fully connected and convolutional networks with significant decreases in model complexity while maintaining accuracy. Our results suggest that pruning based on the activations of neurons is more effective than on magnitude based metrics that localize network activity to single neurons. We can observe that inter-dependencies between kernels in layers exist and can be used effectively to reduce model size. Overall, our algorithm provides a solution to find an optimal acceptable accuracy along with a compressed model, which can be tuned for more generalized applications.

References

Alvarez, Jose M and Salzmann, Mathieu. Compression-aware training of deep networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 856–867. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6687-compression-aware-training-of-deep-networks.pdf.

Babaeizadeh, Mohammad, Smaragdis, Paris, and Campbell, Roy H. Noiseout: A simple way to prune neural networks. *CoRR*, abs/1611.06211, 2016. URL http://arxiv.org/abs/1611.06211.

Han, Song, Mao, Huizi, and Dally, William J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015a. URL http://arxiv.org/abs/1510.00149.

Han, Song, Pool, Jeff, Tran, John, and Dally, William J. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015b. URL http://arxiv.org/abs/1506.02626.

Hu, Yiming, Sun, Siyang, Li, Jianquan, Wang, Xingang, and Gu, Qingyi. A novel channel pruning method for deep neural network compression. *CoRR*, abs/1805.11394, 2018. URL http://arxiv.org/abs/1805.11394.

LeCun, Yann, Denker, John S., and Solla, Sara A. Optimal brain damage. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2*, pp. 598–605. Morgan-Kaufmann, 1990. URL http://papers.nips.cc/paper/250-optimal-brain-damage.pdf.

Lee, Jeongtae, Yoon, Jaehong, Yang, Eunho, and Hwang, Sung Ju. Lifelong learning with dynamically expandable networks. *CoRR*, abs/1708.01547, 2017.

Li, Hao, Kadav, Asim, Durdanovic, Igor, Samet, Hanan, and Graf, Hans Peter. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. URL http://arxiv.org/abs/1608.08710.