# Self-Optimizing Networks

Clement Lee

## 1 Introduction

Deep learning is a highly active area of research, and constant improvements are being delivered to current algorithms. However, many of the present improvements are minor tweaks to existing architectures, and hyperparameter optimization is left largely to human expertise. It is unfortunate that such amount of manual labor is necessary when developing networks. Most important discoveries are in large-scale architectural differences, but the process of verifying and choosing correct hyperparameters is a time-consuming process, especially as network training times can often reach weeks for larger and deeper networks. This project aims to simplify that process by allowing a network to determine some hyperparameters automatically; this would not necessarily reduce training time directly, but could alleviate the necessity for human intervention throughout the process.

## 2 Background & Related Work

In general, network accuracy is the primary goal of any deep learning system, and performance is a secondary concern. However, network complexity has increased at a pace that far outstrips the corresponding hardware improvements. Therefore, in recent years, network optimization has begun to pick up as a legitimate topic worthy of investigation.

### 2.1 Optimizing Networks

Deep networks are extremely computationally expensive; many techniques are simply far too complex to be deployed on smaller devices without cloud backing. In general, neural networks, while accomplishing astounding performance, are inefficient for their tasks. As explored in Squeezenet [5], it is possible to achieve competitive performance (often within 1% of classification accuracy) while reducing a neural network to merely 2% of the network's original size. Squeezenet utilizes a specialized design in order to minimize space usage, but does not focus on minimizing training time or classification speed. This is dependent on extremely careful tuning, and represents the kind of architecture that would be desirable if designed by an algorithm. As it stands, the Squeezenet architecture is a blend of various space-saving techniques, but the optimality of the specific parameters is derived from a great deal of experimentation.

Some other modern attempts at optimizing deep learning performance have focused on minimizing the individual parameter size; in this space, Binarynet [1] has found that it is possible to constrain network weights to binary values of $\pm 1$ without sacrificing significant performance. We want to avoid such methodology, since it is generally not applicable to GPU-accelerated training. These types of space reduction tend to be largely focused on CPU performance, in which smaller parameters can take advantage of extremely fast binary logic to achieve larger speedups.

### 2.2 Parameter Deletion

A basic method to parameter reduction is given in Han et al [2], who show that it is possible to use a simple heuristic (remove small weights) as a form of regularization to reduce network paramters dramatically. AlexNet is reduced to one-tenth of its original size with no discernable difference in performance. This is dependent on retraining of the remaining weights, which allows for the network to better handle the loss of the pruned weights.

Hu et al [3] provide a careful analysis of network parameters within VGG-16 and demonstrate that many of the weights are in fact effectively zero. They extend their analysis to the network neurons, and further show that the activations of many neurons across the data set are zero, allowing them to be removed from the network without loss in performance. Crucially, it is noted that the "existence [of these neurons] can only increase the chance of overfitting and optimization difficulty, both of which are harmful to the network."

However, no known results demonstrate an improvement in classification performance while simultaneously reducing the size of a large-scale convolutional network, as is potentially possible. Smaller-scale tests such as the optimal brain damage [7] have shown promise, as smaller networks are less likely to be trapped in local optima. Optimal Brain Damage represents an improvement on older heuristic-based methods for weight deletions, and was briefly explored by Lebedev et al [6] as a possible methodology to apply on convolutional networks. The main drawback of weight deletion algorithms is with respect to sparsity—while many parameters can be eliminated within the fully-connected layers easily, it is hard to remove individual convolutional kernel elements. Because convolutions are always specified by a matrix, removing a single value is of no benefit; instead, it is necessary to remove a whole matrix altogether, which can have a far more significant impact on the network as a whole.

## 3 Progress

Primarily, the work done to this point has been in investigating Residual Networks within Google's TensorFlow software; the image dataset of choice has been CIFAR-100. This dataset contains 60,000 $32 \times 32$ images, with 100 classes to differentiate. Work has been done to augment the dataset appropriately by standard methodologies.

### 3.1 Learning

Learning TensorFlow has been a difficult but necessary task; while many deep learning projects can rely on Keras or other simplified APIs, this project is heavily dependent on the ability to modify important parts of the training process. Understanding the computational model of TensorFlow and working through the examples has been a time-consuming but relatively rewarding process. The abstractions are largely immaterial to the project, and provide an easy way to utilize the GPU resources available.

### 3.2 Demonstrated Performance

Convolutional residual networks have been a central focus in this project, and a lot of work up to this point has been achieving performance on par with current methodologies. This is highly important to establish a baseline of performance, and ensure that all of the algorithms are running acceptably. Designing the residual network to match current methods has been crucial, and this has taken significant time, especially with limited training resources until recently. A residual network is currently being trained on the CIFAR dataset, and results will be forthcoming shortly.

## 4 Next Steps

A lot of work is still yet to be done, but much of the important setup has been finished. This is important in laying the groundwork for the more interesting components of this thesis.

### 4.1 Whole-Layer Pruning

Apart from the general sparse pruning discussed above, I intend to explore the possibility of removing whole layers beyond spatial dropout methodologies. By utilizing either a heuristic or analysis-based algorithm, a "efficiency" metric can be determined, and isolated to the layer level. This can potentially improve on dropout by reducing the size of the final network, allowing the learning capacity of the network to be more optimized.

### 4.2 Additive Methods

One primary goal of this thesis is to explore ways networks can be expanded, rather than just compressed, with the aim of automatically being able to architect networks. To my knowledge, no such work has been done with standard convolutional networks. A main advantage of using residual networks is that a zero layer can be reduced to the identity function; this promises the capability of adding and removing layers on the fly since the network has already been designed to handle this case better. The first steps of this process are to simply let the network randomly add layers and weights, potentially

followed by pruning. These cycles, in conjunction with the necessary retraining, might not have a significant effect on network performance, but would help give some intuition towards the efficiency and importance of each network component.

### 4.3 Structural Analysis

Another interesting point of investigation is to see whether such algorithms can create networks that are particularly optimized for specific problems. That is, by examining the structure remaining in the network after repeated cycles of these algorithms, we can explore whether the network can converge to a human-understandable internal structure. This is a possible result because some problems have known optimal network structures, so it would be ideal if the algorithm can begin to recreate a similar structure using iterative methods. This is, however, a much more complicated goal to achieve, as it involves not just analysis of network components in terms of performance, but also a sense of the structure that underlies the architecture.

### 4.4 General Date Outline

I intend to be modifying residual networks with layers within a week or two, especially with spring break approaching. Having a server set up with a GPU has been extremely productive, and results are coming in far faster because of it. Having a trained residual network locally will allow me to experiment with these weight algorithms that depend heavily on analyzing the network's activations over the full dataset. I hope to begin experimenting with additive methods towards the end of spring break, and to provide a methodology for enumerating the quality of a weight/layer/convolutional kernel soon after. I intend to leave the structural analysis component of this thesis as a stretch goal, to be completed if possible given the time remaining.

## References


[1] COURBARIAUX, M., AND BENGIO, Y. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[2] HAN, S., POOL, J., TRAN, J., AND DALLY, W. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems* (2015), pp. 1135–1143.

[3] HU, H., PENG, R., TAI, Y.-W., AND TANG, C.-K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[4] HUANG, G., SUN, Y., LIU, Z., SEDRA, D., AND WEINBERGER, K. Q. Deep networks with stochastic depth. In *European Conference on Computer Vision* (2016), Springer, pp. 646–661.

[5] IANDOLA, F. N., MOSKEWICZ, M. W., ASHRAF, K., HAN, S., DALLY, W. J., AND KEUTZER, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 1mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[6] LEBEDEV, V., AND LEMPITSKY, V. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2554–2564.

[7] LECUN, Y., DENKER, J. S., SOLLA, S. A., HOWARD, R. E., AND JACKEL, L. D. Optimal brain damage. In *NIPs* (1989), vol. 2, pp. 598–605.