Abstract

I investigate an algorithm for growing neural networks. The algorithm adds neurons to the network, then judges the quality of these neurons using an adding metric. The neuron with the best adding metric is kept and the rest are discarded. The algorithm shows promise in some regards but additional research is necessary.

Introduction

Neural network pruning can reduce accuracy by over 90%, without significant effects on accuracy (Mixter & Akoglu, 2020; Frankle & Carbin, 2018; Blalock et al., 2020). Nonetheless, training sparse networks or discovering network models that can achieve this is difficult (Frankle & Carbin, 2018). Machine learning engineers are often forced to make decisions about structure based on intuition or experience, or rely on considerable model trial and error. A possible solution is developing an algorithm which can discover an appropriate structure for a neural network during or before training.

One such class of algorithms are growth algorithms. Growth algorithms start with a small 'seed' network, and iteratively add neurons until an appropriate structure is found. Not only can this provide a method of finding a structure, but it can also provide gradient descent with a route away from plateaus and local minima. By starting with a smaller network, growth algorithms can also be quicker to train (Mixter & Akoglu, 2020; Vinod & Ghose, 1996).

A widely used method of structure determination includes pruning, which involves removing redundant parts of the neural network by analysis post training. This has been shown to be quite effective, and in some rare cases, can even improve performance (Blalock et al., 2020). Pruning, however, is a post training procedure, and its effectiveness can vary from problem to problem (Blalock et al., 2020).

Many growth algorithms have been proposed in more dated literature, but there are fewer examples of more recent implementations. The most applicable papers to this work are 'Growing Artificial Neural Networks' (Mixter & Akoglu, 2020) and 'Growing nonuniform feedforward networks for continuous mappings' (Vinod & Ghose, 1996).

'Growing Artificial Neural Networks' discusses a method of growing algorithms based on adding neurons and connecting them to the extreme members of the prior layer (where extreme means that the output is outside a given standard deviation of the mean). The paper receives good accuracy using LeNet-5 as a baseline on the MNIST dataset, with similar results to pruning. They do not however provide any discussion of convergence or mathematical backing for the algorithm.

On the other hand, 'Growing nonuniform feedforward networks for continuous mappings' provides a greedy algorithm which chooses the neuron addition that minimizes the least squares error. They provide thorough mathematical backing, but, because the paper is quite dated (1996), the computational backing is limited.

I hope to implement a growth algorithm that can take the better parts of these primary sources.

Implementation

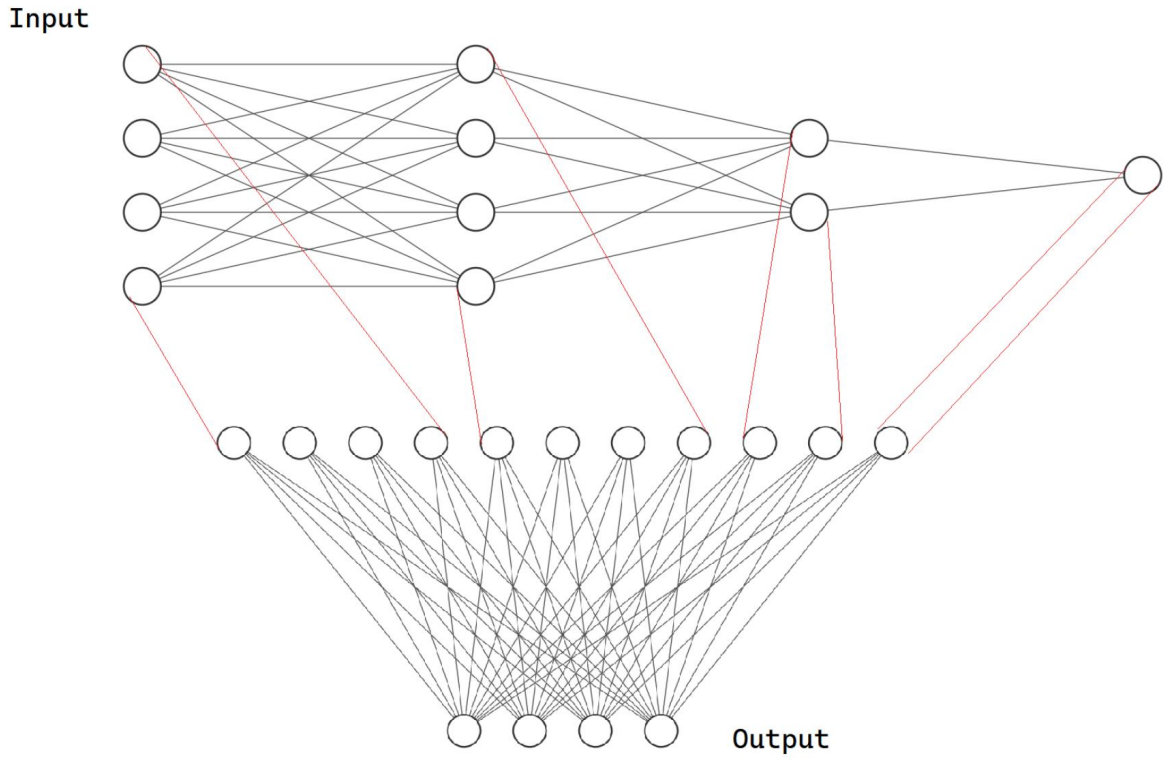
The algorithm will be tested on the MNIST dataset [], with 50000 training examples and 10000 test examples. Because the algorithm was quite different from regular gradient descent algorithms, particularly during backpropagation, the algorithm was implemented from scratch without machine learning libraries.

Forward propagation is performed as usual, but with the final layer taking all prior layers as input:

$$
\begin{gathered}
Z^{[1]} = W^{[1]T} X + b^{[1]} \\
A^{[1]} = \sigma^{[1]}(Z^{[1]}) \\
\cdots \\
Z^{[L-1]} = W^{[L-1]T} A^{[L-2]} + b^{[L-1]} \\
A^{[L-1]} = \sigma^{[L-1]}(Z^{[L-1]}) \\
Z^{[L]} = W^{[L]T} \begin{bmatrix} A^{[1]} & A^{[2]} & \cdots & A^{[L-1]} \end{bmatrix} + b^{[L]} \\
A^{[L]} = \sigma^{[L]}(Z^{[L]}) \\
J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log(\hat{y}_i)
\end{gathered}
$$

where $\sigma^{[k]}$ represents the activation function for the kth layer.

Taking in all prior outputs into the final layer reduces vanishing gradient problems, and allows us to not only add neurons to the network, but to also add layers, without adjusting the outputs of prior layers. The resulting network will look as follows. When a layer is added, it will be added to the end of sequential layers. Notice that fully connected neural networks are a subset of neural networks of this form.



For backpropagation we adjust the weights matrices as follows:

$$W^{[k]} = \begin{bmatrix} W_{11}^{[k]} & \cdots & W_{1j}^{[k]} \\ \vdots & \ddots & \\ W_{i1}^{[k]} & & W_{ij}^{[k]} \\ r_1^{[k]} & \cdots & r_j^{[k]} \end{bmatrix} \text{ if } k = 1,$$

$$W^{[k]} = \begin{bmatrix} W_{11}^{[k]} & \cdots & W_{1j}^{[k]} & 0 \\ \vdots & \ddots & & 0 \\ W_{i1}^{[k]} & & W_{ij}^{[k]} & 0 \\ r_1^{[k]} & \cdots & r_j^{[k]} & 0 \end{bmatrix} \text{ if } k > 1, k < L$$

$$W^{[k]} = \begin{bmatrix} W_{11}^{[k]} & \cdots & W_{1,c-1}^{[k]} & 0 & W_{1,c+1}^{[k]} & \cdots & W_{1j}^{[k]} & 0 \\ \vdots & \ddots & & 0 & \vdots & \ddots & & 0 \\ W_{i1}^{[k]} & & W_{i,c-1}^{[k]} & 0 & W_{i,c+1}^{[k]} & & W_{ij}^{[k]} & 0 \end{bmatrix} \text{ if } k = L$$

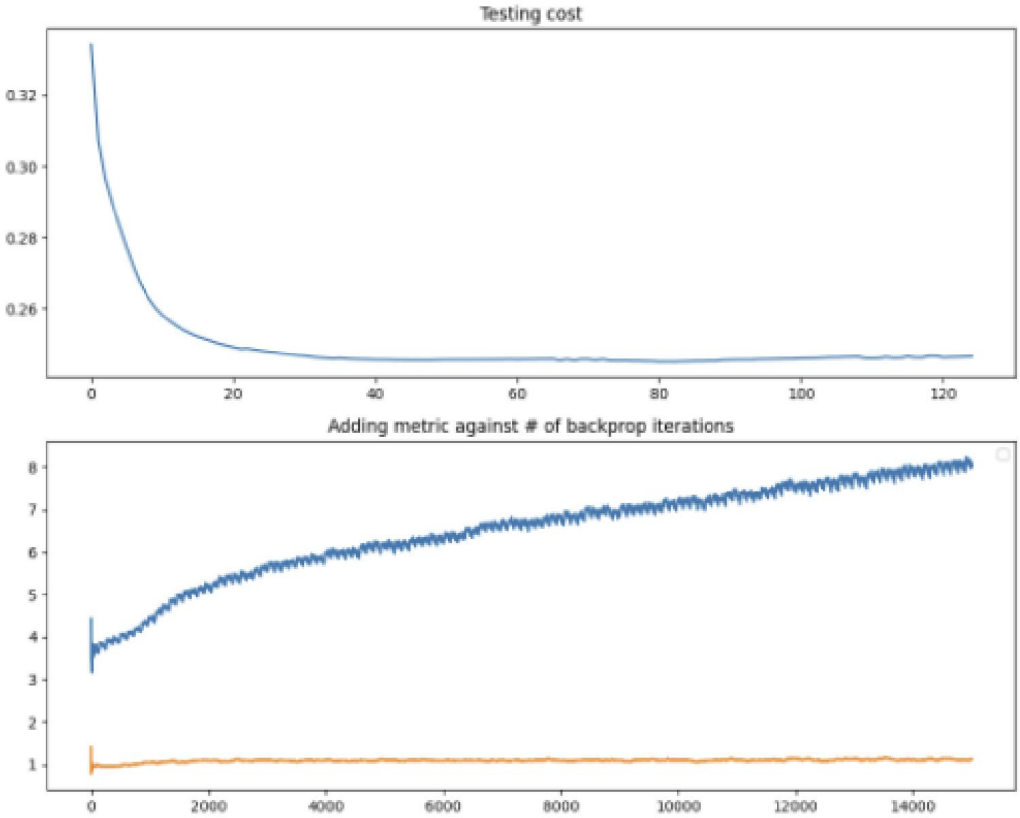
Where i, j give the number of outputs and inputs respectively, $r^{[k]}$ are the weights for a new neuron, and c represents the index of the newly generated neuron output. Note, while only one c is shown here, there will be as many zeroed columns as there are prior layers.

These weight matrices are set up in a way that the final output will not change (because the zeroes will remove the outputs of the new neurons), but when backpropagation is performed, the magnitude of $\delta W_{a,j+1}^{[k]}$ will give a measure of how much to change the zeroes by. In other words, a measure of how much input the newly generated neurons should have. This value is what is used as an adding metric for additions to the network. When the adding metric passes the adding threshold hyperparameter, that neuron is kept.

Results

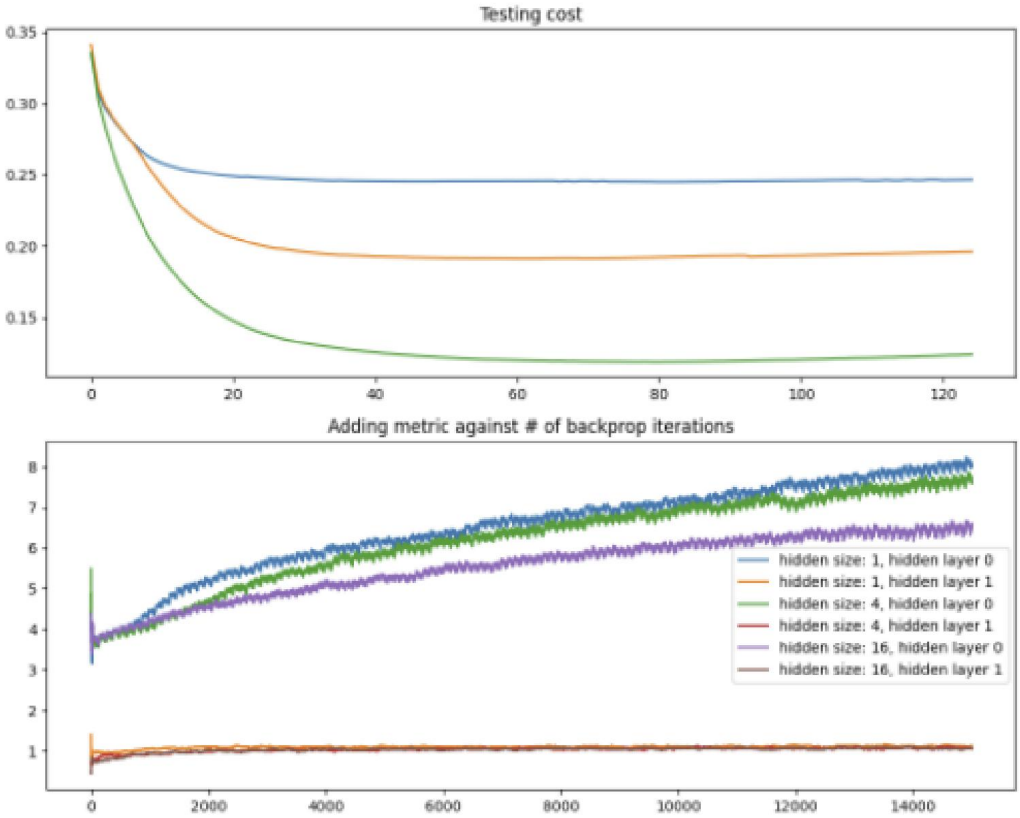
Before using the adding metric to add neurons, we must check that the metric behaves appropriately.

First, it was tested whether the metric correctly increased as the network trained, to show the need for additional neurons when the networks' expressibility is not sufficient to continue improving. A model with layer sizes $(785 \times 1), (786 \times 10)$ was trained with the MNIST dataset was trained (recall that the output layer takes all prior layers as well as inputs as an input). The model was not allowed to grow, but the metric was measured as it trained for 125 epochs.



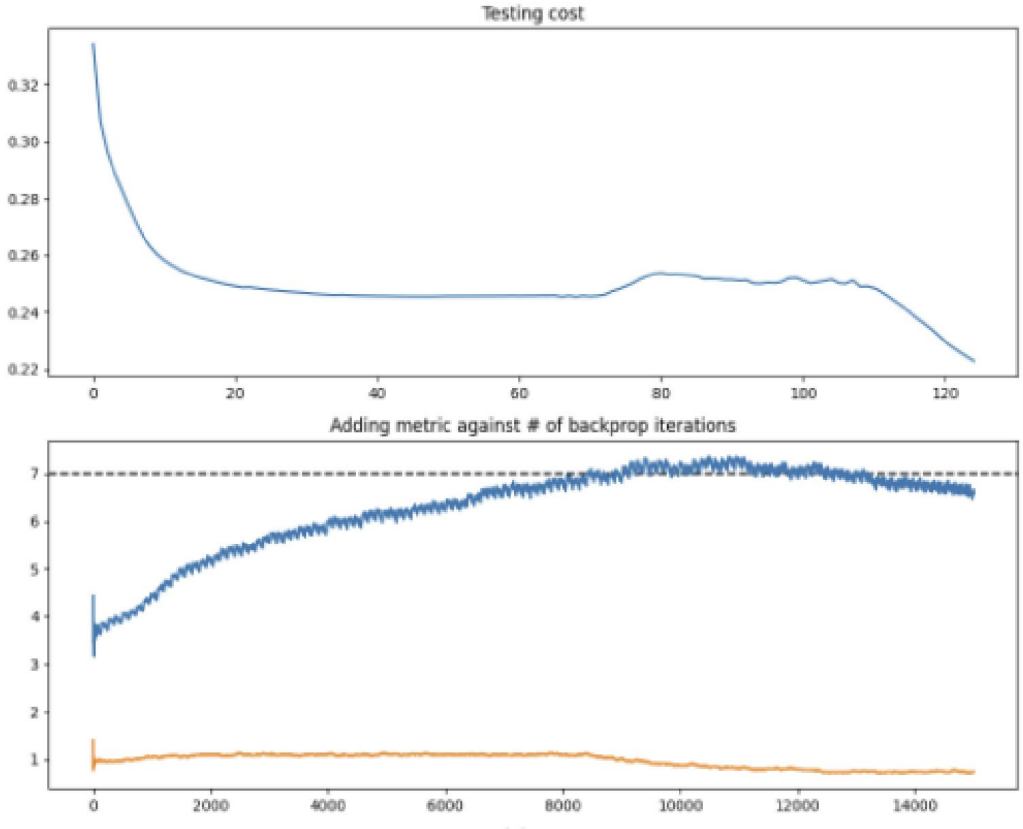
As you can see in the graphs, the metric appropriately grew. We can also see that the metric for the first layer was much higher than for the second layer (an added layer that would be inserted between the currently existing layers). This means that the algorithm judges a model with layers $(785 \times 2), (787 \times 10)$ to be superior to a model with layers $(785 \times 1), (2 \times 1), (787 \times 10)$, which is what we would expect intuitively. In this manner, the metric behaves appropriately.

Another behavior we wish the metric to show, is that it decreases as model complexity increases - as a model gets larger, it will need less added neurons. Three models were trained with the MNIST dataset, with layers of size $(785 \times 1), (786 \times 10)$ for the first model, $(785 \times 4), (789 \times 10)$ for the second model, and $(785 \times 16), (802 \times 10)$ for the final model.



The graphs show that the metric did show this property.

Similar to the prior metric behavior, the metric should also decrease when additional neurons are added during training. To test this, a model with layers $(785 \times 1), (786 \times 10)$ was used as a seed network, then the network was trained. Whenever a layers' metric reached a threshold value, a neuron was added.



We can see that the metrics appropriately flattened when the metric reached the threshold value. We also see that this was associated with a drop in testing cost, which suggests the possibility of performance improvements.

Conclusion

The constructed metric showed signs of being a good candidate for an adding metric, and behaved as desired.

Effects on training size / time

However, there are some problems to be addressed with this approach. Firstly, a method of appropriately adding neurons must be decided. Possible approaches include adding neurons when the metric reaches a threshold or adding the neuron with the highest adding metric every k iterations. The first option was what was implemented here, but this offers the problem of choosing a good adding threshold, which in practice was quite difficult to do.

To find an appropriate implementation for this algorithm, an investigation beyond the scope of this project is required. This would include a thorough mathematical investigation and proof of convergence, as well as more computational testing. It would also be interesting to see this method applied to finding sparse networks, that is, not using all inputs when adding a neuron, and seeing how this affects performance. Despite the further investigation necessary, I believe the work done here has some merit.

Bibliography

Mixter, J., & Akoglu, A. (2020). Growing Artificial Neural Networks. *arXiv preprint arXiv:2006.06629*.

Vinod, V. V., & Ghose, S. (1996). Growing nonuniform feedforward networks for continuous mappings. *Neurocomputing*, *10*(1), 55-69.

Blalock, D., Ortiz, J. J. G., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning?. *arXiv preprint arXiv:2003.03033*.

Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

MacLeod, C., & Maxwell, G. M. (2001). Incremental evolution in ANNs: Neural nets which grow. *Artificial Intelligence Review*, *16*(3), 201-224.