THE INCREDIBLE SHRINKING NEURAL NETWORK: PRUNING TO OPERATE IN CONSTRAINED MEMORY ENVIRONMENTS

Nikolas Wolfe, Aditya Sharma, Bhikhsa Raj
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{nwolfe, bhiksha}@cs.cmu.edu, adityasharma@cmu.edu

ABSTRACT

We propose and investigate several methods for pruning neural networks to operate in constrained memory environments such as mobile or embedded devices. We first evaluate a simple pruning technique using first-order derivative approximations of the gradient of each neuron in an optimally trained network, and turning off those neurons which contribute least to the output of the network. We then show the limitations of this method by comparing its pruning decisions of this method against an exhaustive brute force approach to determining the change in error resulting from the removal of a given neuron. We attempt to improve on this using a second-order derivative approximation. We also explore the correlation between neurons in a trained network and attempt to improve our choice of candidate neurons for removal to account for faults that can occur from the removal of a single neuron at a time. We argue that this method of pruning allows for the optimal tradeoff in network size versus accuracy in order to operate within the memory constraints of a particular device or application environment.

1 INTRODUCTION

Neural network pruning algorithms were first popularized by Sietsma & Dow (1988) as a mechanism to determine the proper size network required to solve a particular problem. To this day, network design and optimal pruning remain inherently difficult tasks. For problems which cannot be solved using linear threshold units alone, Baum & Haussler (1989) demonstrate there is no way to precisely determine the appropriate size of a neural network a priori given any random set of training instances. Using too few neurons inhibits learning, and so in practice it is common to attempt to over-parameterize networks initially using a large number of hidden units and weights. However, as Chauvin (1990) writes, this approach can lead to over-fitting as the network's unnecessary free parameters start to latch on to idiosyncrasies in the training data.

Pruning algorithms, as comprehensively surveyed by Reed (1993), are a useful set of heruristics designed to identify and remove network parameters which do not contribute significantly to the output of the network and potentially inhibit generalization performance. At the time of Reed's writing, reducing network size was also a practical concern, as smaller networks are preferable in situations where computational resources are scarce. In this paper we are particularly concerned with application domains in which space is limited and network size constraints must be imposed with minimal impact on performance.

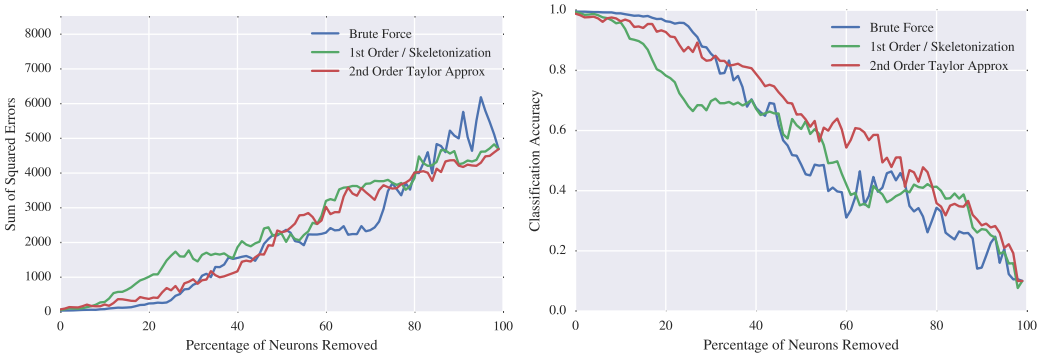

Figure 1: Degradation in squared error (left) and classification accuracy (right) after pruning a single-layer network trained on MNIST using a single-pass overall ranking procedure (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

2 Experimental Results

2.1 MNIST Dataset Results

2.1.1 Pruning A 1-Layer Network: Single-Pass Ranking

We first present the results for a single-layer neural network in Figure 1, using a one-time computation of the overall ranking of all neurons to make our pruning decisions. After training, each neuron is assigned a permanent ranking based on three criteria: A brute force "ground truth" ranking, and two approximations of this ranking using first and second order Taylor estimations of the change in network output error resulting from the removal of each neuron. (We note that this algorithm is intentionally naive and is used for comparison only. Its performance should be expected to be poor.)

An interesting observation here is that with only a single layer, no method for ranking the neurons in the network with a single pass emerges superior, indicating that the 1st and 2nd order methods are actually reasonable approximations of the brute force method under certain conditions. Of course, this method is still quite bad in terms of the rate of degradation of the classification accuracy and in practice we would likely follow Mozer & Smolensky (1989)'s advice to retrain after each neuron removal in order to allow remaining neurons to adjust and compensate for network faults triggered by poor pruning decisions. The purpose of the present investigation, however, is to demonstrate how much of a trained network can be theoretically removed without altering the network's learned parameters in any way. Once again, we only present this method as a basis for comparison and we will investigate more promising methods in greater detail in later sections.

2.1.2 Pruning A 1-Layer Network: Re-Ranking After Each Removal

In Figure 2 we present our results using an iterative re-ranking procedure in which all remaining neurons are re-ranked after each successive neuron is switched off. We compute the same brute force rankings and Taylor series approximations of error deltas over the remaining active neurons in the network after each pruning decision. This is intended to account for the effects of canceling interactions between neurons.

tl;dr: 1 layer is grrrreat for brute force! Clearly 2nd order method is consistently better than 1st order method.

tl;dr: Brute force method is amazing here! 1 layer makes things a lot easier, and a 2nd order method can do okay as well, but can still be improved a lot. 60

2.1.3 Visualization of Error Surface & Pruning Decisions

As explained in the methodology section, these graphs are a visualization the error surface of the network output with respect to the neurons chosen for removal using each algorithm, represented in intervals of 10 neurons. In each graph, the error surface of the network output is displayed in log

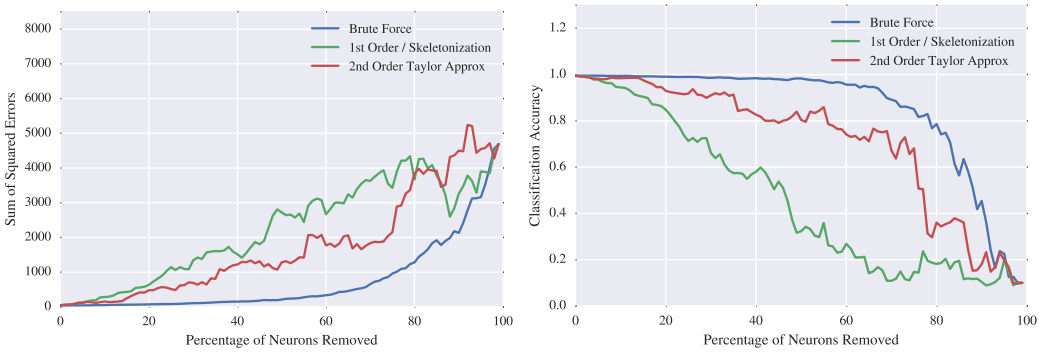

Figure 2: Degradation in squared error (left) and classification accuracy (right) after pruning a single-layer network trained on MNIST using an iterative re-ranking procedure (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

space (left) and in real space (right) with respect to each candidate neuron chosen for removal. We create these plots during the pruning exercise by picking a neuron to switch off, and then multiplying its output by a scalar gain value α which is adjusted from 0.0 to 10.0 with a step size of 0.001. When the value of α is 1.0, this represents the unperturbed neuron output learned during training. Between 0.0 and 1.0, we are graphing the literal effect of turning the neuron off ($\alpha = 0$), and when $\alpha > 1.0$ we are simulating a boosting of the neuron's influence in the network, i.e. inflating the value of its outgoing weight parameters.

We graph the effect of boosting the neuron's output to demonstrate that for certain neurons in the network, even doubling, tripling, or quadrupling the scalar output of the neuron has no effect on the overall error of the network, indicating the remarkable degree to which the network has learned to ignore the value of certain parameters. In other cases, we can get a sense of the sensitivity of the network's output to the value of a given neuron when the curve rises steeply after the red 1.0 line. This indicates that the learned value of the parameters emanating from a given neuron are relatively important, and this is why we should ideally see sharper upticks in the curves for the later-removed neurons in the network, that is, when the neurons crucial to the learning representation start to be picked off.

Remember that lower is better in terms of the height of the curve and minimal horizontal change between the vertical red line at 1.0 (neuron *on*, $\alpha = 1.0$) and 0.0 (neuron *off*, $\alpha = 0.0$) is indicative of a good candidate neuron to prune, i.e. there will be minimal effect on the network output when the neuron is removed. The network architecture in this case consisted of 1 layer, 100 neurons, 10 outputs, logistic sigmoid activations, and a starting test accuracy of 0.998.

Brute Force Method tl;dr: Notice how low to the floor and flat most of these curves are. It's only until the 90th removed neuron that we see a higher cure with a more convex shape (clearly a more influential piece of the network)

1st Order Method tl;dr: Most choices seem to have flat or negatively sloped curves, indicating that the first order approx seems to be pretty good, but examining the brute force choices shows they could be better.

2nd Order Method tl;dr: Looks much more similar to the Brute Force method choices, though clearly not as good (they're more spread out). Notice the difference in convexity between the 2nd and 1st order method choices. It's clear that the first order method is fitting a line and the 2nd order method is fitting a parabola in their approximation.

2.1.4 Pruning A 2-Layer Network: Single-Pass Ranking

Figure 6 shows the result of using a one-time computation of the overall ranking of all neurons to make pruning decisions for a two-layer network. The ranking procedure is identical to the one used to generate Figure 1. (We again note that this algorithm is intentionally naive and is used for comparison only. Its performance should be expected to be poor.)

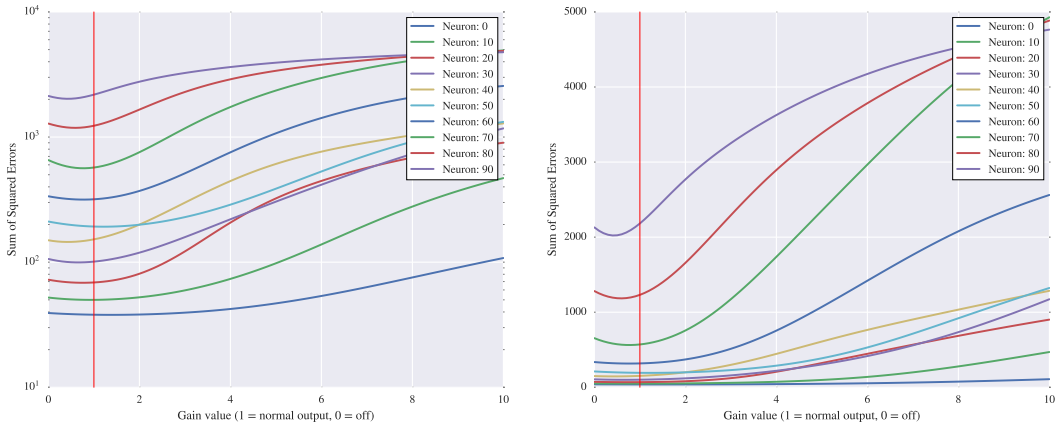

Figure 3: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

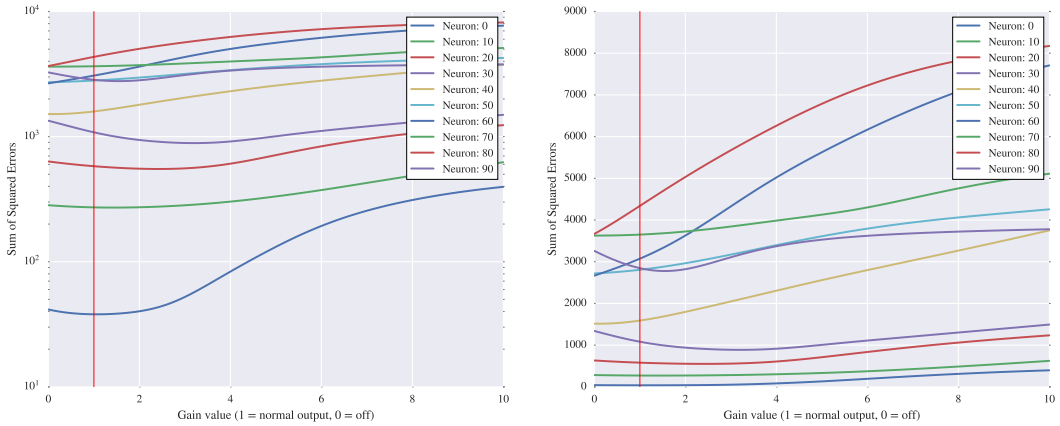

Figure 4: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

tl;dr: Unsurprisingly, a 2-layer network is harder to prune because a single overall ranking will never capture the interdependencies between neurons in different layers. It makes sense that this is much worse than the performance on the 1-layer network, even if this method is already known to be bad, and we'd likely never use it in practice.

2.1.5 Pruning A 2-Layer Network: Re-Ranking After Each Removal

Figure 7 shows the results results using an iterative re-ranking procedure on a two-layer network. We compute the same brute force rankings and Taylor series approximations of error deltas over the remaining active neurons in the network after each pruning decision used to generate Figure 2. Again, this is intended to account for the effects of canceling interactions between neurons.

tl;dr: clearly a higher set of curves, indicating that it becomes harder to remove neurons 1-by-1 with a deeper network (which makes sense because the neurons have more interdependencies in a deeper network), but we see an overall better performance with 2nd order method vs. 1st order, except for the first 20

tl;dr: Shows the clear potential of an ideal pruning technique, and how inconsistent 1st and 2nd order methods can be

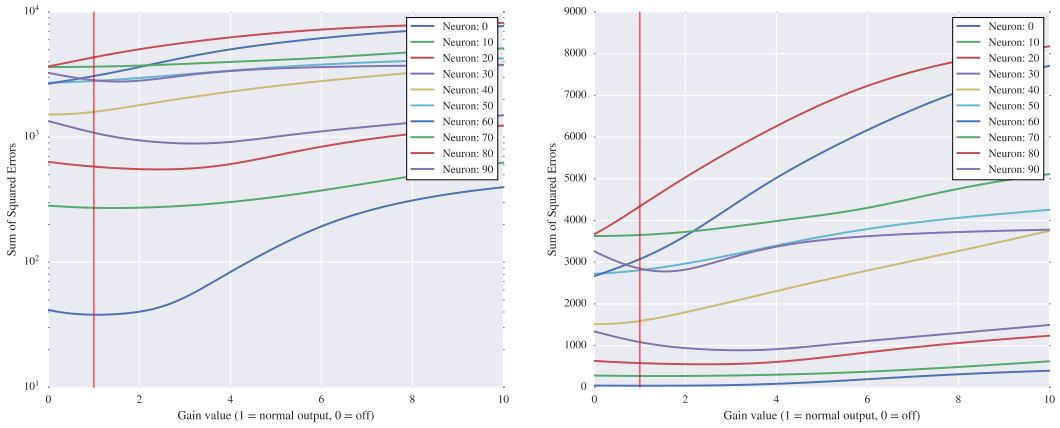

Figure 5: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

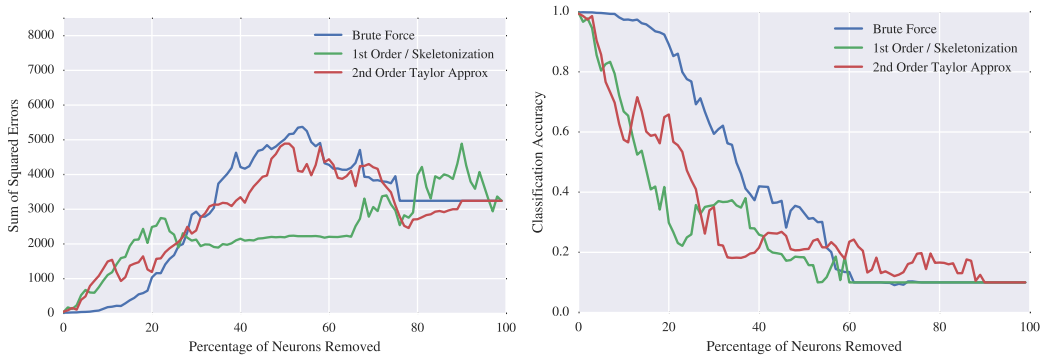

Figure 6: Degradation in squared error (left) and classification accuracy (right) after pruning a 2-layer network trained on MNIST using a single-pass overall ranking procedure (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

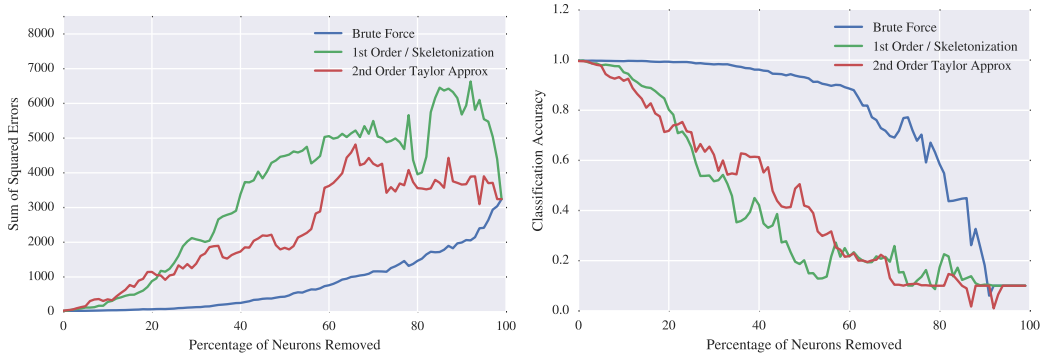

Figure 7: Degradation in squared error (left) and classification accuracy (right) after pruning a 2-layer network trained on MNIST using an iterative re-ranking procedure (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

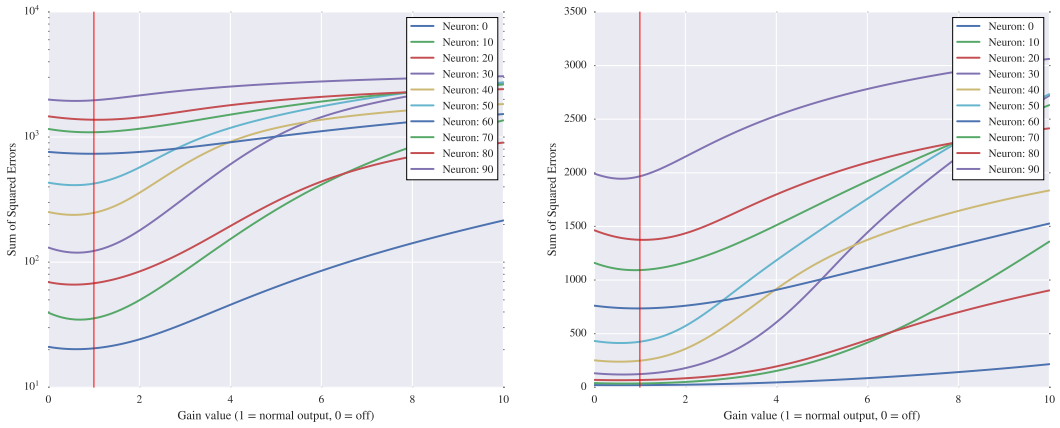

Figure 8: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

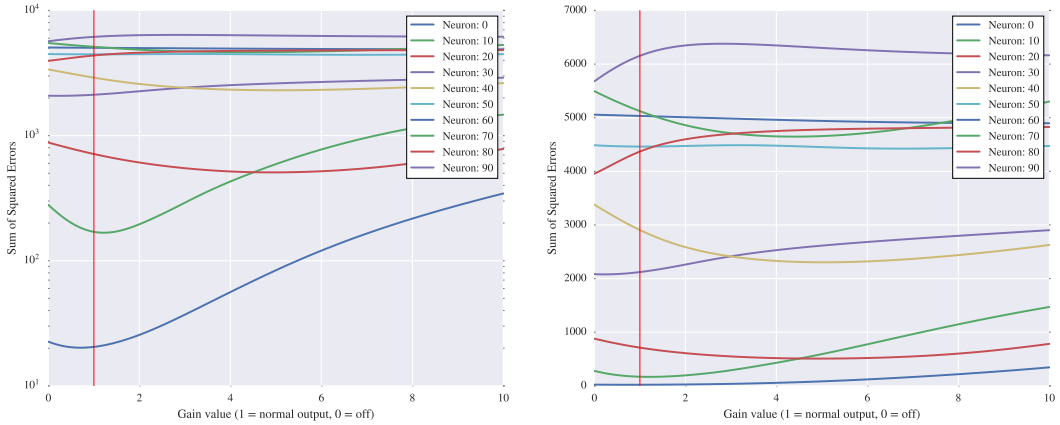

Figure 9: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

2.1.6 Visualization of Error Surface & Pruning Decisions

These graphs are a visualization the error surface of the network output with respect to the neurons chosen for removal using each algorithm, represented in intervals of 10 neurons.

Brute Force Method tl;dr: It's clear why these neurons were chosen, their graphs clearly show little change when neuron is removed, are mostly near the floor, and show convex behavior of error surface, which argues for the rationalization of using 2nd order methods to estimate difference in error when they are turned off

1st Order Method tl;dr: Drawing flat line at the point of each neurons intersection with the red vertical line (no change in gain) shows that the 1st derivative method is actually accurate for estimation of change in error in these cases, but still ultimately leads to poor decisions.

2nd Order Method tl;dr: Clearly these are not overtly poor candidates for removal (error doesn't change much between 1.0 & zero-crossing left-hand-side), but could be better (see brute force)

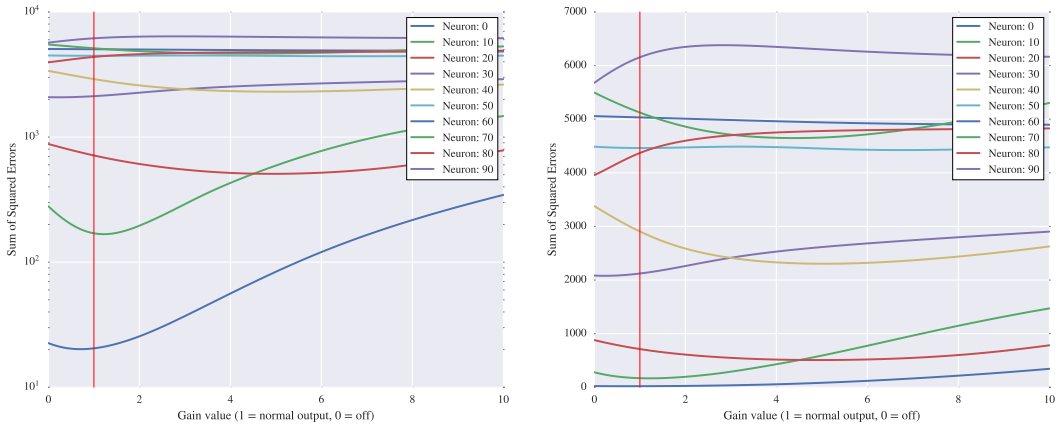

Figure 10: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

3 Conclusions & Future Work

3.1 Conclusions

2nd-Order Method Consistently better than 1st-Order, but still pretty bad Both Methods fail miserably beyond the 1st layer in terms of classification accuracy For the 2-layer networks the 1st/2nd order methods do not appear to have any apparent preference for the 1st or 2nd layer in pruning decisions. Brute force method pulls primarily from the outer layer at the beginning – clearly the 1st and 2nd order methods will suffer from their difference in this regard Performance is worse if initial network is not trained to near-perfect accuracy, but pruning in this case typically improves MSE (not accuracy) for the first 5-10Brute force method is surprisingly good and indicates the degree to which extra parameters are a waste Sum of squared errors is a decent proxy for classification accuracy, but classification accuracy is a clearer signal of performance in such problems If Brute Force method were made comutationally viable, could be used to prune 40-80This empirically shows that the learning representation is not evenly distributed over neurons, indicating NO advantage to larger networks if unnecessary Confirms Mozer & Smolensky's conclusion in 1989: Learning representation NOT distributed! Neurons clearly group together to cancel out each other's effects (see spikes in classification accuracy drop-off graphs), which means neural networks use a few neurons to learn the function approximation, and the rest cooperate to cancel out each other's effects!

* second derivative method works better * still not very good * retraining is a good idea * speedup of brute force method * which layer gets pruned first

3.2 Future Work

Future experiments (NOT for this paper): Try with deeper networks (3/4/5 layers) to see if the performance degrades as we should expect Speed up brute force method through parallelization? How do results improve when we re-train after pruning 1-n neurons in a row? (expectation: smooooooooth curve for brute force method) Examine whether Cascade Correlation arrives at the same minimal number of neurons Use cross entropy to make pruning decision for classification problems Try a regression task on a real-world dataset, e.g. housing prices, stock market, etc.

REFERENCES

Baum, Eric B and Haussler, David. What size net gives valid generalization? *Neural computation*, 1(1):151–160, 1989.

Chauvin, Yves. Generalization performance of overtrained back-propagation networks. In *Neural Networks*, pp. 45–55. Springer, 1990.

Mozer, Michael C and Smolensky, Paul. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pp. 107–115, 1989.

Reed, Russell. Pruning algorithms-a survey. *Neural Networks, IEEE Transactions on*, 4(5):740–747, 1993.

Sietsma, Jocelyn and Dow, Robert JF. Neural net pruning-why and how. In *Neural Networks, 1988., IEEE International Conference on*, pp. 325–333. IEEE, 1988.