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

Nikolas Wolfe, Aditya Sharma, Bhiksha Raj
School of Computer Science
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 of a 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 heuristics 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.

2 RELATED WORK

Neural network over-fitting is fundamentally a problem arising from the use of too many free parameters. Regardless of the number of weights used in a given network, as Segee & Carter (1991) assert, the representation of a learned function approximation is almost never evenly distributed over the hidden units, and the removal of any single hidden unit at random can actually result in a total network fault. Mozer & Smolensky (1989b) suggest that only a subset of the hidden units in a neural network actually latch on to the invariant or generalizing properties of the training inputs, and the rest learn to either mutually cancel each other out or begin over-fitting to the noise in the data. Determining which elements are unnecessary and removing them outright is therefore a well-founded approach to improving network generalization, and simultaneously provides a way to reduce their size in memory.

The generalization performance of neural networks has been well studied, and apart from pruning algorithms many heuristics have been used to avoid overfitting, such as dropout (Srivastava et al. (2014)), maxout (Goodfellow et al. (2013)), and cascade correlation (Fahlman & Lebiere (1989)), among others. However, these algorithms do not explicitly prioritize the reduction of network memory footprint as a part of their optimization criteria per se, (although in the opinion of the authors, Fahlman's cascade correlation architecture holds great promise in this regard.) Computer memory size and processing capabilities have improved so much since the introduction of pruning algorithms in the late 1980s that space complexity has become a relatively negligible concern. The proliferation of cloud-based computing services has furthermore enabled mobile and embedded devices to leverage the power of massive data and computing centres remotely. In this domain, however, it is also reasonable to suggest that certain performance-critical applications running on low-resource devices could benefit from the ability to use neural networks locally.

At present there are few (if any) mechanisms specifically designed to shrink neural networks down in order to meet an externally imposed constraint on byte-size in memory. Without explicitly removing parameters from the network, one could use weight quantization to reduce the number of bytes used to represent each weight parameter, as investigated by Balzer et al. (1991), Dundar & Rose (1994), and Hoehfeld & Fahlman (1992). Of course, this method can only reduce the size of the network by a factor proportional to the byte-size reduction of each weight parameter.

Another method which has recently gained popularity is using the singular values of a trained weight matrix as basis vectors from which to derive a compressed hidden layer.

* describe SVD stuff

If we wanted to continually shrink a network to its absolute minimal size in an optimal manner, we might accomplish this using any number of off-the-shelf pruning algorithms, such as Skeletonization (Mozer & Smolensky (1989a)), Optimal Brain Damage (LeCun et al. (1989)), or later variants such as Optimal Brain Surgeon (Hassibi & Stork (1993)). In fact, we borrow much of our inspiration from these antecedent algorithms, with one major variation.

The aforementioned strategies all focus on the targeting and removal of *weight* parameters. Scoring and ranking individual weight parameters in a large network computationally expensive, and generally speaking the removal of a single weight from a large network is a drop in the bucket in terms of reducing a network's core memory footprint. We therefore train our sights on the ranking and removal of entire neurons along with their associated weight parameters. We argue that this is more efficient computationally as well as practically in terms of quickly reaching a target reduction in memory size. Our approach also attacks the angle of giving downstream applications a realistic expectation of the minimal increase in error resulting from the removal of a specified percentage of neurons from a trained network. Such trade-offs are unavoidable, but performance impacts can be limited if a principled approach is used to find candidate neurons for removal.

* brief overview of approach

3 METHODOLOGY

The general approach taken to prune an optimally trained neural network in the present work is to create a ranked list of all the neurons in the network based off of one of the 3 proposed ranking criteria: Brute Force approximation (which we use as our ground truth), linear approximation and quadratic approximation of the neuron's impact on the overall performance of the network. We then test the effects of removing neurons sequentially on the accuracy of the network. These tests can be found in the Experiments section. Next, we propose our algorithm with 2 variants

3.1 Brute Force Removal Approach

This is perhaps the most naive yet the most accurate method for pruning the network. It is also the slowest and hence unusable on large-scale neural networks with thousands of neurons. The idea is to manually check the effect of every single neuron on the output. This is done by running a forward propagation on the validation set K times (where K is the total number of neurons in the network), turning off exactly one neuron each time (keeping all other neurons active) and noting down the change in error. Turning a neuron off can be achieved by simply setting its output to 0. This results in all the outgoing weights from that neuron being turned off. This change in error is then used to generate the ranked list.

3.2 Taylor Series Representation of Error

Let us denote the total error from the optimally trained neural network for any given validation dataset by E. E can be seen as a function of O, where O is the output of any general neuron in the network. This error can be approximated at a particular neuron's output (say O_k) by using the 2nd order Taylor Series as,

$$\hat{E}(O) \approx E(O_k) + (O - O_k) \cdot \left.\frac{\partial E}{\partial O}\right|_{O_k} + 0.5 \cdot (O - O_k)^2 \cdot \left.\frac{\partial^2 E}{\partial O^2}\right|_{O_k}, \tag{1}$$

When a neuron is pruned, its output O becomes 0. From equation 1, the contribution $E(0)$ of this neuron, then becomes:

$$\hat{E}(0) \approx E(O_k) - O_k \cdot \left.\frac{\partial E}{\partial O}\right|_{O_k} + 0.5 \cdot O_k^2 \cdot \left.\frac{\partial^2 E}{\partial O^2}\right|_{O_k} \tag{2}$$

Replacing O by O_k in equation 1 shows us that the error is approximated perfectly by equation 1 at O_k. Using this and equation 2 we get:

$$\Delta E_k = \hat{E}(0) - \hat{E}(O_k) = -O_k \cdot \left.\frac{\partial E}{\partial O}\right|_{O_k} + 0.5 \cdot O_k^2 \cdot \left.\frac{\partial^2 E}{\partial O^2}\right|_{O_k}, \tag{3}$$

where ΔE_k is the change in the total error of the network when exactly one neuron (k) is turned off.

3.3 Linear Approximation Approach

We define the following network terminology here which will be used in this section and all subsequent sections unless stated otherwise. Figure 1 can be used as a reference to the terminology defined here:

$$E = \frac{1}{2}\sum_i (o_i^{(0)} - t_i)^2 \quad o_i^{(m)} = \sigma(x_i^{(m)}) \quad x_i^{(m)} = \sum_j w_{ji}^{(m)} o_j^{(m+1)} \quad c_{ji}^{(m)} = w_{ji}^{(m)} o_j^{(m+1)} \tag{4}$$

Superscripts represent the index of the layer of the network in question, with 0 representing the output layer. E is the squared-error network cost function. $o_i^{(m)}$ is the ith output in layer m generated by the activation function σ, which in this paper is is the standard logistic sigmoid. $x_i^{(m)}$ is the weighted sum of inputs to the ith neuron in the mth layer, and $c_{ji}^{(m)}$ is the contribution of the jth neuron in the $(m+1)$th layer to the input of the ith neuron in the mth layer. $w_{ji}^{(m)}$ is the weight between the jth neuron in the $(m+1)$th layer and the ith neuron in the mth layer.

We can use equation 3 to get the linear error approximation of the change in error due to the kth neuron being turned off and represent it as ΔE_k^1 as follows:

$$\Delta E_k^1 = -o_k \cdot \left. \frac{\partial E}{\partial o_j^{(m+1)}} \right|_{o_k} \tag{5}$$

The derivative term above is the first-order gradient which represents the change in error with respect to the output of a given neuron o_j in the $(m+1)$th layer. This term can be collected during back-propagation. The derivative term above can be calculated as follows:

$$\frac{\partial E}{\partial o_j^{(m+1)}} = \sum_i \frac{\partial E}{\partial x_i^{(m)}} \cdot w_{ji}^{(m)} \tag{6}$$

The full step-by-step mathematical derivation of the above equation can be found in the appendix.

3.4 Quadratic Approximation Approach

As seen in equation 3, ΔE_k which can now be represented as ΔE_k^2 is the quadratic approximation of the change in error due to the kth neuron being turned off. The quadratic term in equation 3 requires some discussion which we provide here. A more detailed and step-by-step mathematical derivation can be found in the appendix.

Let us reproduce equation 3 in our new terminology here:

$$\Delta E_k^2 = -o_k \cdot \left. \frac{\partial E}{\partial o_j^{(m+1)}} \right|_{o_k} + 0.5 \cdot o_k^2 \cdot \left. \frac{\partial^2 E}{\partial {o_j^{(m+1)}}^2} \right|_{o_k} \tag{7}$$

This equation involves the second-order gradient which represents the second-order change in error with respect to the output of a given neuron o_j in the $(m+1)$th layer. This term can be generated by performing back-propagation using second derivatives. A full derivation of the second derivative back-propagation can be found in the appendix. Some results from the derivation are quoted here. The second-order derivative term can be represented as:

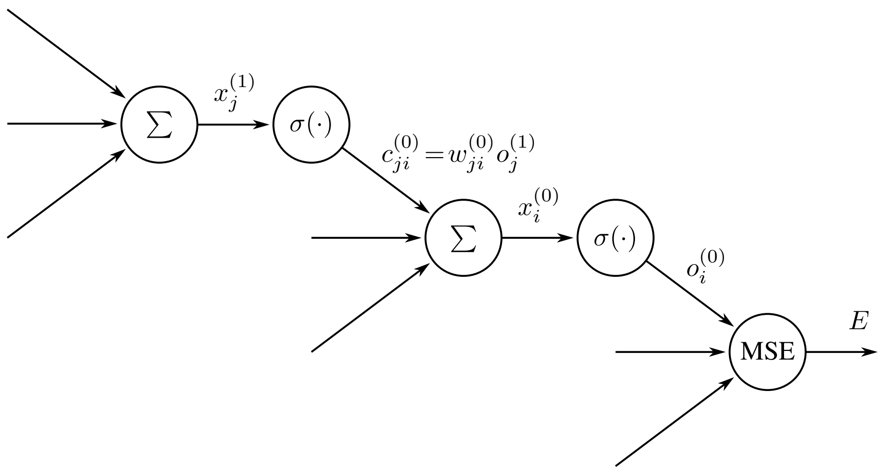

Figure 1: A computational graph of a simple feed-forward network illustrating the naming of different variables, where $\sigma(\cdot)$ is the nonlinearity, MSE is the mean-squared error cost function and E is the overall loss.

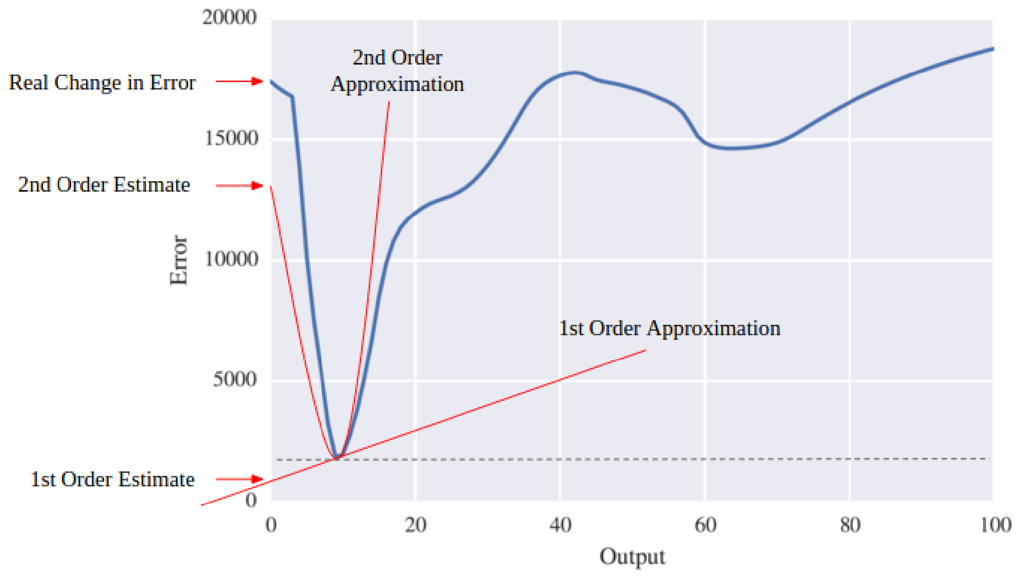

Figure 2: The intuition behind neuron pruning decision.

$$\frac{\partial^2 E}{\partial {o_j^{(m+1)}}^2} = \sum_i \frac{\partial^2 E}{\partial {c_{ji}^{(m)}}^2} \left(w_{ji}^{(m)}\right)^2 \tag{8}$$

Here,$c_{ji}^{(m)}$ is one of the component terms of $x_i^{(m)}$, as follows from the equations in 4. Hence, it can be easily proved that (full proof in appendix):

$$\frac{\partial^2 E}{\partial {c_{ji}^{(m)}}^2} = \frac{\partial^2 E}{\partial {x_i^{(m)}}^2} \tag{9}$$

Now, the value of $x_i^{(m)}$ can be easily calculated through the steps of the second-order back-propagation using Chain Rule. The full derivation can again, be found in the appendix.

$$\frac{\partial^2 E}{\partial {x_i^{(m)}}^2} = \frac{\partial^2 E}{\partial {o_i^{(m)}}^2} \left(\sigma'\left(x_i^{(m)}\right)\right)^2 + \frac{\partial E}{\partial o_i^{(m)}} \sigma''\left(x_i^{(m)}\right) \tag{10}$$

3.5 Proposed Pruning Algorithm

Figure 2 shows a random error function plotted against the output of any given neuron. Note that this figure is for illustration purposes only. The error function is minimized at a particular value of the neuron output as can be seen in the figure. The process of training a neural network is essentially the process of finding these minimizing output values for all the neurons in the network. Pruning this particular neuron (which translates to getting a zero output from it will result in a change in the total overall error. This change in error is represented by distance between the original minimum error (shown by the dashed line) and the top red arrow. This neuron is clearly a bad candidate for removal since removing it will result in a huge error increase.

The straight red line in the figure represents the first-order approximation of the error using Taylor Series as described before while the parabola represents a second-order approximation. It can be clearly seen that the second-order approximation is a much better estimate of the change in error.

One thing to note here is that it is possible in some cases that there is some thresholding required when trying to approximate the error using the 2nd order Taylor Series expansion. These cases might arise when the parabolic approximation undergoes a steep slope change. To take into account such cases, mean and median thresholding were employed, where any change above a certain threshold was assigned a mean or median value respectively.

Two pruning algorithms are proposed here. They are different in the way the neurons are ranked but both of them use ΔE_k, the approximation of the change in error as the basis for the ranking. ΔE_k can be calculated using the Brute Force method, or one of the two Taylor Series approximations discussed previously.

The first step in both the algorithms is to decide a stopping criterion. This can vary depending on the application but some intuitive stopping criteria can be: maximum number of neurons to remove, percentage scaling needed, maximum allowable accuracy drop etc.

3.5.1 Algorithm I: Single Overall Ranking

The complete algorithm is shown in Algorithm 1. The idea here is to generate a single ranked list based on the values of ΔE_k. This involves a single pass of second-order back-propagation (without weight updates) to collect the gradients for each neuron. The neurons from this rank-list (with the lowest values of ΔE_k) are then pruned according to the stopping criterion decided.

```
Data: optimally trained network, training set
Result: A pruned network
initialize and define stopping criterion ;
perform forward propagation over the training set ;
perform second-order back-propagation without updating weights and collect linear and quadratic
gradients ;
rank the remaining neurons based on $\Delta E_k^2$;
while stopping criterion is not met do
    remove the last ranked neuron ;
end
```

Algorithm 1: Single Overall Ranking

3.5.2 Algorithm II: Iterative Re-Ranking

In this greedy variation of the algorithm (Algorithm 2), after each neuron removal, the remaining network undergoes a single forward and backward pass of second-order back-propagation (without weight updates) and the rank list is formed again. Hence, each removal involves a new pass through the network. This method is computationally more expensive but takes into account the dependencies the neurons might have on one another which would lead to a change in error contribution every time a dependent neuron is removed.

```
Data: optimally trained network, training set
Result: A pruned network
initialize and define stopping criterion ;
while stopping criterion is not met do
    perform forward propagation over the training set ;
    perform second-order back-propagation without updating weights and collect linear and
    quadratic gradients ;
    rank the remaining neurons based on $\Delta E_k$ ;
    remove the worst neuron based on the ranking ;
end
```

Algorithm 2: Iterative Re-Ranking

4 Experimental Results

4.1 MNIST Dataset Results

4.1.1 Pruning a 1-Layer Network: Single-Pass Ranking

We first present the results for a single-layer neural network in Figure 3, 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.)

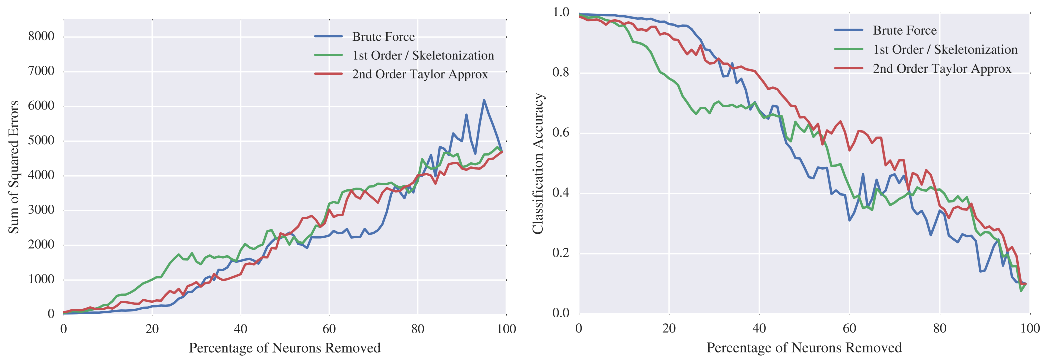

Figure 3: Degradation in squared error (left) and classification accuracy (right) after pruning a single-layer network trained on MNIST using a single-pass overall ranking procedure (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

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 (1989a)'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.

4.1.2 Pruning a 1-Layer Network: Re-Ranking After Each Removal

In Figure 4 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

4.1.3 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 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.

4.1.4 Pruning a 2-Layer Network: Single-Pass Ranking

Figure 8 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 3. (We again note that this algorithm is intentionally naive and is used for comparison only. Its performance should be expected to be poor.)

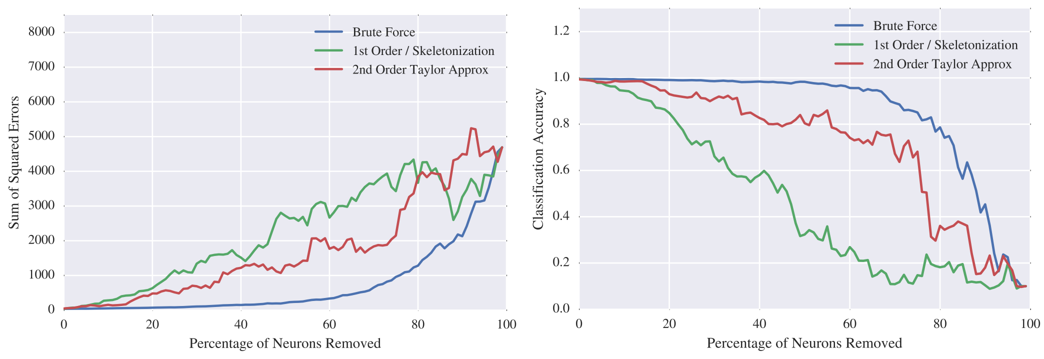

Figure 4: Degradation in squared error (left) and classification accuracy (right) after pruning a single-layer network trained on MNIST using an iterative re-ranking procedure (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

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.

4.1.5 Pruning a 2-Layer Network: Re-Ranking After Each Removal

Figure 9 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 4. 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

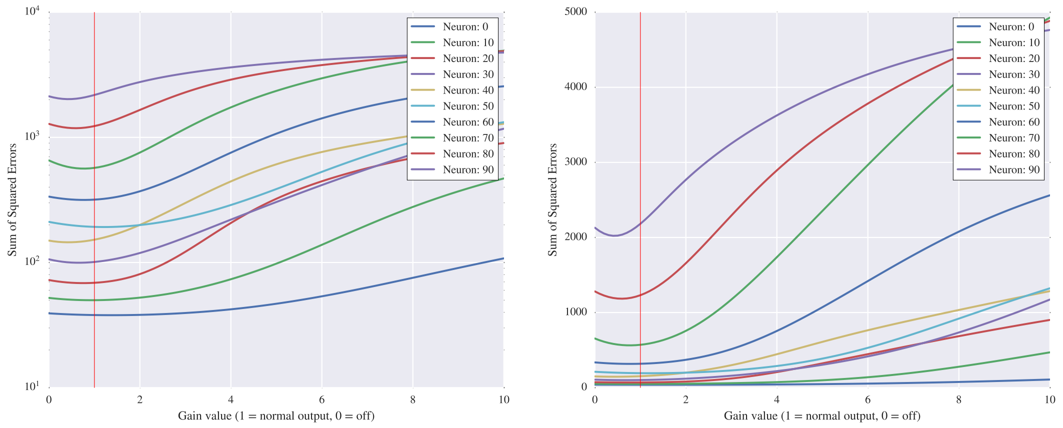

Figure 5: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

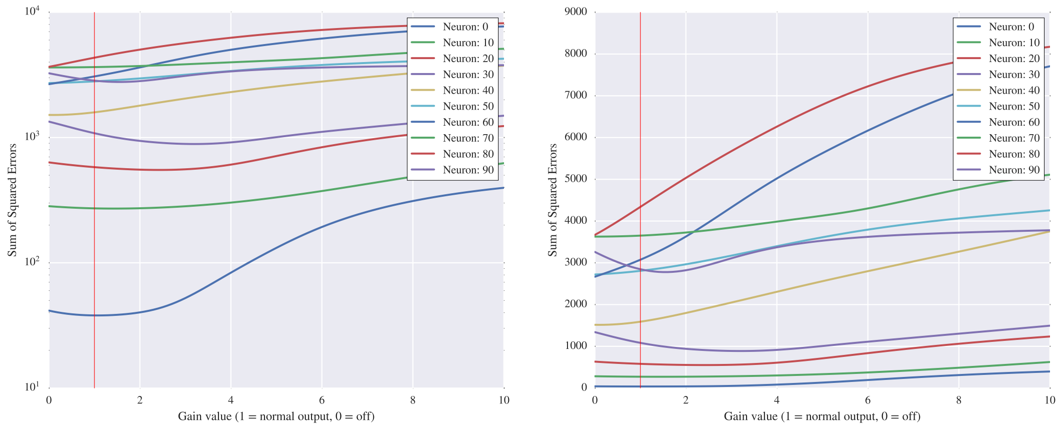

Figure 6: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

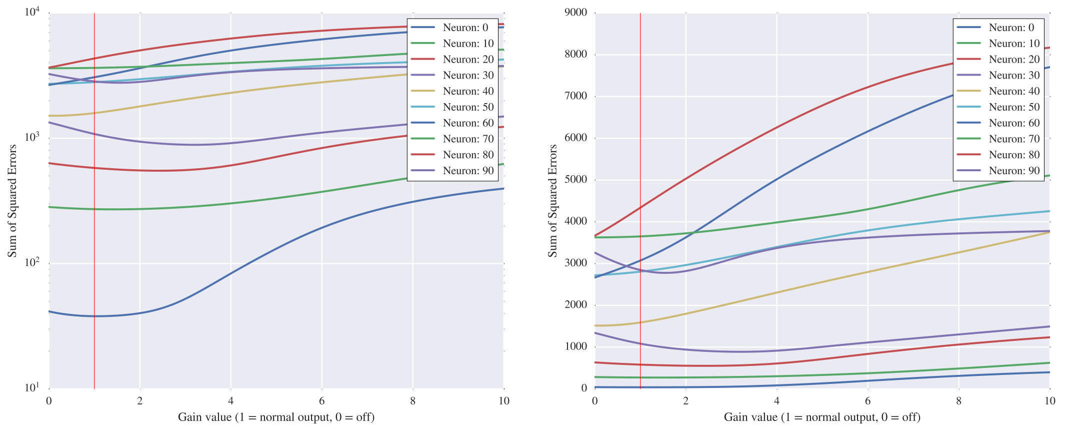

Figure 7: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 1 layer, 100 neurons, 10 outputs, logistic sigmoid activation, starting test accuracy: 0.998)

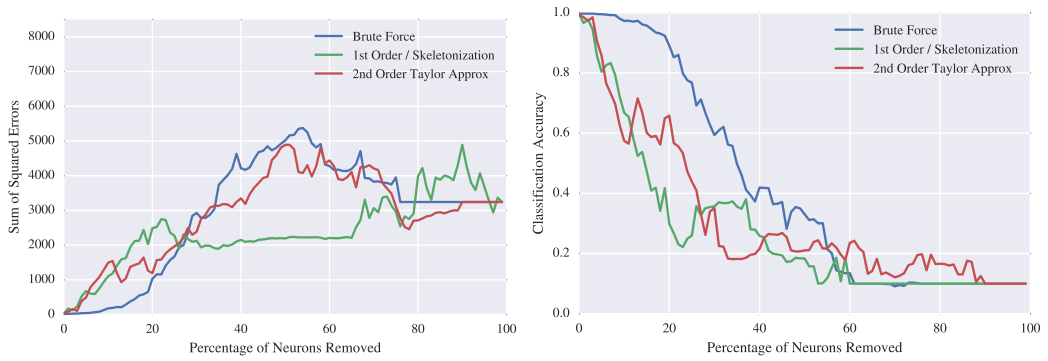

Figure 8: Degradation in squared error (left) and classification accuracy (right) after pruning a 2-layer network trained on MNIST using a single-pass overall ranking procedure (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

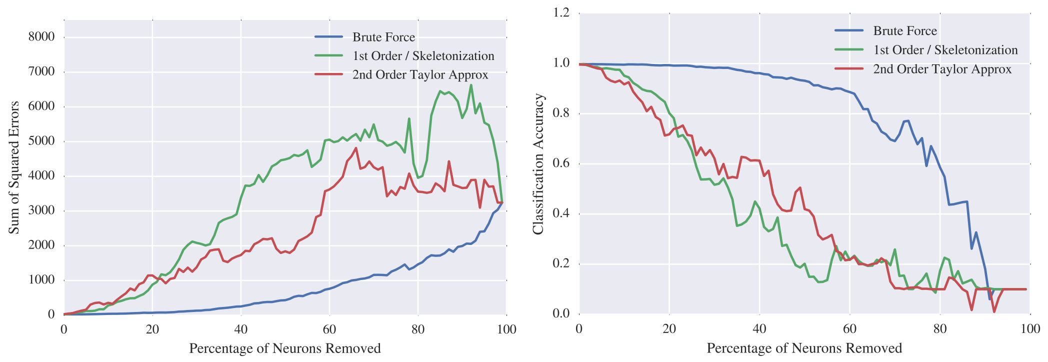

Figure 9: Degradation in squared error (left) and classification accuracy (right) after pruning a 2-layer network trained on MNIST using an iterative re-ranking procedure (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

4.1.6 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

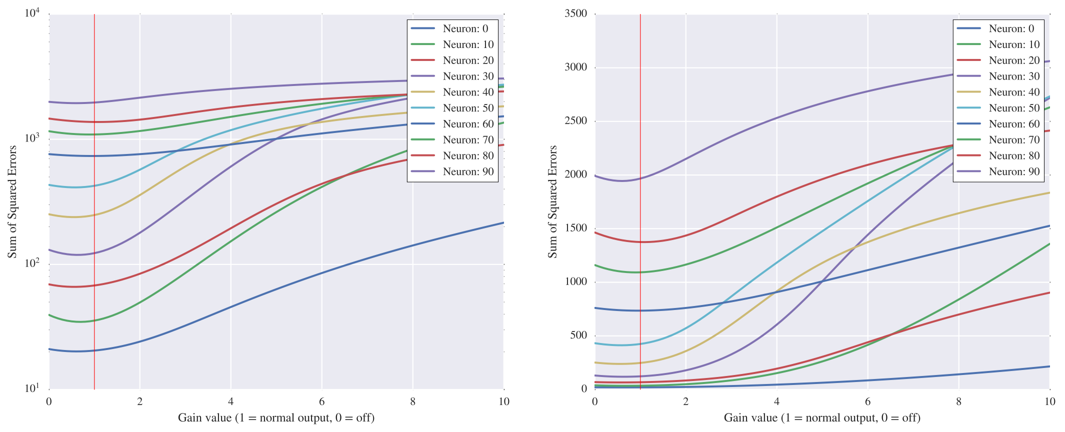

Figure 10: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

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 ver-

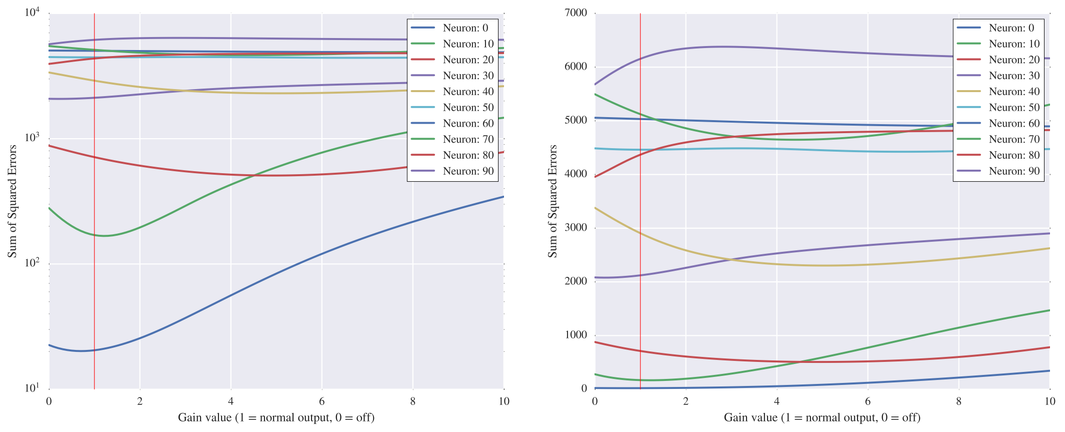

Figure 11: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

tical 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)

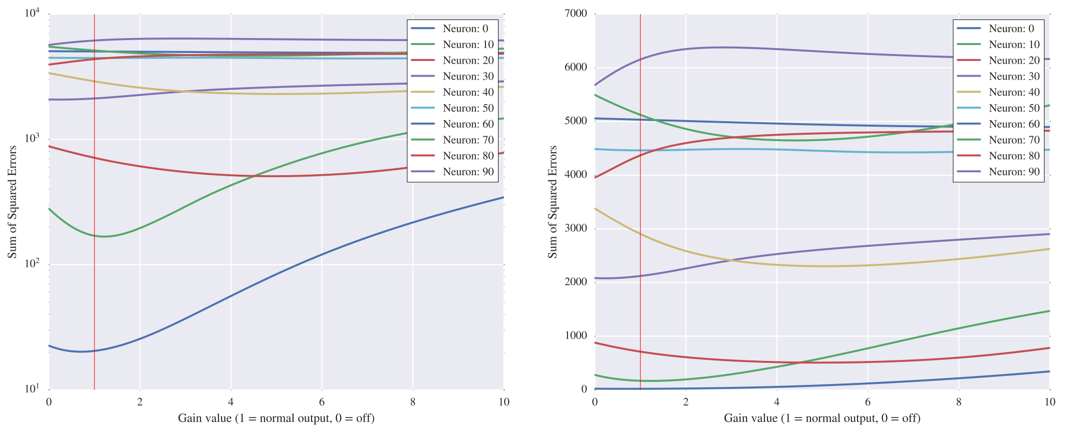

Figure 12: Error surface of the network output in log space (left) and in real space (right) with respect to each candidate neuron chosen for removal; (**Network:** 2 layers, 50 neurons/layer, 10 outputs, logistic sigmoid activation, starting test accuracy: 1.000)

5 Discussion of Results

* first or second layer? * cascade correlation in reverse? * retraining? * investigation of the true independence of the elements

6 Conclusions & Future Work

6.1 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

6.2 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: smoooooooth 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.

Acknowledgments

* thank: lukas drude abelino jiminez thomas schaaf don wolfe

REFERENCES

Balzer, Wolfgang, Takahashi, Masanobu, Ohta, Jun, and Kyuma, Kazuo. Weight quantization in boltzmann machines. *Neural Networks*, 4(3):405–409, 1991.

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.

Dundar, Gunhan and Rose, Kenneth. The effects of quantization on multilayer neural networks. *IEEE transactions on neural networks/a publication of the IEEE Neural Networks Council*, 6(6): 1446–1451, 1994.

Fahlman, Scott E and Lebiere, Christian. The cascade-correlation learning architecture. 1989.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Hassibi, Babak and Stork, David G. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

Hoehfeld, Markus and Fahlman, Scott E. Learning with limited numerical precision using the cascade-correlation algorithm. *IEEE Transactions on Neural Networks*, 3(4):602–611, 1992.

LeCun, Yann, Denker, John S, Solla, Sara A, Howard, Richard E, and Jackel, Lawrence D. Optimal brain damage. In *NIPs*, volume 89, 1989.

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, 1989a.

Mozer, Michael C and Smolensky, Paul. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989b.

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

Segee, Bruce E and Carter, Michael J. Fault tolerance of pruned multilayer networks. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pp. 447–452. IEEE, 1991.

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.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.