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

Nikolas Wolfe, Aditya Sharma, Lukas Drude & Bhiksha Raj
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{nwolfe, adityasharma, bhiksha}@cmu.edu
{drude@nt.upb.de}

ABSTRACT

We propose and evaluate a method for pruning neural networks to operate in constrained memory environments such as mobile or embedded devices. We 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 type of approximation by comparing against the ground truth value for 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 **?** as a mechanism to determine the size of 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, **?** 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 **?** writes, this approach can lead to overfitting as the network starts to latch on to idiosyncrasies in the training data.

Pruning algorithms, as comprehensively surveyed by **?**, 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.

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 (**?**), maxout (**?**), and cascade correlation (**?**), among others. However, these algorithms do not prioritize the reduction of network memory footprint as an explicit part of their optimization criteria per se, (although cascade correlation holds promise in this regard.) Computer memory size and processing capabilities have improved so much since the introduction of pruning algorithms that space complexity has become an almost 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 centers

remotely. It is of course conceivable, however, that performance-critical applications running on low-resource devices could benefit immensely from the ability to use powerful neural networks locally.

Unfortunately, at present there are few if any mechanisms to shrink neural networks down in order to meet an externally imposed constraint on byte-size in memory. Arguably one could accomplish this using any number of off-the-shelf pruning algorithms, such as Skeletonization (**?**), Optimal Brain Damage (**?**), or later variants such as Optimal Brain Surgeon (**?**). 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 is 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.

2 Second Derivative Gradient Terms

Given a neuron n with output a_i, and outgoing weights $[w_{i,1}, w_{i,2}, \ldots w_{i,j}]$, the input x to each of its j forward-connected neurons is given by:

$$x_j = \sum_i (w_{i,j} \cdot a_i) \tag{1}$$

For simplicity's sake, we will drop the index variable i and examine only the connection between the output a of neuron n and each of its j connections to the next forward layer. So the contribution c_j from neuron n to the input of each forward-connected neuron is given by:

$$c_j = w_j \cdot a \tag{2}$$

Where w_j is the weight connecting the output a from n to the input of the jth neuron. We will denote the error function of an optimally trained network as E. The second-derivative of E with respect to the output of neuron n is given by:

$$\frac{d^2E}{da^2} = \frac{d}{da}\frac{dE}{da} = \frac{d}{da}\sum_j \left(\frac{dE}{dc_j} \cdot \frac{dc_j}{da}\right) = \sum_j \left(\frac{d}{da}\frac{dE}{dc_j} \cdot \frac{dc_j}{da}\right) \tag{3}$$

Which states that the 2nd derivative of E with respect to a is the sum of the 2nd derivative terms of all outgoing connections.

3 APPENDIX A: SECOND DERIVATIVE BACK-PROPAGATION

Name and network definitions:

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

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 $m+1$ layer to the input of the ith neuron in the mth layer.

3.1 FIRST AND SECOND DERIVATIVES

The first and second derivatives of the cost function with respect to the outputs:

$$\frac{\partial E}{\partial o_i^{(0)}} = o_i^{(0)} - t_i \tag{5}$$

$$\frac{\partial^2 E}{\partial {o_i^{(0)}}^2} = 1 \tag{6}$$

The first and second derivatives of the sigmoid function in forms depending only on the output:

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right) \tag{7}$$
$$\sigma''(x) = \sigma'(x)\left(1 - 2\sigma(x)\right) \tag{8}$$

The second derivative of the sigmoid is easily derived from the first derivative:

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right) \tag{9}$$
$$\sigma''(x) = \frac{\mathrm{d}}{\mathrm{d}x} \underbrace{\sigma(x)}_{f(x)} \underbrace{\left(1 - \sigma(x)\right)}_{g(x)} \tag{10}$$
$$\sigma''(x) = f'(x)g(x) + f(x)g'(x) \tag{11}$$
$$\sigma''(x) = \sigma'(x)(1 - \sigma(x)) - \sigma(x)\sigma'(x) \tag{12}$$
$$\sigma''(x) = \sigma'(x) - 2\sigma(x)\sigma'(x) \tag{13}$$
$$\sigma''(x) = \sigma'(x)(1 - 2\sigma(x)) \tag{14}$$

And for future convenience:

$$\frac{\mathrm{d}o_i^{(m)}}{\mathrm{d}x_i^{(m)}} = \frac{\mathrm{d}}{\mathrm{d}x_i^{(m)}}\left(o_i^{(m)} = \sigma(x_i^{(m)})\right) \tag{15}$$
$$= \left(o_i^{(m)}\right)\left(1 - o_i^{(m)}\right) \tag{16}$$
$$= \sigma'\left(x_i^{(m)}\right) \tag{17}$$
$$\frac{\mathrm{d}^2 o_i^{(m)}}{\mathrm{d}{x_i^{(m)}}^2} = \frac{\mathrm{d}}{\mathrm{d}x_i^{(m)}}\left(\frac{\mathrm{d}o_i^{(m)}}{\mathrm{d}x_i^{(m)}} = \left(o_i^{(m)}\right)\left(1 - o_i^{(m)}\right)\right) \tag{18}$$
$$= \left(o_i^{(m)}\left(1 - o_i^{(m)}\right)\right)\left(1 - 2o_i^{(m)}\right) \tag{19}$$
$$= \sigma''\left(x_i^{(m)}\right) \tag{20}$$

Derivative of the error with respect to the ith neuron's input $x_i^{(0)}$ in the output layer:

$$\frac{\partial E}{\partial x_i^{(0)}} = \frac{\partial E}{\partial o_i^{(0)}} \frac{\partial o_i^{(0)}}{\partial x_i^{(0)}} \tag{21}$$

$$= \underbrace{\left(o_i^{(0)} - t_i\right)}_{\text{from (5)}} \underbrace{\sigma\left(x_i^{(0)}\right)\left(1 - \sigma\left(x_i^{(0)}\right)\right)}_{\text{from (7)}} \tag{22}$$

$$= \left(o_i^{(0)} - t_i\right)\left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right) \tag{23}$$

$$= \left(o_i^{(0)} - t_i\right)\sigma'\left(x_i^{(0)}\right) \tag{24}$$

Second derivative of the error with respect to the ith neuron's input $x_i^{(0)}$ in the output layer:

$$\frac{\partial^2 E}{\partial {x_i^{(0)}}^2} = \frac{\partial}{\partial x_i^{(0)}}\left(\frac{\partial E}{\partial o_i^{(0)}} \frac{\partial o_i^{(0)}}{\partial x_i^{(0)}}\right) \tag{25}$$

$$= \frac{\partial^2 E}{\partial x_i^{(0)} \partial o_i^{(0)}} \frac{\partial o_i^{(0)}}{\partial x_i^{(0)}} + \frac{\partial E}{\partial o_i^{(0)}} \frac{\partial^2 o_i^{(0)}}{\partial {x_i^{(0)}}^2} \tag{26}$$

$$= \frac{\partial^2 E}{\partial x_i^{(0)} \partial o_i^{(0)}} \underbrace{\left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right)}_{\text{from (7)}} + \underbrace{\left(o_i^{(0)} - t_i\right)}_{\text{from (5)}} \underbrace{\left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right)\left(1 - 2o_i^{(0)}\right)}_{\text{from (8)}} \tag{27}$$

$$\left(\frac{\partial^2 E}{\partial x_i^{(0)} \partial o_i^{(0)}}\right) = \frac{\partial}{\partial x_i^{(0)}} \frac{\partial E}{\partial o_i^{(0)}} = \frac{\partial}{\partial x_i^{(0)}} \underbrace{\left(o_i^{(0)} - t_i\right)}_{\text{from (5)}} = \frac{\partial o_i^{(0)}}{\partial x_i^{(0)}} = \underbrace{\left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right)}_{\text{from (7)}} \tag{28}$$

$$\frac{\partial^2 E}{\partial {x_i^{(0)}}^2} = \left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right)^2 + \left(o_i^{(0)} - t_i\right)\left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right)\left(1 - 2o_i^{(0)}\right) \tag{29}$$

$$= \left(\sigma'\left(x_i^{(0)}\right)\right)^2 + \left(o_i^{(0)} - t_i\right)\sigma''\left(x_i^{(0)}\right) \tag{30}$$

First derivative of the error with respect to a single input contribution $c_{ji}^{(0)}$ from neuron j to neuron i with weight $w_{ji}^{(0)}$ in the output layer:

$$\frac{\partial E}{\partial c_{ji}^{(0)}} = \frac{\partial E}{\partial o_i^{(0)}} \frac{\partial o_i^{(0)}}{\partial x_i^{(0)}} \frac{\partial x_i^{(0)}}{\partial c_{ji}^{(0)}} \tag{31}$$

$$= \underbrace{\left(o_i^{(0)} - t_i\right)}_{\text{from (5)}} \underbrace{\left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right)}_{\text{from (7)}} \frac{\partial x_i^{(0)}}{\partial c_{ji}^{(0)}} \tag{32}$$

$$\left(\frac{\partial x_i^{(m)}}{\partial c_{ji}^{(m)}}\right) = \frac{\partial}{\partial c_{ji}^{(m)}}\left(x_i^{(m)} = \sum_j w_{ji}^{(m)} o_j^{(m+1)}\right) = \frac{\partial}{\partial c_{ji}^{(m)}}\left(c_{ji}^{(m)} + k\right) = 1 \tag{33}$$

$$\frac{\partial E}{\partial c_{ji}^{(0)}} = \left(o_i^{(0)} - t_i\right)\left(o_i^{(0)}\left(1 - o_i^{(0)}\right)\right) \tag{34}$$

$$= \underbrace{\left(o_i^{(0)} - t_i\right)\sigma'\left(x_i^{(0)}\right)}_{\text{from (24)}} \tag{35}$$

$$\frac{\partial E}{\partial c_{ji}^{(0)}} = \frac{\partial E}{\partial x_i^{(0)}} \tag{36}$$

Second derivative of the error with respect to a single input contribution $c_{ji}^{(0)}$:

$$\frac{\partial^2 E}{\partial {c_{ji}^{(0)}}^2} = \frac{\partial}{\partial c_{ji}^{(0)}} \left(\underbrace{\frac{\partial E}{\partial c_{ji}^{(0)}} = \left(o_i^{(0)} - t_i\right) \sigma'\left(x_i^{(0)}\right)}_{\text{from (35)}} \right) \tag{37}$$

$$= \frac{\partial}{\partial c_{ji}^{(0)}} \left(\sigma\left(x_i^{(0)}\right) - t_i\right) \sigma'\left(x_i^{(0)}\right) \tag{38}$$

$$= \frac{\partial}{\partial c_{ji}^{(0)}} \left(\sigma\left(\sum_j w_{ji}^{(m)} o_j^{(m+1)}\right) - t_i\right) \sigma'\left(\sum_j w_{ji}^{(m)} o_j^{(m+1)}\right) \tag{39}$$

$$= \frac{\partial}{\partial c_{ji}^{(0)}} \left(\sigma\left(\sum_j c_{ji}^{(0)}\right) - t_i\right) \sigma'\left(\sum_j c_{ji}^{(0)}\right) \tag{40}$$

$$= \frac{\partial}{\partial c_{ji}^{(0)}} \underbrace{\left(\sigma\left(c_{ji}^{(0)} + k\right) - t_i\right)}_{f\left(c_{ji}^{(0)}\right)} \underbrace{\sigma'\left(c_{ji}^{(0)} + k\right)}_{g\left(c_{ji}^{(0)}\right)} \tag{41}$$

We now make use of the abbreviations f and g:

$$= f'\left(c_{ji}^{(0)}\right) g\left(c_{ji}^{(0)}\right) + f\left(c_{ji}^{(0)}\right) g'\left(c_{ji}^{(0)}\right) \tag{42}$$

$$= \sigma'\left(c_{ji}^{(0)} + k\right) \sigma'\left(c_{ji}^{(0)} + k\right) + \left(\sigma\left(c_{ji}^{(0)} + k\right) - t_i\right) \sigma''\left(c_{ji}^{(0)} + k\right) \tag{43}$$

$$= \sigma'\left(c_{ji}^{(0)} + k\right)^2 + \left(o_i^{(0)} - t_i\right) \sigma''\left(c_{ji}^{(0)} + k\right) \tag{44}$$

$$\left(c_{ji}^{(0)} + k = \sum_j c_{ji}^{(0)} = \sum_j w_{ji}^{(m)} o_j^{(m+1)} = x_i^{(0)}\right) \tag{45}$$

$$\frac{\partial^2 E}{\partial {c_{ji}^{(0)}}^2} = \underbrace{\left(\sigma'\left(x_i^{(0)}\right)\right)^2 + \left(o_i^{(0)} - t_i\right) \sigma''\left(x_i^{(0)}\right)}_{\text{from (30)}} \tag{46}$$

$$\frac{\partial^2 E}{\partial {c_{ji}^{(0)}}^2} = \frac{\partial^2 E}{\partial {x_i^{(0)}}^2} \tag{47}$$

3.1.1 Summary Of Output Layer Derivatives

$$\frac{\partial E}{\partial o_i^{(0)}} = o_i^{(0)} - t_i \qquad \frac{\partial^2 E}{\partial {o_i^{(0)}}^2} = 1 \tag{48}$$

$$\frac{\partial E}{\partial x_i^{(0)}} = \left(o_i^{(0)} - t_i\right) \sigma'\left(x_i^{(0)}\right) \qquad \frac{\partial^2 E}{\partial {x_i^{(0)}}^2} = \left(\sigma'\left(x_i^{(0)}\right)\right)^2 + \left(o_i^{(0)} - t_i\right) \sigma''\left(x_i^{(0)}\right) \tag{49}$$

$$\frac{\partial E}{\partial c_{ji}^{(0)}} = \frac{\partial E}{\partial x_i^{(0)}} \qquad \frac{\partial^2 E}{\partial {c_{ji}^{(0)}}^2} = \frac{\partial^2 E}{\partial {x_i^{(0)}}^2} \tag{50}$$

3.1.2 HIDDEN LAYER DERIVATIVES

The first derivative of the error with respect to a neuron with output $o_j^{(1)}$ in the first hidden layer, summing over all partial derivative contributions from the output layer:

$$\frac{\partial E}{\partial o_j^{(1)}} = \sum_i \frac{\partial E}{\partial o_i^{(0)}} \frac{\partial o_i^{(0)}}{\partial x_i^{(0)}} \frac{\partial x_i^{(0)}}{\partial c_{ji}^{(0)}} \frac{\partial c_{ji}^{(0)}}{\partial o_j^{(1)}} = \sum_i \underbrace{\left(o_i^{(0)} - t_i\right) \sigma'\left(x_i^{(0)}\right)}_{\text{from (24)}} w_{ji}^{(0)} \tag{51}$$

$$\frac{\partial c_{ji}^{(m)}}{\partial o_j^{(m+1)}} = \frac{\partial}{\partial o_j^{(m+1)}} \left(c_{ji}^{(m)} = w_{ji}^{(m)} o_j^{(m+1)}\right) = w_{ji}^{(m)} \tag{52}$$

$$\frac{\partial E}{\partial o_j^{(1)}} = \sum_i \frac{\partial E}{\partial x_i^{(0)}} w_{ji}^{(0)} \tag{53}$$

Note that this equation does not depend on the specific form of $\frac{\partial E}{\partial x_i^{(0)}}$, whether it involves a sigmoid or any other activation function. We can therefore replace the specific indexes with general ones, and use this equation in the future.

$$\frac{\partial E}{\partial o_j^{(m+1)}} = \sum_i \frac{\partial E}{\partial x_i^{(m)}} w_{ji}^{(m)} \tag{54}$$

The second derivative of the error with respect to a neuron with output $o_j^{(1)}$ in the first hidden layer:

$$\frac{\partial^2 E}{\partial {o_j^{(1)}}^2} = \frac{\partial}{\partial o_j^{(1)}} \frac{\partial E}{\partial o_j^{(1)}} \tag{55}$$

$$= \frac{\partial}{\partial o_j^{(1)}} \sum_i \frac{\partial E}{\partial x_i^{(0)}} w_{ji}^{(0)} \tag{56}$$

$$= \frac{\partial}{\partial o_j^{(1)}} \sum_i \left(o_i^{(0)} - t_i\right) \sigma'\left(x_i^{(0)}\right) w_{ji}^{(0)} \tag{57}$$

If we now make use of the fact, that $o_i^{(0)} = \sigma\left(x_i^{(0)}\right) = \sigma\left(\sum_j \left(w_{ji}^{(0)} o_j^{(1)}\right)\right)$, we can evaluate the expression further.

$$\frac{\partial^2 E}{\partial {o_j^{(1)}}^2} = \frac{\partial}{\partial o_j^{(1)}} \sum_i \underbrace{\left(\sigma\left(\sum_j w_{ji}^{(0)} o_j^{(1)}\right) - t_i\right)}_{f\left(o_j^{(1)}\right)} \underbrace{\sigma'\left(\sum_j w_{ji}^{(0)} o_j^{(1)}\right)}_{g\left(o_j^{(1)}\right)} w_{ji}^{(0)} \tag{58}$$

$$= \sum_i \left(f'\left(o_j^{(1)}\right) g\left(o_j^{(1)}\right) + f\left(o_j^{(1)}\right) g'\left(o_j^{(1)}\right)\right) \tag{59}$$

$$= \sum_i \sigma'\left(\sum_j w_{ji}^{(0)} o_j^{(1)}\right) w_{ji}^{(0)}\ \sigma'\left(\sum_j w_{ji}^{(0)} o_j^{(1)}\right) w_{ji}^{(0)} + \ldots \tag{60}$$

$$\sum_i \left(\sigma\left(\sum_j w_{ji}^{(0)} o_j^{(1)}\right) - t_i\right) \sigma''\left(\sum_j w_{ji}^{(0)} o_j^{(1)}\right) \left(w_{ji}^{(0)}\right)^2 \tag{61}$$

$$= \sum_i \left(\left(\sigma'\left(x_i^{(0)}\right)\right)^2 \left(w_{ji}^{(0)}\right)^2 + \left(o_i^{(0)} - t_i\right) \sigma''\left(x_i^{(0)}\right) \left(w_{ji}^{(0)}\right)^2\right) \tag{62}$$

$$= \sum_i \underbrace{\left(\left(\sigma'\left(x_i^{(0)}\right)\right)^2 + \left(o_i^{(0)} - t_i\right) \sigma''\left(x_i^{(0)}\right)\right)}_{\text{from (30)}} \left(w_{ji}^{(0)}\right)^2 \tag{63}$$

Summing up, we obtain the more general expression:

$$\frac{\partial^2 E}{\partial {o_j^{(1)}}^2} = \sum_i \frac{\partial^2 E}{\partial {x_i^{(0)}}^2} \left(w_{ji}^{(0)}\right)^2 \tag{64}$$

Note that the equation in (64) does not depend on the form of $\frac{\partial^2 E}{\partial {x_x^{(0)}}^2}$, which means we can replace the specific indexes with general ones:

$$\frac{\partial^2 E}{\partial {o_j^{(m+1)}}^2} = \sum_i \frac{\partial^2 E}{\partial {x_i^{(m)}}^2} \left(w_{ji}^{(m)}\right)^2 \tag{65}$$

At this point we are beginning to see the recursion in the form of the 2nd derivative terms which can be thought of analogously to the first derivative recursion which is central to the back-propagation algorithm. The formulation above which makes specific reference to layer indexes also works in the general case.

Consider the ith neuron in any layer m with output $o_i^{(m)}$ and input $x_i^{(m)}$. The first and second derivatives of the error E with respect to this neuron's *input* are:

$$\frac{\partial E}{\partial x_i^{(m)}} = \frac{\partial E}{\partial o_i^{(m)}} \frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} \tag{66}$$

$$\frac{\partial^2 E}{\partial {x_i^{(m)}}^2} = \frac{\partial}{\partial x_i^{(m)}} \frac{\partial E}{\partial x_i^{(m)}} \tag{67}$$

$$= \frac{\partial}{\partial x_i^{(m)}} \left(\frac{\partial E}{\partial o_i^{(m)}} \frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} \right) \tag{68}$$

$$= \frac{\partial^2 E}{\partial x_i^{(m)} \partial o_i^{(m)}} \frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} + \frac{\partial E}{\partial o_i^{(m)}} \frac{\partial^2 o_i^{(m)}}{\partial {x_i^{(m)}}^2} \tag{69}$$

$$= \frac{\partial}{\partial o_i^{(m)}} \left(\frac{\partial E}{\partial x_i^{(m)}} = \frac{\partial E}{\partial o_i^{(m)}} \frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} \right) \frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} + \frac{\partial E}{\partial o_i^{(m)}} \sigma'' \left(x_i^{(m)}\right) \tag{70}$$

$$= \frac{\partial^2 E}{\partial {o_i^{(m)}}^2} \left(\frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} \frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} \right) + \frac{\partial E}{\partial o_i^{(m)}} \sigma'' \left(x_i^{(m)}\right) \tag{71}$$

$$\frac{\partial^2 E}{\partial {x_i^{(m)}}^2} = \frac{\partial^2 E}{\partial {o_i^{(m)}}^2} \left(\sigma' \left(x_i^{(m)}\right)\right)^2 + \frac{\partial E}{\partial o_i^{(m)}} \sigma'' \left(x_i^{(m)}\right) \tag{72}$$

Note the form of this equation is the general form of what was derived for the output layer in (30). Both of the above first and second terms are easily computable and can be stored as we propagate back from the output of the network to the input. With respect to the output layer, the first and second derivative terms have already been derived above. In the case of the $m+1$ hidden layer during back propagation, there is a summation of terms calculated in the mth layer. For the first derivative, we have this from (54).

$$\frac{\partial E}{\partial o_j^{(m+1)}} = \sum_i \frac{\partial E}{\partial x_i^{(m)}} w_{ji}^{(m)} \tag{73}$$

And the second derivative for the jth neuron in the $m+1$ layer:

$$\frac{\partial^2 E}{\partial {x_j^{(m+1)}}^2} = \frac{\partial^2 E}{\partial {o_j^{(m+1)}}^2} \left(\sigma' \left(x_j^{(m+1)}\right)\right)^2 + \frac{\partial E}{\partial o_j^{(m+1)}} \sigma'' \left(x_j^{(m+1)}\right) \tag{74}$$

We can replace both derivative terms with the forms which depend on the previous layer:

$$\frac{\partial^2 E}{\partial {x_j^{(m+1)}}^2} = \underbrace{\sum_i \frac{\partial^2 E}{\partial {x_i^{(0)}}^2} \left(w_{ji}^{(0)}\right)^2}_{\text{from } (65)} \left(\sigma' \left(x_j^{(m+1)}\right)\right)^2 + \underbrace{\sum_i \frac{\partial E}{\partial x_i^{(m)}} w_{ji}^{(m)}}_{\text{from } (54)} \sigma'' \left(x_j^{(m+1)}\right) \tag{75}$$

And this horrible mouthful of an equation gives you a general form for any neuron in the jth position of the $m+1$ layer. Taking very careful note of the indexes, this can be more or less translated painlessly to code. You are welcome, world.

3.1.3 SUMMARY OF HIDDEN LAYER DERIVATIVES

$$\frac{\partial E}{\partial o_j^{(m+1)}} = \sum_i \frac{\partial E}{\partial x_i^{(m)}} w_{ji}^{(m)} \qquad \frac{\partial^2 E}{\partial o_j^{(m+1)^2}} = \sum_i \frac{\partial^2 E}{\partial x_i^{(m)^2}} \left(w_{ji}^{(m)}\right)^2 \tag{76}$$

$$\frac{\partial E}{\partial x_i^{(m)}} = \frac{\partial E}{\partial o_i^{(m)}} \frac{\partial o_i^{(m)}}{\partial x_i^{(m)}} \tag{77}$$

$$\frac{\partial^2 E}{\partial x_j^{(m+1)^2}} = \frac{\partial^2 E}{\partial o_j^{(m+1)^2}} \left(\sigma'\left(x_j^{(m+1)}\right)\right)^2 + \frac{\partial E}{\partial o_j^{(m+1)}} \sigma''\left(x_j^{(m+1)}\right) \tag{78}$$

$$\frac{\partial E}{\partial c_{ji}^{(m)}} = ... \qquad \frac{\partial^2 E}{\partial c_{ji}^{(m)^2}} = ... \tag{79}$$

3.1.4 TAYLOR SERIES REPRESENTATION OF ERROR

Let us denote the total error from the optimally trained neural network for any given validation dataset with N instances as E_{total}. Then,

$$E_{\text{total}} = \sum_n E_n, \tag{80}$$

where E_n is the error from the network over one validation instance. E_n can be seen as a function O, where O is the output of any general neuron in the network (In reality this error depends on each neuron's output, but for the sake of simplicity we use O to represent that). This error can be approximated at a particular neuron's output (say O_k) by using the 2nd order Taylor Series as,

$$\hat{E}_n(O) \approx E_n(O_k) + (O - O_k) \cdot \left.\frac{\partial E_n}{\partial O}\right|_{O_k} + 0.5 \cdot (O - O_k)^2 \cdot \left.\frac{\partial^2 E_n}{\partial O^2}\right|_{O_k}, \tag{81}$$

where $\hat{E}_n(O_k)$ represents the contribution of a neuron k to the total error E_n of the network for any given validation instance n. When this neuron is pruned, its output O_k becomes 0. From equation 81, the contribution $E_n(0)$ of this neuron, then becomes:

$$\hat{E}_n(0) \approx E_n(O_k) - O_k \cdot \left.\frac{\partial E_n}{\partial O}\right|_{O_k} + 0.5 \cdot O_k^2 \cdot \left.\frac{\partial^2 E_n}{\partial O^2}\right|_{O_k} \tag{82}$$

Replacing O by O_k in equation 81 shows us that the error is approximated perfectly by equation 81 at O_k. Using this and equation 82 we get:

$$\Delta E_{n,k} = \hat{E}_n(0) - \hat{E}_n(O_k) = -O_k \cdot \left.\frac{\partial E_n}{\partial O}\right|_{O_k} + 0.5 \cdot O_k^2 \cdot \left.\frac{\partial^2 E_n}{\partial O^2}\right|_{O_k}, \tag{83}$$

where $\Delta E_{n,k}$ is the change in the total error of the network given a validation instance n, when exactly one neuron (k) is turned off.

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.

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.

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, 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.

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.