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

Nikolas Wolfe, Aditya Sharma, Lukas Drude & Bhiksha Raj
School of Computer Science
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 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 ranking criteria we discuss further in this section. This ranking is done using a validation dataset which is different from the dataset used for training the network. The effects of removing an increasing percentage of neurons based off their ranks are then analysed in the results section.

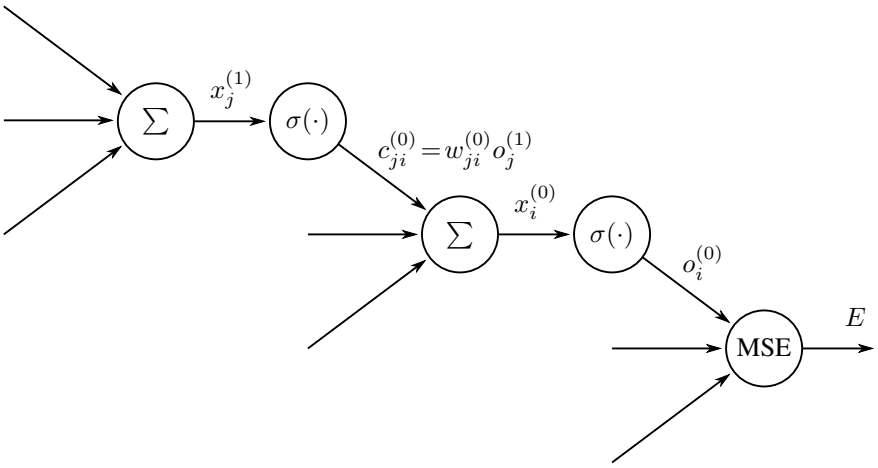
1.1 BRUTE FORCE REMOVAL

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.

1.2 LINEAR APPROXIMATION APPROACH (THE GAIN-SWITCH)

Figure ???????????? shows a neuron in a neural network

$$z = O \cdot g \frac{\mathrm{d}z}{\mathrm{d}O} = g \qquad \frac{\mathrm{d}E}{\mathrm{d}O} = \frac{\mathrm{d}E}{\mathrm{d}z} \cdot \frac{\mathrm{d}z}{\mathrm{d}O} = \frac{\mathrm{d}E}{\mathrm{d}z} \cdot g \frac{\mathrm{d}E}{\mathrm{d}g} = \frac{\mathrm{d}E}{\mathrm{d}z} \cdot \frac{\mathrm{d}z}{\mathrm{d}g} = \frac{\mathrm{d}E}{\mathrm{d}z} \cdot O \tag{1}$$



1.2.1 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,

$$E_{\text{total}} = \sum_n E_n, \tag{2}$$

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,

$$\hat{E}_n(O) \approx E_n(O_k) + (O - O_k) \cdot E_n(O_k) + \left.\frac{\partial E_n}{\partial O}\right|_{O_k} + 0.5 \cdot (O - O_k)^2 \cdot \left.\frac{\partial^2 E_n}{\partial O^2}\right|_{O_k}, \tag{3}$$

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 3, the contribution $E_n(0)$ of this neuron, then becomes:

$$\hat{E}_n(0) \approx E_n(O_k) - O_k \cdot \left.\frac{\partial E_n}{\partial O}\right|_{O_k} + 0.5 \cdot O_k^2 \cdot \left.\frac{\partial^2 E_n}{\partial O^2}\right|_{O_k} \tag{4}$$

Replacing O by O_k in equation 3 shows us that the error is approximated perfectly by equation 3 at O_k. Using this and equation 4 we get:

$$\Delta E_{n,k} = \hat{E}_n(0) - \hat{E}_n(O_k) = -O_k \cdot \left.\frac{\partial E_n}{\partial O}\right|_{O_k} + 0.5 \cdot O_k^2 \cdot \left.\frac{\partial^2 E_n}{\partial O^2}\right|_{O_k}, \tag{5}$$

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.

1.3 Quadratic Approximation Approach

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

$$E = \frac{1}{2}\sum_i (o_i^{(0)} - t_i)^2 \quad o_i^{(m)} = \sigma(x_i^{(m)}) \quad x_i^{(m)} = \sum_j w_{ji}^{(m)} o_j^{(m+1)} \quad c_{ji}^{(m)} = w_{ji}^{(m)} o_j^{(m+1)} \tag{6}$$

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. Note that we are dropping the E_n notation used previously as the subsequent discussion is insusceptible to the data instances. $o_i^{(m)}$ 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.

As seen in equation 5, $\Delta E_{n,k}$ which can now be represented as ΔE_k is the quadratic approximation of the change in error due to the kth neuron being turned off which becomes the basis of our ranking in this approach. The quadratic term in equation 5 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 5 in our new terminology here:

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

The second term here 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. We will quote some results from the derivation here. The second-order derivative term can be represented as:

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

Here,$c_{ji}^{(m)}$ is one of the component terms of $x_i^{(m)}$, as follows from the equations in 6. 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.

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