Neural nets and Back propagation derivation

AI – Gabor

Although the same ideas run through all variations of back propagation, the specifics do affect the details of the math involved. If you use a different architecture, you must understand the architectural differences and the math consequences (or be able to transform one to the other).

In the layout for the architecture that we use in this class, the output of the adders are the y_i (that is, the inputs to the hidden cells) and the output of those cells are the x_i. One could also say the y_i are scaled, summed versions of the x_i.

The final layer is only for scaling. That is, the number of outputs is equal to the number of cells in the final hidden layer, and the number of weights from the final hidden layer to the outputs is equal to the number of outputs. (Note: this is **non-standard**)

The layers between the input and the output layers are the hidden layers.

We'll start numbering the inputs as layer 0; the first hidden layer is layer 1. The weights, x's and y's immediately to the right of a given cell (or input) are construed to be in the same layer. In the below, however, the hidden cell's layer is never important as the layer that the x and y are in drive the calculations.

Example: What are the number of inputs, outputs, and cells in each layer if the number of weights in a weights file (on each line) are as follows:

```
10
6
6
6
3
```

Answer: There are three outputs (determined from the 3 in the final line), hence 3 in the final hidden layer. Working backwards from that point, each number gives the product of the number of cells in adjacent layers. Hence, in reverse order, 3 outputs, then hidden layers with counts 3, 2, 3, 2, and 5 inputs.

Given an architecture (ie. number of layers, and number of nodes per layer), **what describes a (neural) network?**

Answer: It's NOT the node values. Those change with each test case. Instead, it's **the weights** (on the connections).

The goal with neural networks is to **find the weights by minimizing the error** of the network.

OK, so what is the error?

Answer: Given some input vector $\overrightarrow{\boldsymbol{x^0}} =< x_0^0, x_1^0, x_2^0, \ldots x_{Cnt_0}^0 >$ with n hidden layers, there is some output vector $\overrightarrow{\boldsymbol{y^n}} =< y_0^n, y_1^n, y_2^n, \ldots y_{Cnt_n}^n >$. However, for that input vector, we actually want (were expecting) a fixed training vector: $\vec{\boldsymbol{t}} =< t_0, t_1, t_2, \ldots t_{Cnt_n} >$. Then the typical error is defined as:

$$E = \frac{1}{2}\sum_{k}^{Cnt_n}(t_k - y_k^n)^2 = \frac{1}{2}\left|\vec{\boldsymbol{t}} - \overrightarrow{\boldsymbol{y^n}}\right|^2$$

Note that superscript on the vector components indicates the layer and the subscripts are for node ordering. The squared symbols are not superscripts of 2.

So how do we minimize this?

Answer, by heading in the direction of the **negative of the gradient**.

How do we find it?

One should take partials with respect to all the independent variables.

What are the independent variables?

The $x's$ right?

ACK! NO!

The weights are the independent variables. Those are the parameters (that we can change) defining a neural network. The inputs are fixed values. The remaining $x's$ and $y's$ are dependent variables (dependent on the weights, given an input/expected output training set). The inputs to the network and expected outputs have been set by the person constructing the neural net and are non-negotiable. It's the weights that are changeable.

Thus, we'd like to **take partials with respect to each weight**.

The trick is to do it in an orderly fashion, so, we'll start with the easiest set of weights, namely the rightmost ones (ie. in the final layer). These are special since for each output of a cell (x_i^n) in

the final layer, there is only one weight (w_{ii}^n) connecting it to the output (y_i^n). Nevertheless, we'd still like to take partials with respect to those Cnt_n weights.

In other words, we'd like to find: $-\frac{\partial E}{\partial w_{ii}^n}$ for each w_{ii}^n in the final layer.

The key observation is that each of the weights is independent of each other. We may change one without having to change the other. That means that as far as any particular weight is concerned, if it tries to take a partial of some expression and it sees a different weight in the expression, it's just a constant as far as it's concerned. Here we go:

$$-\frac{\partial E}{\partial w_{ii}^n} = -\frac{\partial\left(\frac{1}{2}\sum_k^{Cnt_n}(t_k - y_k^n)^2\right)}{\partial w_{ii}^n}$$

Apply chain rule to the squared expression, noting that two negatives cancel

$$= \sum_k^{Cnt_n}(t_k - y_k^n)\frac{\partial y_k^n}{\partial w_{ii}^n}$$

Expand y_k^n in terms of the weights in the final layer:

$$= \sum_k^{Cnt_n}(t_k - y_k^n)\frac{\partial(x_k^n\, w_{kk}^n)}{\partial w_{ii}^n}$$

Observe that the x_k^n do not depend on w_{kk}^n so they are like constants as far as a w_{kk}^n is concerned:

$$= \sum_k^{Cnt_n}(t_k - y_k^n)\, x_k^n \frac{\partial(w_{kk}^n)}{\partial w_{ii}^n}$$

Each w_{ii}^n picks out exactly one element from the sum, so

$$\boxed{-\frac{\partial E}{\partial w_{ii}^n} = (t_i - y_i^n)x_i^n}$$

In further calculations, it will be understood that sums go over each node in the indicated layer, so only the index will be shown, and not the count, $Cnt.$ of the number of nodes in the layer. We'd now like to ratchet up what we just did to the next level and find the partials of $-E$ with respect to the weights in the penultimate layer, layer $n-1$. In other words, we'd like to find

$$-\frac{\partial E}{\partial w_{ij}^{n-1}} = -\frac{\partial\left(\frac{1}{2}\sum_k(t_k - y_k^n)^2\right)}{\partial w_{ij}^{n-1}}$$

Apply the chain rule to the squared expression, noting that two negatives cancel

$$= \sum_k (t_k - y_k^n) \frac{\partial y_k^n}{\partial w_{ij}^{n-1}}$$

Now, having gone through it once, we realize the only y_k^n that is dependent on w_{ij}^{n-1} is y_j^n, so

$$= (t_j - y_j^n) \frac{\partial y_j^n}{\partial w_{ij}^{n-1}} = (t_j - y_j^n) \frac{\partial (x_j^n \, w_{jj}^n)}{\partial w_{ij}^{n-1}} = (t_j - y_j^n) w_{jj}^n \frac{\partial x_j^n}{\partial w_{ij}^{n-1}}$$

At this point, in order to get x_j^n in terms of w_{ij}^{n-1} there's no help for it but to "push" this calculation back through the node at layer n. It looks like this:

$$= (t_j - y_j^n) w_{jj}^n \frac{\partial f(y_j^{n-1})}{\partial w_{ij}^{n-1}} = (t_j - y_j^n) w_{jj}^n f'(y_j^{n-1}) \frac{\partial y_j^{n-1}}{\partial w_{ij}^{n-1}}$$

And now we express y_j^{n-1} in terms of the $\overrightarrow{\boldsymbol{x^{n-1}}}$ and the weights connecting that vector to the particular cell that y_j^{n-1} feeds into:

$$= (t_j - y_j^n) w_{jj}^n f'(y_j^{n-1}) \frac{\partial \left(\overrightarrow{x^{n-1}} \bullet < w_{0j}^{n-1}, w_{1j}^{n-1}, \dots > \right)}{\partial w_{ij}^{n-1}}$$

But w_{ij}^{n-1} picks out exactly one of these, since none of the other ones, either the x^{n-1} or other weights in that layer depend on it so:

$$\boxed{-\frac{\partial E}{\partial w_{ij}^{n-1}} = (t_j - y_j^n) w_{jj}^n f'(y_j^{n-1}) x_i^{n-1}}$$

It's pretty neat that the summations disappear, but it would be an incorrect conclusion to think it keeps happening as you keep propagating backwards. The weights in the rightmost two layers are special cases since only one output node depends on each such weight. The more general situation starts to emerge with the next set of weights (those in layer $n-2$), so let's go through them. We'd like to find

$$-\frac{\partial E}{\partial w_{ij}^{n-2}} = -\frac{\partial \left(\frac{1}{2} \sum_k (t_k - y_k^n)^2 \right)}{\partial w_{ij}^{n-2}}$$

Apply the chain rule to the squared expression as before to get

$$= \sum_k (t_k - y_k^n) \frac{\partial y_k^n}{\partial w_{ij}^{n-2}} = \sum_k (t_k - y_k^n) \frac{\partial (x_k^n \, w_{kk}^n)}{\partial w_{ij}^{n-2}}$$

$$= \sum_k (t_k - y_k^n) w_{kk}^n \frac{\partial x_k^n}{\partial w_{ij}^{n-2}} = \sum_k (t_k - y_k^n) w_{kk}^n \frac{\partial f(y_k^{n-1})}{\partial w_{ij}^{n-2}}$$

However, since w_{ij}^{n-2} feeds into node j in layer $n-1$, and since that node (along with all other nodes in layer $n-1$) feed into all nodes into the final hidden layer, all of the x_k^n (hence y_k^{n-1}) are dependent on w_{ij}^{n-2}. In other words, we can't get rid of the summation at this point. But we do apply the chain rule again.

$$= \sum_k (t_k - y_k^n) w_{kk}^n f'(y_k^{n-1}) \frac{\partial y_k^{n-1}}{\partial w_{ij}^{n-2}}$$

$$= \sum_k (t_k - y_k^n) w_{kk}^n f'(y_k^{n-1}) \frac{\partial \left(\overrightarrow{x^{n-1}} \cdot < w_{0k}^{n-1}, w_{1k}^{n-1}, \dots > \right)}{\partial w_{ij}^{n-2}}$$

It is at this point that we can finally say that weight w_{ij}^{n-2} affects only x_j^{n-1} out of all the x^{n-1} since w_{ij}^{n-2} affects only y_j^{n-2} out of all the y^{n-2}. Since all the layer $n-1$ weights are independent of w_{ij}^{n-2} it means that ∂w_{ij}^{n-2} picks out only term $x_j^{n-1} w_{jk}^{n-1}$, so we have

$$= \sum_k (t_k - y_k^n) w_{kk}^n f'(y_k^{n-1}) \frac{\partial \left(x_j^{n-1} w_{jk}^{n-1} \right)}{\partial w_{ij}^{n-2}}$$

$$= \sum_k (t_k - y_k^n) w_{kk}^n f'(y_k^{n-1}) w_{jk}^{n-1} \frac{\partial x_j^{n-1}}{\partial w_{ij}^{n-2}}$$

$$= \sum_k (t_k - y_k^n) w_{kk}^n f'(y_k^{n-1}) w_{jk}^{n-1} f'\left(y_j^{n-2}\right) \frac{\partial y_j^{n-2}}{\partial w_{ij}^{n-2}}, \qquad \text{so}$$

$$\boxed{-\frac{\partial E}{\partial w_{ij}^{n-2}} = \sum_k (t_k - y_k^n) w_{kk}^n f'(y_k^{n-1}) w_{jk}^{n-1} f'\left(y_j^{n-2}\right) x_i^{n-2}}$$

At this point it may just be possible to start seeing a pattern emerge, although it might take another layer. But the care that must be taken in taking the partials is becoming somewhat problematic,

and if we got one sum with this layer, it would be reasonable to expect another sum when we push the calculations by an additional layer. The typical text bails out earlier than here with an "it can be shown", which they never do. We'd prefer a better approach.

Therefore, we reconsider the problem. Given a layer L, the error E can be defined completely in terms of the y^L and the weights in all layers forward of L. In other words, $E = E\left(y^L, w^{(L+1)+}\right)$. We would like to find

$$-\frac{\partial E}{\partial w_{ij}^L} = -\frac{\partial E}{\partial y_j^L}\frac{\partial y_j^L}{\partial w_{ij}^L} = -\frac{\partial E}{\partial y_j^L}\frac{\partial\left(x_i^L w_{ij}^L\right)}{\partial w_{ij}^L}$$

by the chain rule as only y_j^L out of all the variables contributing to E is dependent on w_{ij}^L, so

$$\boxed{-\frac{\partial E}{\partial w_{ij}^L} = -x_i^L\frac{\partial E}{\partial y_j^L}}$$

To determine $\frac{\partial E}{\partial y_j^L}$, we will consider the same E as before and determine, instead,

$$-\frac{\partial E}{\partial y_j^{L-1}} = -\sum_k \frac{\partial E}{\partial y_k^L}\frac{\partial\left(\overrightarrow{x^L}\cdot < w_{0k}^L, w_{1k}^L, \ldots >\right)}{\partial y_j^{L-1}}$$

as the weights do not depend on y_j^{L-1} but the y^L do. Too, only x_j^L out of x^L depends on y_j^{L-1}, so

$$= -\sum_k \frac{\partial E}{\partial y_k^L}\frac{\partial\left(x_j^L \cdot w_{jk}^L\right)}{\partial y_j^{L-1}} = -\sum_k \frac{\partial E}{\partial y_k^L} w_{jk}^L \frac{dx_j^L}{dy_j^{L-1}} = -\sum_k \frac{\partial E}{\partial y_k^L} w_{jk}^L f'\left(y_j^{L-1}\right), \qquad \text{or}$$

$$\boxed{-\frac{\partial E}{\partial y_j^{L-1}} = \sum_k -\frac{\partial E}{\partial y_k^L} w_{jk}^L f'\left(y_j^{L-1}\right)}$$

Finally, for outputs,

$$-\frac{\partial E}{\partial y_i^n} = -\frac{\partial\left(\frac{1}{2}\sum_k^{Cnt_n}(t_k - y_k^n)^2\right)}{\partial y_i^n} \qquad \text{or} \qquad \boxed{-\frac{\partial E}{\partial y_i^n} = (t_i - y_i^n)}$$

This says that starting with the rightmost layer n, $-\frac{\partial E}{\partial y_i^n}$ can easily be determined. Then, the negative partial of E with respect to the y_j^L in the prior layers can be recursively determined by dotting all the partials of ($-E$ with respect to) y^{L+1}'s that a given y feeds (when going forward) with the weights to those y's and multiplying this dot product by the derivative of that y's node.

Finally, the original item being sought, $\frac{\partial E}{\partial w_{ij}^L}$ is just the product of x_i^L with $-\frac{\partial E}{\partial y_j^L}$. In other words, there is now an orderly process to determine the negative of the gradient of E.

In practice, this gradient is multiplied by a small number alpha (typically .01 to .1) before updating all the weights.

What might help in making the gradient calculation is to clone the complete feed forward data structures once the feed forward computation is done (this means all the node values and all the weights). It is possible to avoid the cloning if one is careful, but we'll consider the cloned version, which means that we will have the outputs of all the internal nodes available for use. When you get down to it, you will see that you do not need the y values (except the final outputs), which is anyway fine, since the feed forward network is probably not preserving them.

The general strategy is to update the values in the boxes and circles from the back propagation architecture handout (going from right to left), and each time you update a layer of cells (in the clone), you'll be able to overwrite weight values (in the clone) that are to the immediate left of that layer of cells with the correct (negative of the) gradient values. Thus, the rightmost boxes get the $-\frac{\partial E}{\partial y_i^n}$ values while the circles (nodes) get the $-\frac{\partial E}{\partial y_j^L}$ values as you get to them. The partials with respect to the weights are then easy.

Specifically, for each rectangle (that is to say the output of the feed forward network) on the far right (in the clone), overwrite the value with the expected value less the actual value: $t_i - y_i^n$.

Now to compute any circle (in the clone) to the left of a computed layer (in the clone), do the following: take the dot product of the computed layer with the weights going from the circle being computed to the already computed layer and multiply that by the derivative at the circle. As you do this (or afterwards), you can compute the gradient as follows: For each weight, w_{ij}^k for which you would like to find $-\frac{\partial E}{\partial w_{ij}^k}$ simply find node j in layer $k+1$ in the clone (ie. the cell which that weight goes to in the clone), multiply it by the output of the cell (NOT in the clone) that it's coming from, and that gives you the negative partial with respect to that weight.

In the next 3 labs, the transfer function, $f(x)$ will always be the logistic function $f(x) = \frac{1}{1+e^{-x}}$. Therefore $f'(x) = f(x)\big(1 - f(x)\big)$. That is to say, when the prior paragraphs talk of derivatives, then mean a simple product where what you use for $f(x)$ is the output of the cell in question. In other words, $f'\big(y_j^k\big) = x_j^{k+1}\big(1 - x_j^{k+1}\big)$.