6.034f Neural Net Notes
October 28, 2010

These notes are a supplement to material presented in lecture. I lay out the mathematics more prettily and extend the analysis to handle multiple-neurons per layer. Also, I develop the back propagation rule, which is often needed on quizzes.

I use a notation that I think improves on previous explanations. The reason is that the notation here plainly associates each input, output, and weight with a readily identified neuron, a left-side one and a right-side one. When you arrive at the update formulas, you will have less trouble relating the variables in the formulas to the variables in a diagram.

One the other hand, seeing yet another notation may confuse you, so if you already feel comfortable with a set of update formulas, you will not gain by reading these notes.

The sigmoid function

The sigmoid function, $y = 1/(1 + e^{-x})$, is used instead of a step function in artificial neural nets because the sigmoid is continuous, whereas a step function is not, and you need continuity whenever you want to use gradient ascent. Also, the sigmoid function has several desirable qualities. For example, the sigmoid function's value, y, approaches 1 as x becomes highly positive; 0 as x becomes highly negative; and equals $1/2$ when $x = 0$.

Better yet, the sigmoid function features a remarkably simple derivative of the output, y, with respect to the input, x:

$$\begin{aligned}
\frac{dy}{dx} &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{d}{dx}(1+e^{-x})^{-1} \\
&= -1 \times (1+e^{-x})^{-2} \times e^{-x} \times -1 \\
&= \frac{1}{1+e^{-x}} \times \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \times \frac{1+e^{-x}-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \times \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= y(1-y)
\end{aligned}$$

Thus, remarkably, the derivative of the output with respect to the input is expressed as a simple function of the output.

The performance function

The standard performance function for gauging how well a neural net is doing is given by the following:

$$P = -\frac{1}{2}(d_{\text{sample}} - o_{\text{sample}})^2$$

where P is the performance function, d_{sample} is the desired output for some specific sample and o_{sample} is the observed output for that sample. From this point forward, assume that d and o are the desired and observed outputs for a specific sample so that we need not drag a subscript around as we work through the algebra.

The reason for choosing the given formula for P is that the formula has convenient properties. The formula yields a maximum at $o = d$ and monotonically decreases as o deviates from d. Moreover, the derivative of P with respect to o is simple:

$$\begin{aligned}\frac{dP}{do} &= \frac{d}{do}[-\frac{1}{2}(d-o)^2] \\ &= -\frac{2}{2} \times (d-o)^1 \times -1 \\ &= d - o\end{aligned}$$

Gradient ascent

Backpropagation is a specialization of the idea of gradient ascent. You are trying to find the maximum of a performance function P, by changing the weights associated with neurons, so you move in the direction of the gradient in a space that gives P as a function of the weights, w. That is, you move in the direction of most rapid ascent if we take a step in the direction with components governed by the following formula, which shows how much to change a weight, w, in terms of a partial derivative:

$$\Delta w \propto \frac{\partial P}{\partial w}$$

The actual change is influenced by a rate constant, α; accordingly, the new weight, w', is given by the following:

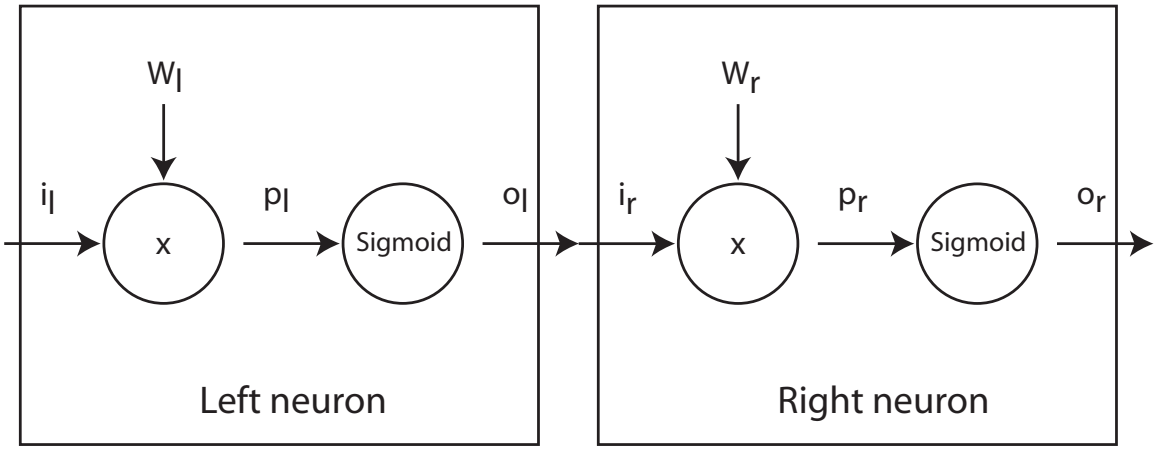
$$w' = w + \alpha \times \frac{\partial P}{\partial w}$$

Gradient descent

If the performance function were $\frac{1}{2}(d_{\text{sample}} - o_{\text{sample}})^2$ instead of $-\frac{1}{2}(d_{\text{sample}} - o_{\text{sample}})^2$, then you would be searching for the minimum rather than the maximum of P, and the change in w would be subtracted from w instead of added, so w' would be $w - \alpha \times \frac{\partial P}{\partial w}$ instead of $w + \alpha \times \frac{\partial P}{\partial w}$. The two sign changes, one in the performance function and the other in the update formula cancel, so in the end, you get the same result whether you use gradient ascent, as I prefer, or gradient descent.

The simplest neural net

Consider the simplest possible neural net: one input, one output, and two neurons, the left neuron and the right neuron. A net with two neurons is the smallest that illustrates how the derivatives can be computed layer by layer.



Note that the subscripts indicate layer. Thus, i_l, w_l, p_l, and o_l are the input, weight, product, and output associated with the neuron on the left while i_r, w_r, p_r, and o_r are the input, weight, product, and output associated with the neuron on the right. Of course, $o_l = i_r$.

Suppose that the output of the right neuron, o_r, is the value that determines performance P. To compute the partial derivative of P with respect to the weight in the right neuron, w_r, you need the chain rule, which allows you to compute partial derivatives of one variable with respect to another in terms of an intermediate variable. In particular, for w_r, you have the following, taking o_r to be the intermediate variable:

$$\frac{\partial P}{\partial w_r} = \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial w_r}$$

Now, you can repeat, using the chain-rule to turn $\frac{\partial o_r}{\partial w_r}$ into $\frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial w_r}$:

$$\frac{\partial P}{\partial w_r} = \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial w_r}$$

Conveniently, you have seen two of the derivatives already, and the third, $\frac{\partial p_r}{\partial w_r} = \frac{\partial (w_r \times o_l)}{\partial w_r}$, is easy to compute:

$$\frac{\partial P}{\partial w_r} = [(d - o_r)] \times [o_r(1 - o_r)] \times [i_r]$$

Repeating the analysis for w_l yields the following. Each line is the same as the previously, except that one more partial derivative is expanded using the chain rule:

$$\begin{aligned}
\frac{\partial P}{\partial w_l} =& \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial w_l} \\
=& \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial w_l} \\
=& \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l} \times \frac{\partial o_l}{\partial w_l} \\
=& \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l} \times \frac{\partial o_l}{\partial p_l} \times \frac{\partial p_l}{\partial w_l} \\
=& [(d - o_r)] \times [o_r(1 - o_r)] \times [w_r] \times [o_l(1 - o_l)] \times [i_l]
\end{aligned}$$

Thus, the derivative consists of products of terms that have already been computed and terms in the vicinity of w_l. This is clearer if you write the two derivatives next to one another:

$$\frac{\partial P}{\partial w_r} = (d - o_r) \times o_r(1 - o_r) \times i_r$$
$$\frac{\partial P}{\partial w_l} = (d - o_r) \times o_r(1 - o_r) \times w_r \times o_l(1 - o_l) \times i_l$$

You can simplify the equations by defining δs as follows, where each delta is associated with either the left or right neuron:

$$\delta_r = o_r(1 - o_r) \times (d - o_r)$$
$$\delta_l = o_l(1 - o_l) \times w_r \times \delta_r$$

Then, you can write the partial derivatives with the δs:

$$\frac{\partial P}{\partial w_r} = i_r \times \delta_r$$
$$\frac{\partial P}{\partial w_l} = i_l \times \delta_l$$

If you add more layers to the front of the network, each weight has a partial derivatives that is computed like the partial derivative of the weight of the left neuron. That is, each has a partial derivative determined by its input and its delta, where its delta in turn is determined by its output, the weight to its right, and the delta to its right. Thus, for the weights in the final layer, you compute the change as follows, where I use f as the subscript instead of r to emphasize that the computation is for the neuron in the final layer:

$$\Delta w_f = \alpha \times i_f \times \delta_f$$

where

$$\delta_f = o_f(1 - o_f) \times (d - o_f)$$

For all other layers, you compute the change as follows:

$$\Delta w_l = \alpha \times i_l \times \delta_l$$

where

$$\delta_l = o_l(1 - o_l) \times w_r \times \delta_r$$

More neurons per layers

Of course, you really want back propagation formulas for not only any number of layers but also for any number of neurons per layer, each of which can have multiple inputs, each with its own weight. Accordingly, you need to generalize in another direction, allowing multiple neurons in each layer and multiple weights attached to each neuron.

The generalization is an adventure in summations, with lots of subscripts to keep straight, but in the end, the result matches intuition. For the final layer, there may be many neurons, so the formula's need an index, k, indicating which final node neuron is in play. For any weight contained

in the final-layer neuron, f_k, you compute the change as follows from the input corresponding to the weight and from the δ associated with the neuron:

$$\begin{aligned}\Delta w =& \alpha \times i \times \delta_{f_k} \\ \delta_{f_k} =& o_{f_k}(1 - o_{f_k}) \times (d_k - o_{f_k})\end{aligned}$$

Note that the output of each final-layer neuron output is subtracted from the output desired for that neuron.

For other layers, there may also be many neurons, and the output of each may influence all the neurons in the next layer to the right. The change in weight has to account for what happens to all of those neurons to the right, so a summation appears, but otherwise you compute the change, as before, from the input corresponding to the weight and from the δ associated with the neuron:

$$\begin{aligned}\Delta w =& \alpha \times i \times \delta_{l_i} \\ \delta_{l_i} =& o_{l_i}(1 - o_{l_i}) \times \sum_j w_{l_i \to r_j} \times \delta_{r_j}\end{aligned}$$

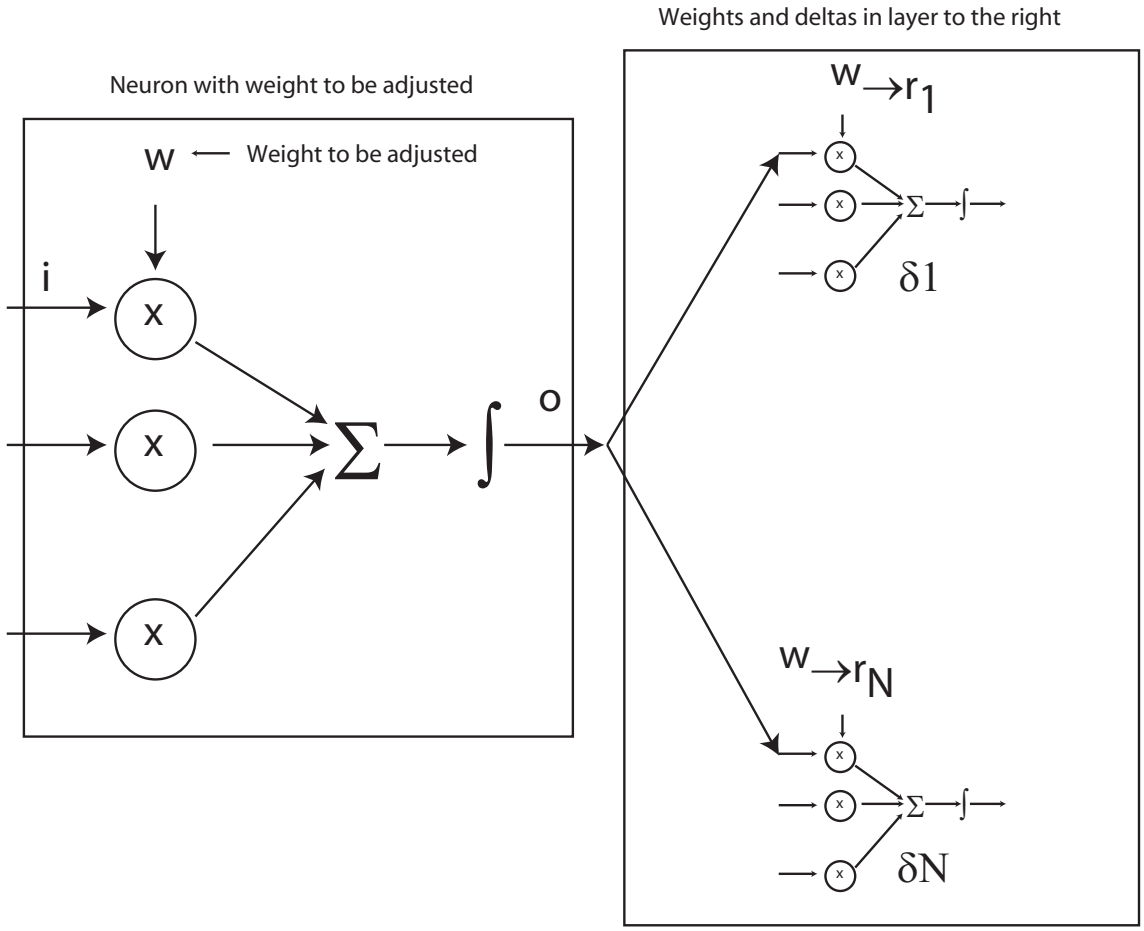
Note that $w_{l_i \to r_j}$ is the weight that connects the j^{th} right-side neuron to the output of the i^{th} left-side neuron.

Summary

Once you understood how to derive the formulas, you can combine and simplify them in preparation for solving problems. For each weight, you compute the weight's change from the input corresponding to the weight and from the δ associated with the neuron. Assuming that δ is the delta associated with that neuron, you have the following, where $w_{\to r_j}$ is the weight connecting the output of the neuron you are working on, the i^{th} left-side neuron, to the j^{th} right-side neuron, and δ_{r_j} is the δ associated with that right-side neuron.

$$\begin{aligned}\delta_o =& o(1 - o) \times (d - o) && \text{for the final layer} \\ \delta_{l_i} =& o_{l_i}(1 - o_{l_i}) \times \sum_j w_{l_i \to r_j} \times \delta_{r_j} && \text{otherwise}\end{aligned}$$

That is, you computed change in a neuron's w, in every layer, by multiplying α times the neuron's input times its δ. The δ is determined for all but the final layer in terms of the neuron's output and all the weights that connect that output to neurons in the layer to the right and the δs associated with those right-side neurons. The δ for each neuron in the final layer is determined only by the output of that neuron and by the difference between the desired output and the actual output of that neuron.


Weights and deltas in layer to the right
Neuron with weight to be adjusted
w
Weight to be adjusted
i
x
x
x
Σ
∫
o
w → r1
δ1
w → rN
δN

6.034 Artificial Intelligence

Fall 2010