

One Last Thing...

In this lesson, we will try to find a similar error slope for the weights between the input and hidden layers just like we did for the output layer.

One almost final bit of work to do.

That expression we slaved over is for refining the weights between the hidden and output layers. We now need to finish the job and find a similar error slope for the weights between the input and hidden layers. We could do loads of algebra again, but we don't have to. We simply use that physical interpretation we just did and rebuild an expression for the new set of weights we're interested in. So this time,

- The first part which was the (target - actual) error now becomes the recombined back-propagated error out of the hidden nodes, just as we saw above. Let's call that e_j .
- The sigmoid parts can stay the same, but the sum expressions inside refer to the preceding layers, so the sum is over all the inputs moderated by the weights into a hidden node j . We could call this i_j .
- The last part is now the output of the first layer of nodes o_i , which happen to be the input signals.

This nifty way of avoiding lots of work is simply taking advantage of the symmetry in the problem to construct a new expression. We say it's simple, but it is a very powerful technique, wielded by some of the most big-brained mathematicians and scientists. You can certainly impress your mates with this! So the second part of the final answer we've been striving towards is as follows, the slope of the error function for the weights between the input and hidden layers.

$$\frac{\partial E}{\partial w_{ij}} = -(e_j) \cdot \text{sigmoid}(\sum_i w_{ij} \cdot o_i) (1 - \text{sigmoid}(\sum_i w_{ij} \cdot o_i)) \cdot o_i$$


We've now got all these crucial magic expressions for the slope; we can use them to update the weights after each training example, as we'll see next. Remember the weights are changed in a direction opposite to the gradient, as we saw clearly in the diagrams earlier. We also moderate the change by using a learning factor, which we can tune for a particular problem. We saw this too when we developed linear classifiers as a way to avoid being pulled too far wrong by bad training examples, but also to ensure the weights don't bounce around a minimum by constantly overshooting it. Let's say this in mathematical form.

$$\text{new } w_{jk} = \text{old } w_{jk} - \alpha \cdot \frac{\partial E}{\partial w_{jk}}$$

The updated weight w_{jk} is the old weight adjusted by the negative of the error slope we just worked out. It's negative because we want to increase the weight if we have a positive slope, and decrease it if we have a negative slope, as we saw earlier. The symbol alpha α , is a factor which moderates the strength of these changes to make sure we don't overshoot. It's often called a *learning rate*. This expression applies to the weights between the input and hidden layers too, not just to those between the hidden and output layers. The difference will be the error gradient, for which we have the two expressions above.

Before we can leave this, we need to see what these calculations look like if we try to do them as matrix multiplications. To help us, we'll do what we did before, which is to write out what each element of the weight change matrix should be.

$$\begin{pmatrix} \Delta w_{1,1} & \Delta w_{1,2} & \Delta w_{1,3} & \dots \\ \Delta w_{2,1} & \Delta w_{2,2} & \Delta w_{2,3} & \dots \\ \Delta w_{3,1} & \Delta w_{3,2} & \Delta w_{j,k} & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix} = \begin{pmatrix} E_1 * S_1 (1-S_1) \\ E_2 * S_2 (1-S_2) \\ E_k * S_k (1-S_k) \\ \dots \end{pmatrix} \cdot \begin{pmatrix} o_1 & o_2 & o_j & \dots \end{pmatrix}$$



values from previous layer

I have left out the learning rate α as that's just a constant and doesn't really change how we organize our matrix multiplication. The matrix of weight changes contains values which will adjust the weight $w_{j,k}$ linking node j in one layer with the node k in the next. You can see that first bit of the expression uses values from the next layer (node k), and the last bit uses values from the previous layer (node j).

You might need to stare at the picture above for a while to see that the last part, the horizontal matrix with only a single row, is the transpose of the outputs from the previous layer O_j . The color coding shows that the dot product is the right way around. If you're not sure, try writing out the dot product with these the other way around, and you'll see it doesn't work. So, the matrix form of these weight update matrices is as follows, ready for us to implement in a computer programming language that can work with matrices efficiently.

$$\Delta W_{jk} = \alpha \cdot E_k \cdot O_k (1 - O_k) \cdot O_j^T$$

It is actually not a complicated expression at all. Those sigmoids have disappeared because they were simply the node outputs O_k . That's it! Job done.