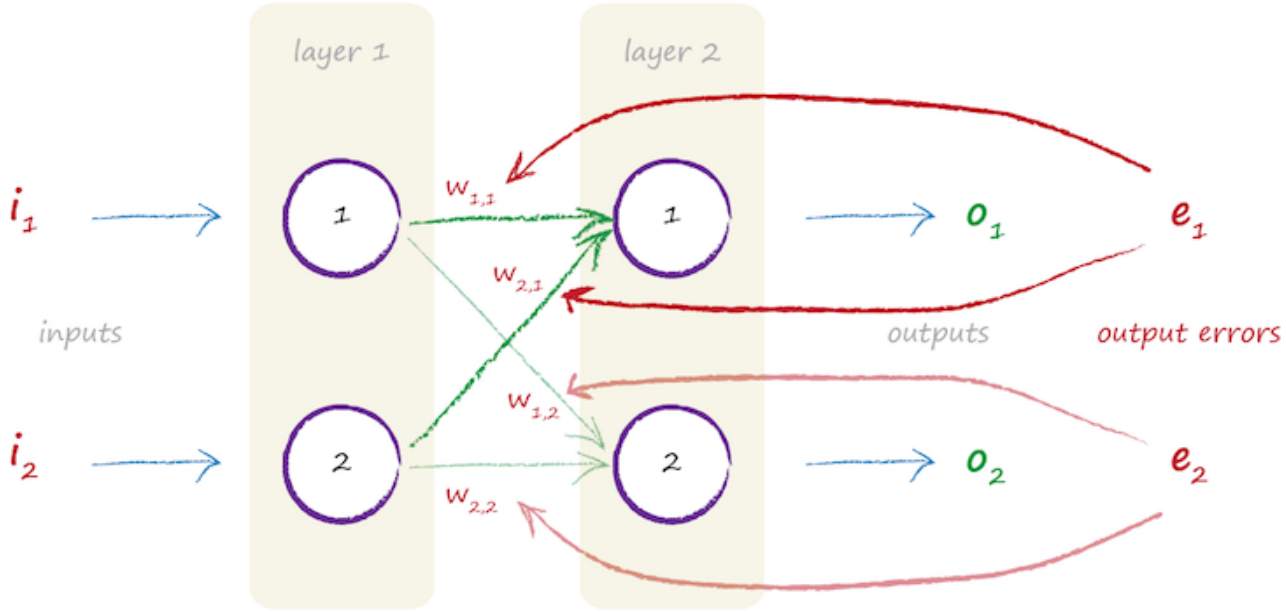
Backpropagating Errors From More Output Nodes

In this lesson, we will learn how to propagate the error backward from the output node back into the network. This method is extensively used when more there are more than one nodes that contributes to the output.

The following diagram shows a simple network with two input nodes, but now with two output nodes.



Both output nodes can have an error — in fact, that's extremely likely when we haven't trained the network. You can see that both of these errors need to inform the refinement of the internal link weights in the network. We can use the same approach as before, where we split an output node's error amongst the contributing links, in a way that's proportionate to their weights. The fact that we have more than one output node doesn't really change anything. We simply repeat for the second output node what we already did for the first one. Why is this so simple? It is simply because the links into an output node don't depend on the links into another output node. There is no dependence between these two sets of links.

Looking at that diagram again, we have labeled the error at the first output node as e_1. Remember this is the difference between the desired output

provided by the training data t_1 and the actual output o_1. That is,

$e_1 = (t_1 - o_1)$. The error at the second output node is labeled e_2. You can see from the diagram that the error e_1 is split in proportion to the connected links, which have weights w_{11} and w_{21}. Similarly, e_2 would be split in proportion to weights w_{12} and w_{22}.

Let's write out what these splits are, so we're not in any doubt. The error e1 is used to inform the refinement of both weights w_{11} and w_{21}. It is split so that the fraction of e1 used to update w_{11} is

$$\frac{w_{11}}{w_{11} + w_{21}}$$

Similarly the fraction of e_1 used to refine w_{21} is

$$\frac{w_{21}}{w_{11} + w_{21}}$$

These fractions might look a bit puzzling, so let's illustrate what they do. Behind all these symbols is the very simple idea that the error e1 is split to give more to the weight that is larger, and less of the weight that is smaller. If w11 is twice as large as w_{21}, say w_{11}=6 and w_{21}=3, then the fraction of e_1 used to update w_{11} is $6/(6+3) = 6/9 = 2/3$. That should leave $1/3$ of e_1 for the other smaller weight w_{21} which we can confirm using the expression $3/(6+3) = 3/9$ which is indeed 1/3. If the weights were equal, the fractions will be half, as you'd expect. Let's see this just to be sure. Let's say w_{11}=4 and w_{21}=4, then the fraction is $4/(4+4) = 4/8 = 1/2$ for both cases.

Before we go further, let's pause and take a step back and see what we've done from a distance. We knew we needed to use the error to guide the refinement of some parameter inside the network, in this case, the link weights. We've just seen how to do that for the link weights which moderate signals into the final output layer of a neural network. We've also seen that

there isn’t a complication when there is more than one output node, we just do the same thing for each output node. Great! The next question to ask is what happens when we have more than two layers? How do we update the link weights in the layers further back from the final output layer?