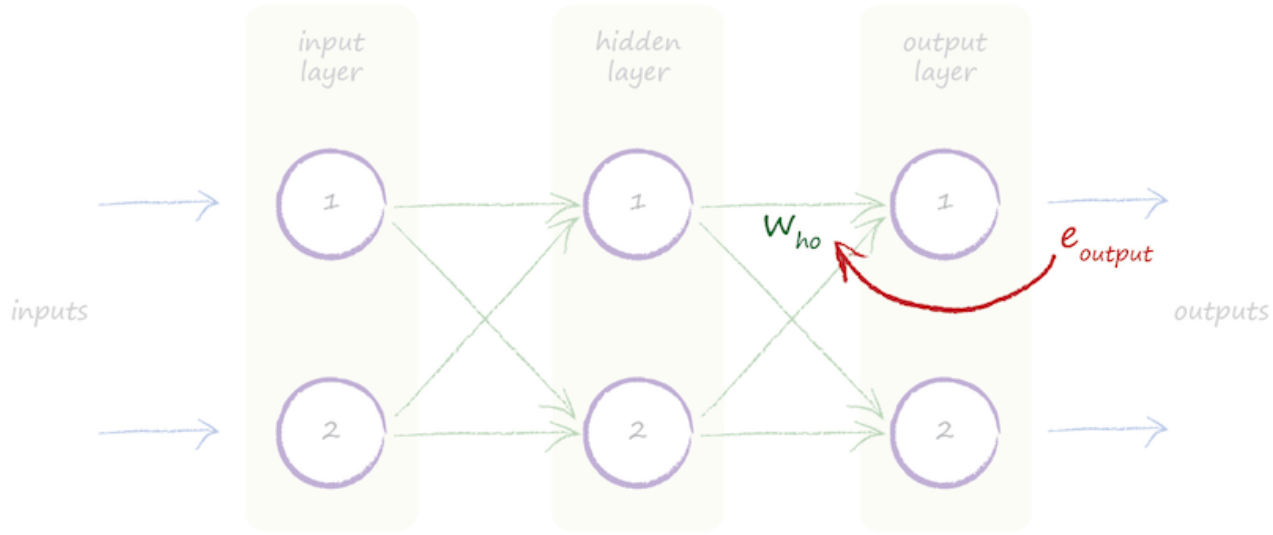
Backpropagation: Splitting the Error

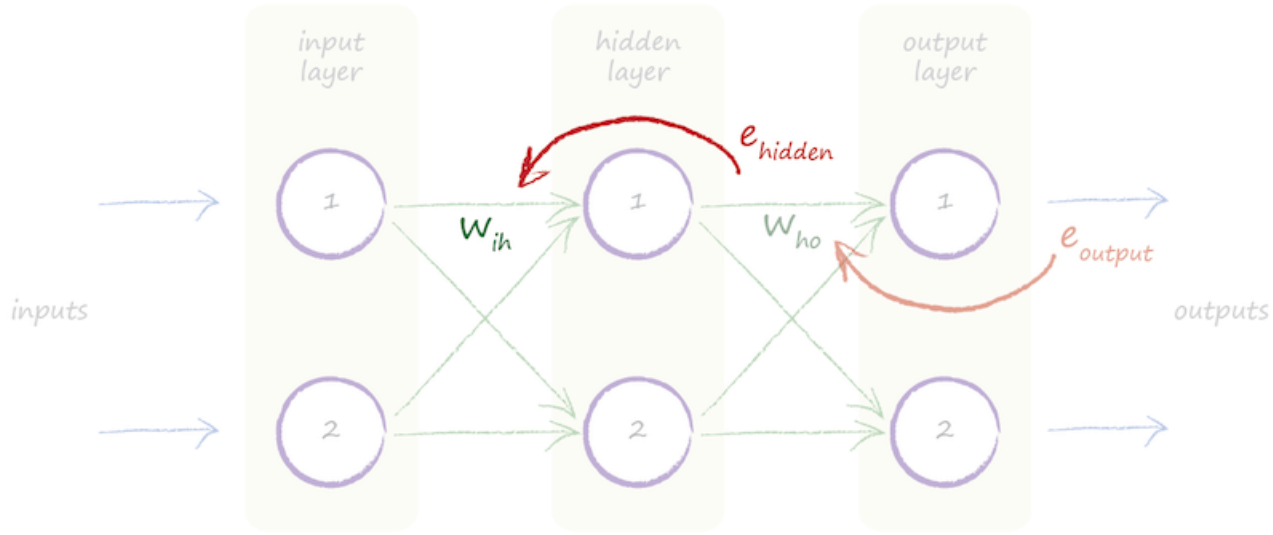
Neural networks learn by refining their link weights. So to backpropagate the error to internal nodes, we split the output layer errors in proportion to the size of the connected link weights, and then recombine these bits at each internal node.

The following diagram shows a simple neural network with three layers, an input layer, a hidden layer and the final output layer.



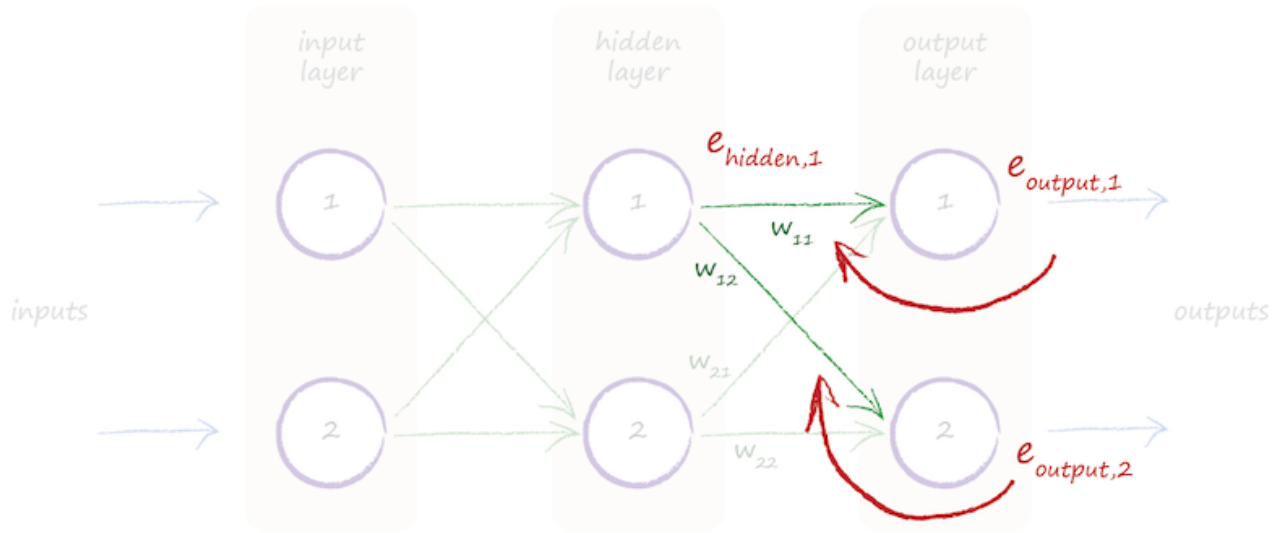
Working back from the final output layer at the right-hand side, we can see that we use the errors in that output layer to guide the refinement of the link weights feeding into the final layer. We've labeled the output errors more generically as e_{output} and the weights of the links between the hidden and output layer as w_{ho}. We worked out the specific errors associated with each link by splitting the errors in proportion to the size of the weights themselves.

By showing this visually, we can see what we need to do for the new additional layer. We simply take those errors associated with the output of the hidden layer nodes e_{hidden} and split those again proportionately across the preceding links between the input and hidden layers w_{ih}. The next diagram shows this logic.



If we had even more layers, we'd repeatedly apply this same idea to each layer working backward from the final output layer. The flow of error information makes intuitive sense. You can see again why this is called error *backpropagation*. If we first used the error in the output of the output layer nodes e_{output}, what error do we use for the hidden layer nodes e_{hidden}? This is a good question to ask because a node in the middle hidden layer doesn't have an obvious error. We know from feeding forward the input signals that, yes, each node in the hidden layer does indeed have a single output. You'll remember that was the activation function applied to the weighted sum of the inputs to that node. But how do we work out the error?

We don't have the target or desired outputs for the hidden nodes. We only have the target values for the final output layer nodes, and these come from the training examples. Let's look at that diagram above again for inspiration! That first node in the hidden layer has two links emerging from it to connect it to the two output layer nodes. We know we can split the output error along each of these links, just as we did before. That means we have some kind of error for each of the two links that emerge from this middle layer node. We could recombine these two link errors to form the error for this node as a second-best approach because we don't actually have a target value for the middle layer node. The following shows this idea visually.



You can see more clearly what's happening, but let's go through it again just to be sure. We need an error for the hidden layer nodes so we can use it to update the weights in the preceding layer. We call these e_{hidden}. But we don't have an obvious answer to what they actually are. We can't say the error is the difference between the desired target output from those nodes and the actual outputs because our training data examples only give us targets for the very final output nodes.