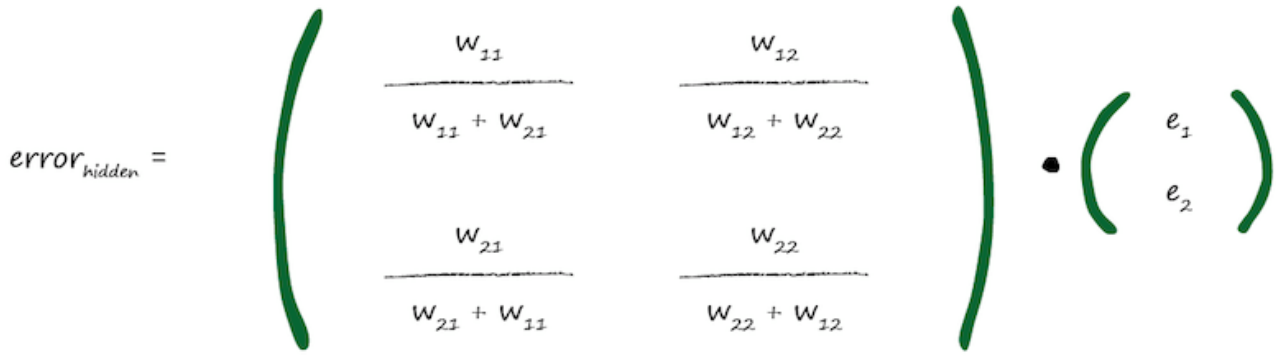
Backpropagating Errors with Matrix Multiplication

Backpropagating the error can be expressed as a matrix multiplication. It makes feeding signals forward and error backpropagation quite efficient.

Can we use matrix multiplication to simplify all that laborious calculation? It helped earlier when we were doing loads of calculations to feed forward the input signals. To see if error backpropagation can be made more concise using matrix multiplication, let's write out the steps using symbols. By the way, this is called trying to vectorize the process. Being able to express a lot of calculations in matrix form makes it more concise for us to write down, and also allows computers to do all that work much more efficiently because they take advantage of the repetitive similarities in the calculations that need to be done. The starting point is the errors that emerge from the neural network at the final output layer. Here we only have two nodes in the output layer, so these are e_1 and e_2.

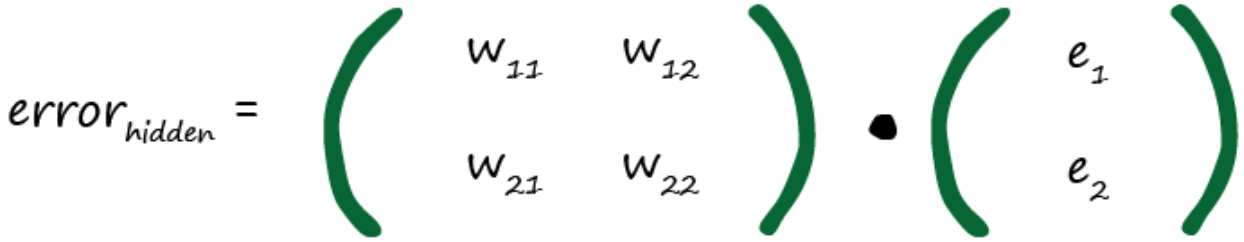
$$error_{output} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Next, we want to construct the matrix for the hidden layer errors. That might sound hard so let's do it bit by bit. The first bit is the first node in the hidden layer. If you look at the diagrams above again you can see that the first hidden node's error has two paths contributing to it from the output layer. Along these paths come the error signals $e_1 * w_{11}/(w_{11} + w_{21})$ and $e_2 * w_{12}/(w_{12} + w_{22})$. Now look at the second hidden layer node and we can again see two paths contributing to its error, $e_1 * w_{21}/(w_{21} + w_{11})$ and $e_2 * w_{22}/(w_{22} + w_{12})$. We've already seen how these expressions are worked out earlier. So we have the following matrix for the hidden layer. It's a bit more complex than I'd like.

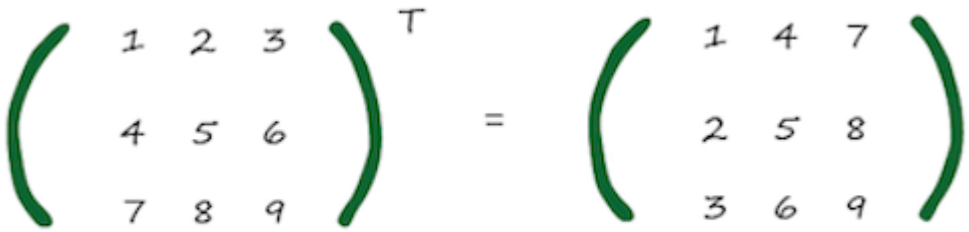


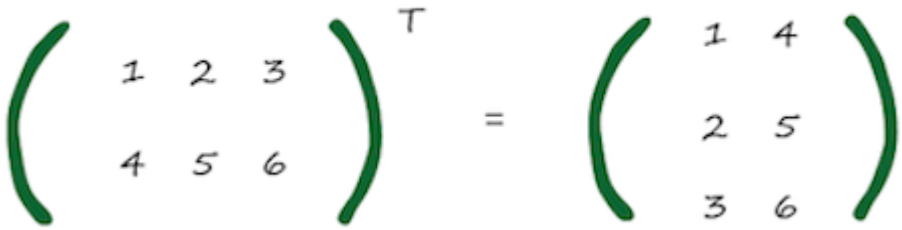
It would be great if this could be rewritten as a very simple multiplication of matrices we already have available to us. These are the weight, forward signal and output error matrices. Remember the benefits are huge if we can do this. Sadly we can't easily turn this into a super simple matrix multiplication like we could with the feeding forward of signals we did earlier. Those fractions in that big busy matrix above are hard to untangle! It would have been helpful if we could neatly split that busy matrix into a simple combination of the available matrices.

What can we do? We still really really want those benefits of using matrix multiplications to do the calculations efficiently. Time to get a bit naughty! Have a look again at that expression above. You can see that the most important thing is the multiplication of the output errors e_n with the linked weights w_{ij}.The larger the weight, the more of the output error is carried back to the hidden layer. That's the important bit. The bottom of those fractions is a kind of normalizing factor. If we ignored that factor, we'd only lose the scaling of the errors being fed back. That is, $e_1 * w_{11} / (w_{11} + w_{21})$ would become the much simpler $e_1 * w_{11}$. If we did that, then the matrix multiplication is easy to spot. Here it is:



That weight matrix is like the one we constructed before but has been flipped along a diagonal line so that the top right is now at the bottom left, and the bottom left is at the top right. This is called *transposing* a matrix and is written as w^T. Here are two examples of a transposing matrix of numbers, so we can see clearly what happens. You can see this works even when the matrix has a different number of rows from columns.





So we have what we wanted, a matrix approach to propagating the errors back:

$$error_{hidden} = w^T_{hidden\_output} \cdot error_{output}$$

This is great but did we do the right thing cutting out that normalizing factor? It turns out that this simpler feedback of the error signals works just as well as the more sophisticated one we worked out earlier. This book's blog has a post showing the results of several different ways of back-propagating the error. If our simpler way works really well, we'll keep it! If we want to think about this more, we can see that even if overly large or small errors are fed back, the network will correct itself during the next iterations of learning. The important thing is that the errors being fed back respect the strength of the link weights because that is the best indication we have of sharing the blame

for the error. We've done a lot of work, a huge amount! Take a well-deserved

break, because the next and final theory section is really very cool but does require a fresh brain.