Refining the Weights

In this lesson, we will try to refine the weights. We will be using the Numpy library to calculate the dot product.

So we have what we need to refine the weights at each layer. For the weights between the hidden and final layers, we use the `output_errors`. For the weights between the input and hidden layers, we use these `hidden_errors` we just calculated. We previously worked out the expression for updating the weight for the link between a node *j* and a node *k* in the next layer in matrix form:

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

The *alpha* is the *learning rate*, and the sigmoid is the squashing activation function we saw before. Remember that the * multiplication is the normal element by element multiplication, and the • dot is the matrix dot product. That last bit, the matrix of outputs from the previous layer, is transposed. In effect, this means the column of outputs becomes a row of outputs. That should translate nicely into Python code. Let's do the code for the weights between the hidden and final layers first:

```
# update the weights for the links between the hidden and output layers
self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), nump
```

That's a long line of code, but the color coding should help you see how it relates back to that mathematical expression. The learning rate is `self.lr` and simply multiplied by the rest of the expression. There is a matrix multiplication done by `numpy.dot()`, and the two elements are colored red and green to show the part related to the error and sigmoids from the next layer, and the transposed outputs from the previous layer. That += simply means and the transposed outputs from the previous layer. That += simply means increasing the preceding variable by the next amount. So $x += 3$ means

increase x by 3. It's just a shorter way of writing $x = x + 3$. You can use other arithmetic too so $x/ = 3$ means divide x by 3.

The code for the other weights between the input and hidden layers will be very similar. We just exploit the symmetry and rewrite the code replacing the names so that they refer to the previous layers. Here is the code for both sets of weights, colored so you can see the similarities and differences:

```
# update the weights for the links between the hidden and output layers
self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), nump

# update the weights for the links between the input and hidden layers
self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), nu
```