Training the Network

In this lesson, we will try to come up with a solution to train our network.

Let's now tackle the slightly more involved training task. There are two parts to this:

- The first part is working out the output for a given training example. That is no different to what we just did with the `query()` function.
- The second part is taking this calculated output, comparing it with the desired output, and using the difference to guide the updating of the network weights.

We have already done the first part so let's write that out:

```
# train the neural network
def train(self, inputs_list, targets_list):
    # convert inputs list to 2d array
    inputs = numpy.array(inputs_list, ndmin=2).T
    targets = numpy.array(targets_list, ndmin=2).T

    # calculate signals into hidden layer
    hidden_inputs = numpy.dot(self.wih, inputs)
    # calculate the signals emerging from hidden layer
    hidden_outputs = self.activation_function(hidden_inputs)

    # calculate signals into final output layer
    final_inputs = numpy.dot(self.who, hidden_outputs)
    # calculate the signals emerging from final output layer
    final_outputs = self.activation_function(final_inputs)

    pass
```

This code is almost exactly the same as that in the `query()` function because we're feeding forward the signal from the input layer to the final output layer in exactly the same way. The only difference is that we have an additional parameter, `targets_list`, defined in the function name because you can't train the network without the training examples which include the desired or

target answer.

```
def train(self, inputs_list, targets_list)
```

The code also turns the `targets_list` into a *numpy* array, just as the `inputs_list` is turned into a *numpy* array.

```
targets = numpy.array(targets_list, ndmin=2).T
```

Now we are getting closer to the heart of the neural network's working, improving the weights based on the error between the calculated and target output. Let's do this in gentle manageable steps.

First, we need to calculate the error, which is the difference between the desired target output provided by the training example, and the actual calculated output. That's the difference between the matrices $(targets - final\ outputs)$ done element by element. The Python code for this is really simple, showing again the elegant power of matrices.

```
# error is the (target - actual)
output_errors = targets - final_outputs
```

We can calculate the back-propagated errors for the hidden layer nodes. Remember how we split the errors according to the connected weights, and recombine them for each hidden layer node. We worked out the matrix form of this calculation as

$$errors_{hidden} = weights^{T}_{hidden\_output}.errors_{output}$$

The code for this is again simple because of Python's ability to do matrix dot products using *numpy*.

```
# hidden layer error is the output_errors, split by weights, recombined at hidden nodes
hidden_errors = numpy.dot(self.who.T, output_errors)
```