Querying the Network

In this lesson, we will work on the query function which is the first to train the network.

It sounds logical to start working on the code that trains the neural network next, by fleshing out the currently empty `train()` function. We will delay doing that, and instead, work on the simpler `query()` function. This will give us more time to gradually build confidence and get some practice at using both Python and these weight matrices inside the neural network object.

The `query()` function takes the input to a neural network and returns the network's output. That's simple enough, but to do that you will remember that we need to pass the input signals from the input layer of nodes, through the hidden layer and out of the final output layer. You'll also remember that we use the link weights to moderate the signals as they feed into any given hidden or output node, and we also use the sigmoid activation function to squish the signal coming out of those nodes.

If we had lots of nodes, we would have a horrible task on our hands, writing out the Python code for each of those nodes, doing the weight moderating, summing signals, applying the activation function. And the more nodes, the more code. That would be a nightmare!

Luckily we don't have to do that because we worked out how to write all these instructions in a simple, concise matrix form. The following shows how the matrix of weights for the link between the input and hidden layers can be combined with the matrix of inputs to give the signals to the hidden layer nodes:

$$X_{hidden} = W_{input\_hidden}.I$$

The great thing about this was not just that it was easier for us to write down, but that programming languages like Python can also understand matrices and do all the real work quite efficiently because they recognize the similarities between all the underlying calculations.

You'll be amazed at how simple the Python code actually is. The following applies the *numpy* library's dot product function for matrices to the link weights W_{input_hidden} and the inputs $\mathbf{I}$.

```
hidden_inputs = numpy.dot(self.wih, inputs)
```

That's it!

That simple piece of Python does all the work of combining all the inputs with all the right link weights to produce the matrix of combined moderated signals into each hidden layer node. We don't have to rewrite it either if next time we choose to use a different number of nodes for the input or hidden layers. It just works!

This power and elegance is why we put the effort into trying to understand the matrix multiplication approach earlier.