Weights - The Heart of the Network

The next step is to create the network of nodes and links. The most important part of the network is the link weights. They're used to calculate the signal being fed forward, the error as it's propagated backward, and it is the link weights themselves that are refined in an attempt to improve the network.

We saw earlier that the weights could be concisely expressed as a matrix. So we can create:

- A matrix for the weights for links between the input and hidden layers, W_{input_hidden}, of size (*hidden_nodes* by *input_nodes*).
- And another matrix for the links between the hidden and output layers, W_{hidden_output}, of size (*output_nodes* by *hidden_nodes*).

Remember the convention earlier to see why the first matrix is of size (*hidden_nodes* by *input_nodes*) and not the other way around (*input_nodes* by *hidden_node*).

Remember from Part 1 of this course that the initial values of the link weights should be small and random. The following *numpy* function generates an array of values selected randomly between 0 and 1, where the size is (rows by columns).

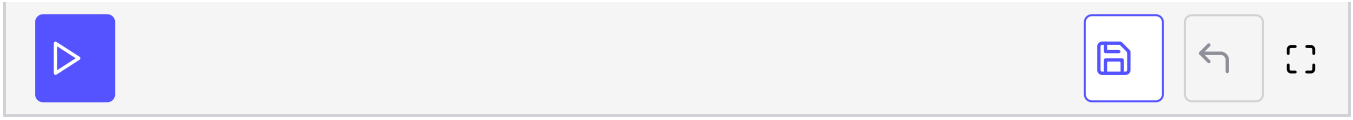
```
numpy.random.rand(rows, columns)
```

All good programmers use internet search engines to find online documentation on how to use cool Python functions, or even to find useful functions they didn't know existed. Google is particularly good for finding stuff about programming. This `numpy.random.rand()` function is described here, for example.

If we're going to use the *numpy* extensions, we need to import the library at the top of our code. Try this function, and confirm for yourself it works. The following shows it working for a (3 by 3) *numpy* array. You will see that each

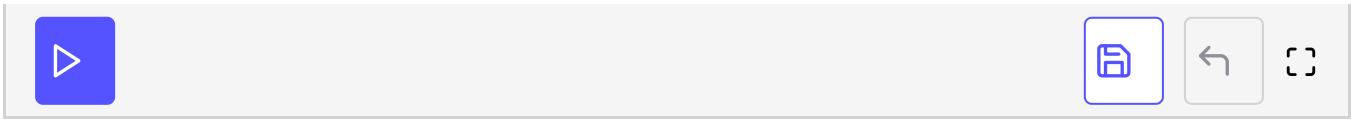
value in the array is random and between 0 and 1.

```
import numpy
print(numpy.random.rand(3,3))
```



We can do better because we have ignored the fact that the weights could legitimately be negative not just positive. The range could be between -1.0 and $+1.0$. For simplicity, we'll simply subtract 0.5 from each of the above values to in effect have a range between -0.5 to $+0.5$. The following shows this neat trick working, and you can see some random values below zero.

```
import numpy
print(numpy.random.rand(3,3) - 0.5)
```



We're ready to create the initial weight matrices in our Python program. These weights are an intrinsic part of a neural network and live with the neural network for all its life, not a temporary set of data that vanishes once a function is called. That means it needs to be part of the initialization too, and accessible from other functions like the training and the querying.

The following code, with comments included, creates the two link weight matrices using the `self.inodes`, `self.hnodes` and `self.onodes` to set the right size for both of them.

```
# link weight matrices, wih and who
# weights inside the arrays are w_i_j, where link is from node i to node j in the next layer
# w11 w21
# w12 w22 etc
self.wih = (numpy.random.rand(self.hnodes, self.inodes) - 0.5)
self.who = (numpy.random.rand(self.onodes, self.hnodes) - 0.5)
```

Great work! We've implemented the very heart of a neural network, its link weight matrices!