Exercise 2 – back-propagation

In the second exercise you will experiment with the code for a multi-layer network and train it using back-propagation.

The network will have a notional Input layer, a hidden layer and an output layer.

Both the hidden and output layers will use the sigmoid transfer function.

Each layer can implement multiple neurons of the given type, and will initialise the weights randomly.

You can find description of the back-propagation algorithm in this article. The article explains the working of a short Python implementation. *The APL code you will be using is a line-by line translation of the Python code*.

This version of the code does not distinguish between weights and biases; instead it uses a version of the *bias trick* which expects an extra dummy input of 1 to the network and then changes the weight of that input instead of changing a bias. The bias trick leads to simpler code and faster execution.

Exercise

Start by defining and running the implementation of the Backprop function from the file backprop.dyalog in the *exercises* directory of the expanded zip file.

Once it's defined you can train it to learn XOR using the following test data:

```
X ← ↑ (0 0 1) (0 1 1) (1 0 1) (1 1 1)
y ← ⍪0 1 1 0
X backprop y
```

Once you have tested the code, you will probably want to start converting the code into more idiomatic APL. We'll discuss approaches to this as a group and see what you can come up with!