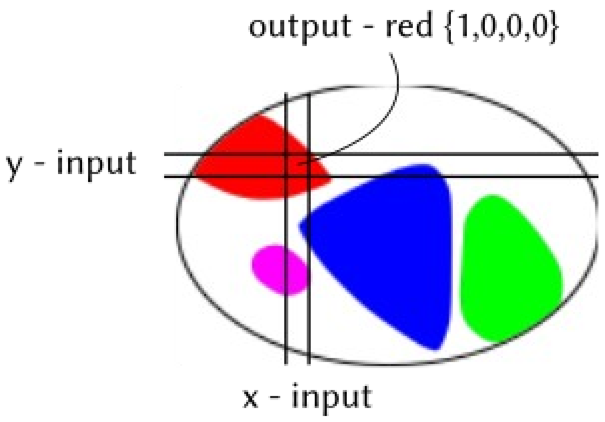
Generalised Backpropagation

The Aim

The point of a neural network is for the network to learn a set output pattern over many trials by adjusting its connections according to its error. These networks normally have layers of nodes feeding forward from inputs to outputs through a series of hidden layers, for which an effective training algorithm called Backpropagation is used. However, the network can actually have any structure of connections and still be trained through backpropagation, this might be useful when using the network as a model of other phenomena with more interesting structures such as Gene Regulatory networks. Here I'll explain my implementation of a backpropagation algorithm which will work for networks of any structure, based on a paper by Pineda (1987 http://brainmaps.org/pdf/pineda1987.pdf)

The network needs a set of data to be trained on, this can be any mapping of inputs to outputs, in our case we're using a couple of integers as inputs, and one of 5 options (Red, Blue, Green, Pink, White) as the output. If you think of the integer inputs as coordinates then we have a 2 dimensional map. To encode the colour outputs we can use 4 nodes, any of which equalling one represents a different colour so White = {0,0,0,0}, Red = {1,0,0,0}, Blue = {0,1,0,0} and so on.



The training algorithm for neural networks consists of repeating this series:
1. forward propagation of the input signal
2. measuring the error of the output
3. backward propagation of the error signal
4. updating the connection weights

In more detail, forward propagation means calculating the value of every node as the weighted sum of its inputs. In the case of a network with some cyclic (recurrent) connections, this could be repeated several times until the nodes values converge on a settled number, or an average value could be used if convergence is an issue.

Precisely, the value of each node, x, is:

$$x_j = \sigma(\sum_i w_{ij} \cdot x_i)$$

Where the sigmoid function is a logistic function to non-linearly squash the values to the range of 0 to 1, and w represents the weight of the connection from i to j.

The error of each output node is calculated, for an individual set of inputs rather than the whole mapping. For an individual node the error = $(\text{output} - \text{target})^2$. These can be added together to give a metric of the success of the network but for training this is unnecessary.

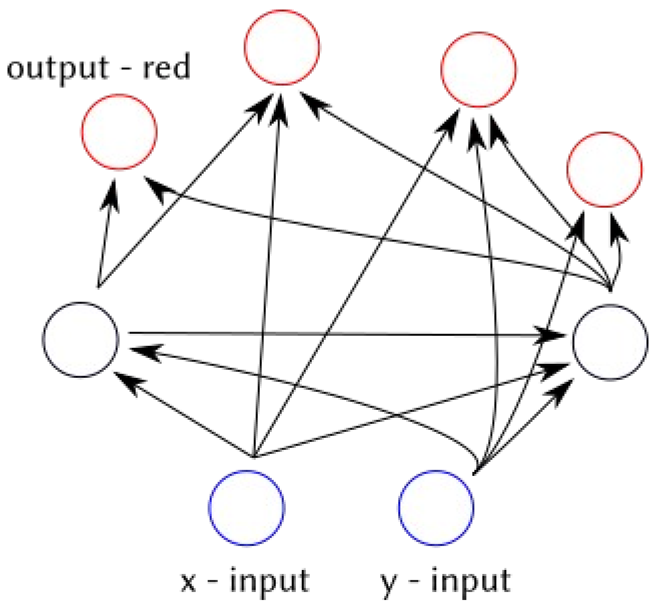
The backpropagation is the most complicated aspect of the process. Each node needs to have a 'delta' calculated, which is a measure of that node's contribution to the overall error. Once every node has a delta value the incoming weights can be updated in proportion to it. The tricky part with a recurrently structured network is tracing backwards through the connections to assign each node its portion of blame for the error. Still, it is fairly intuitive that you start with the output nodes which have a definite error value each. For these delta = 2 x learnrate x error.

Then the error is propogated one step back, for a node one connection behind the output nodes the delta is:

$$\delta_i = x_i(1 - x_i) \sum_j w_{ij} \cdot \delta_j$$

the x(1-x) part is to adjust for the sigmoid term. Once all the known deltas have fed in to this process the new deltas are 'fixed' so that they can't be changed by any further recurrent connections and so they can feed back to their input nodes, using the same delta update equation.

Once every node has a delta the weights can be updated by taking the original weight and adding $x_i * \delta_j$


output - red
x - input
y - input