ECM3412/ECMM409
Nature Inspired Computation

Lecture 14

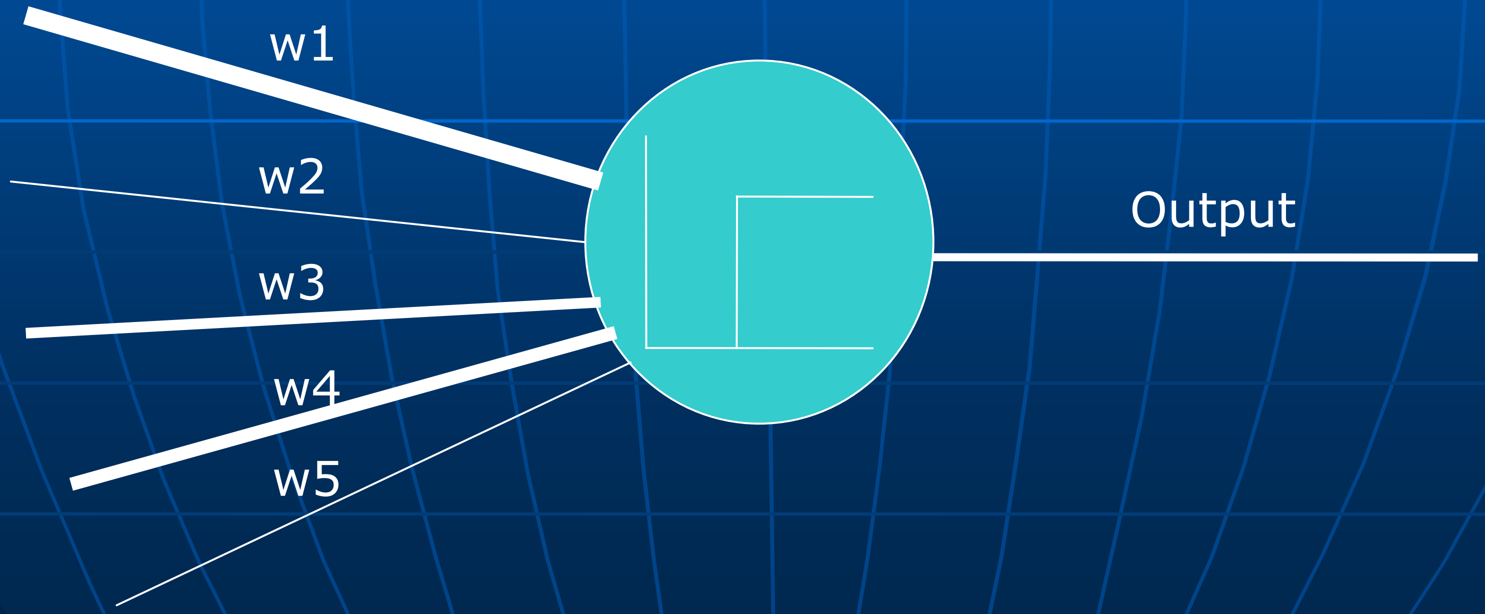
Neural Networks 2: Learning in Neural Networks

Learning in Neural Networks

- Falls broadly into two types:
  - Supervised Learning
  - Unsupervised Learning
- Supervised Learning
  - Similar to the way children learn
  - The output of the neural network is compared against the correct output
  - The network then corrects itself based on that output
- Unsupervised Learning
  - The network organises itself according to patterns in the data
  - No external 'desired output' is provided

The Perceptron

- Consists of a set of weighted connections, the neuron (incorporating the activation function) and the output axon.
- In this case, the activation function is the heaviside or threshold function



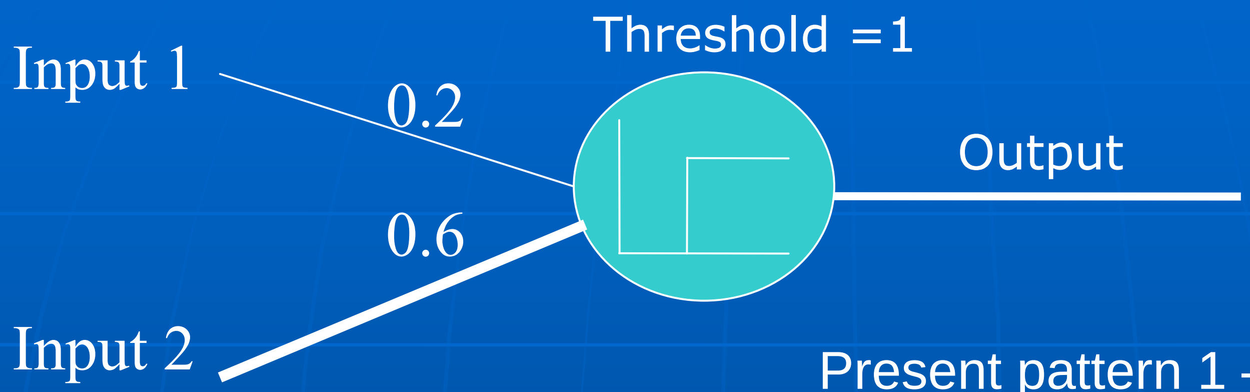
Learning in a Perceptron

- Initialise weights & threshold
- Present the input and desired output
- Calculate the actual output of the network:
- For each input:
  - Multiply the input data (x_i) by its weight (w_i).
  - Sum the weighted inputs and pass through the activation function

$$y_{pj} = f\left[\sum_{i=0}^{n-1} w_i x_i\right]$$

- Adapt the weights:
  - If correct $w_i(t+1) = w_i(t)$
  - If output 0, should be 1 $w_i(t+1) = w_i(t)+x_i(t)$
  - If output 1, should be 0 $w_i(t+1) = w_i(t)-x_i(t)$

Perceptron Learning - OR



Iteration 1

Present pattern 1 – $f_{act}(0*0.2 + 0*0.6) = 0$
desired = 0
weights stay the same

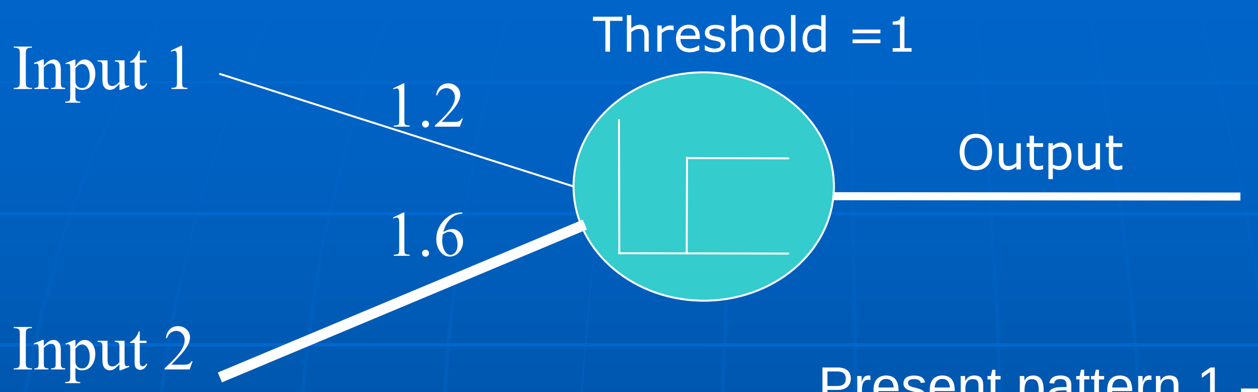
Present pattern 2 – $f_{act}(0*0.2 + 1*0.6) =0$
desired = 1
weight 1 += 0
weight 2 += 1

Present pattern 3 – $f_{act}(1*0.2 + 0*1.6) = 0$
desired = 1
weight 1 += 1
weight 2 += 0

Present pattern 4 – $f_{act}(1*1.2+1*1.6) = 1$
desired = 1
weights stay same

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Perceptron Learning Iteration 2



Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Iteration 2

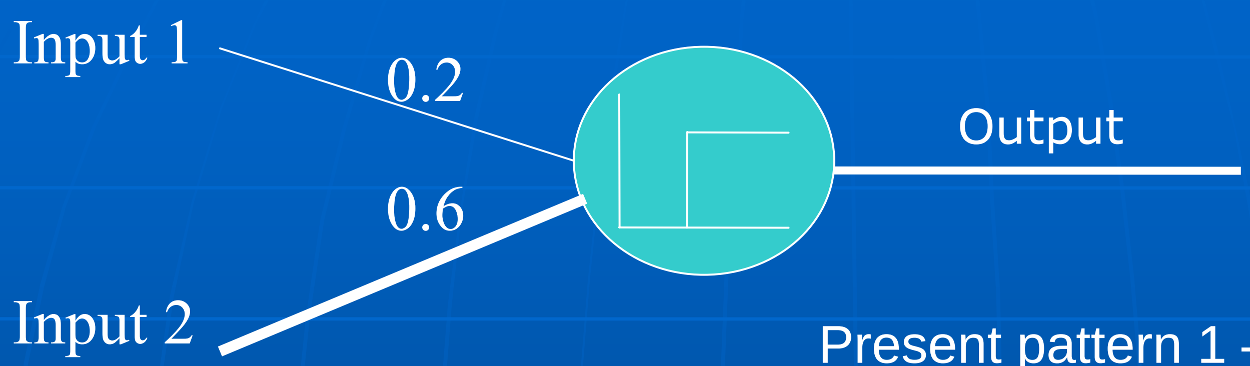
Present pattern 1 – $f_{act}(0*1.2 + 0*1.6) = 0$
desired = 0
weights stay the same

Present pattern 2 – $f_{act}(0*1.2 + 1*1.6) =1$
desired = 1
weights stay the same

Present pattern 3 – $f_{act}(1*1.2 + 0*1.6) = 1$
desired = 1
weights stay the same

Present pattern 4 – $f_{act}(1*1.2+1*1.6) = 1$
desired = 1
weights stay same

Perceptron Learning - XOR



Iteration 1

Present pattern 1 – $f_{act}(0*0.2 + 0*0.6) = 0$
desired = 0
weights stay the same

Present pattern 2 – $f_{act}(0*0.2 + 1*0.6) =0$
desired = 1
weight 1 += 0
weight 2 += 1

Present pattern 3 – $f_{act}(1*0.2 + 0*1.6) = 0$
desired = 1
weight 1 += 1
weight 2 += 0

Present pattern 4 – $f_{act}(1*1.2+1*1.6) = 1$
desired = 0
weight 1 -= 1
weight 2 -= 1

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

We end up back at the start!

Modified Versions of Learning

- The weight update function can use a decimal term η between 0.0 and 1.0 to slow learning. Giving us:

If correct $w_i(t+1) = w_i(t)$

If output 0, should be 1: $w_i(t+1) = w_i(t) + \eta x_i(t)$

If output 1, should be 0: $w_i(t+1) = w_i(t) - \eta x_i(t)$

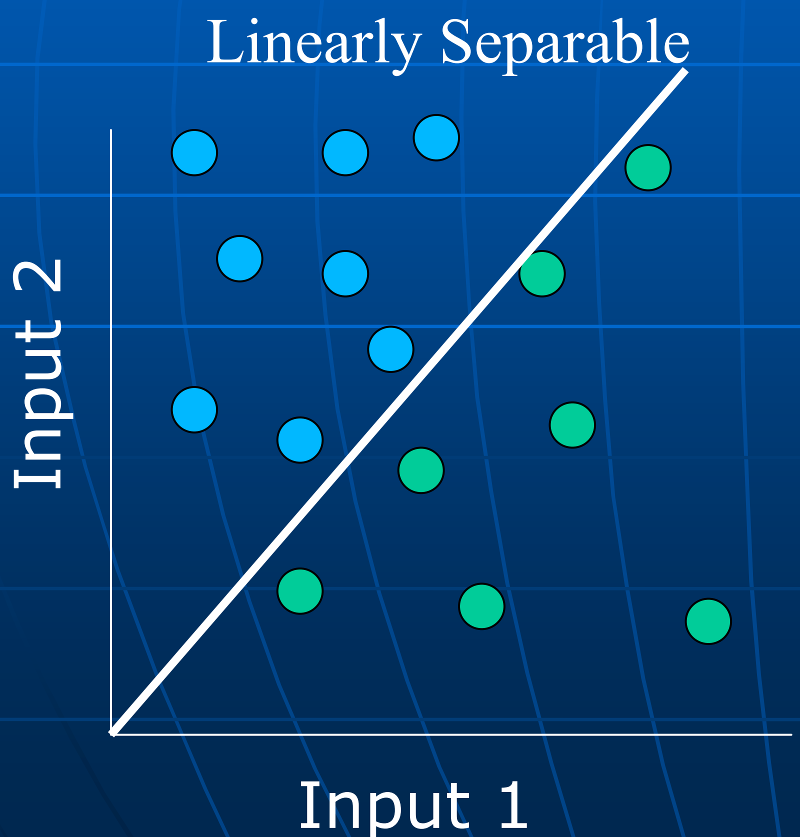
- Widrow-Hoff Learning Rule – weight updates proportionate to the error made. Giving us:

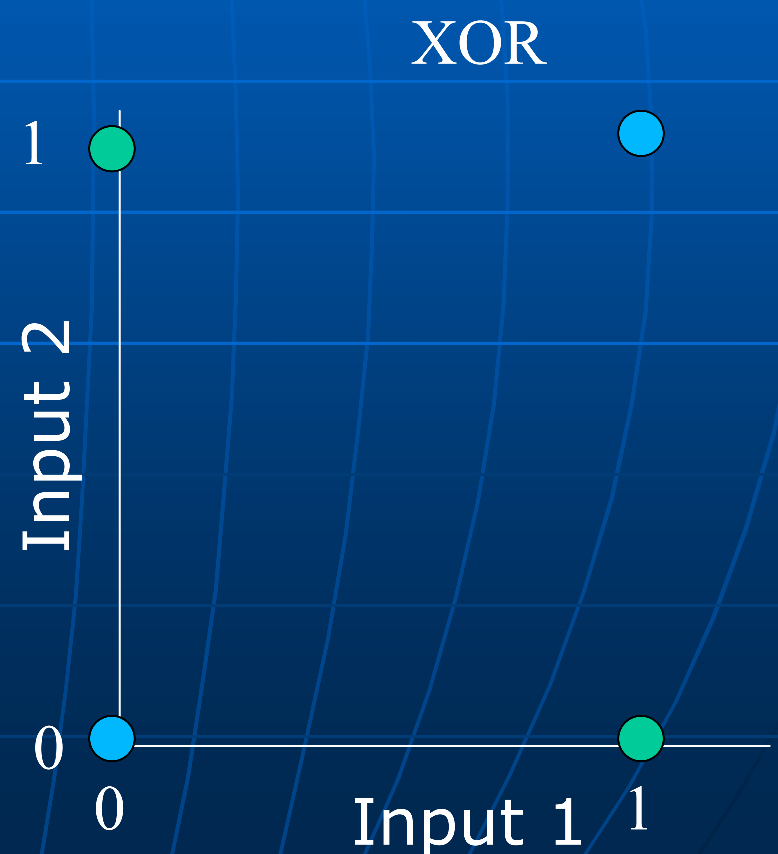
$$\Delta = \text{desired output} - \text{actual output}$$

$$w_i(t+1) = w_i(t) + \eta \Delta x_i(t)$$

Limitations of the Perceptron

- No matter what we do with the learning rule in perceptrons, we can only solve linearly separable problems
- Linearly separable = we can draw a straight line which separates our two classes
- Can we do this for XOR?





Perceptron Demo

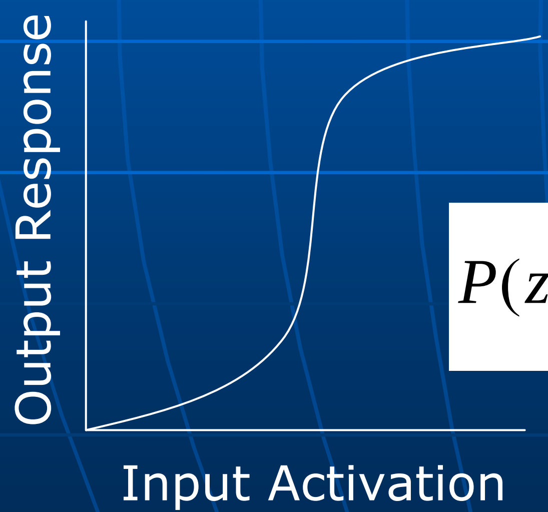
- More Java demos can be found at:
  - neuron.eng.wayne.edu/java

MultiLayer Perceptron

- These limitations can be overcome by adding a further layer to the network
- Three layers
  - Input
  - Hidden
  - Output
- However, we also need a modified algorithm to propagate information through the network and do some learning
- Feedforward, backpropagation neural network

Activation Functions

- Until now – heaviside/threshold function has been used
- In mutlilayer perceptrons a number of different functions can be used, including the Sigmoid function

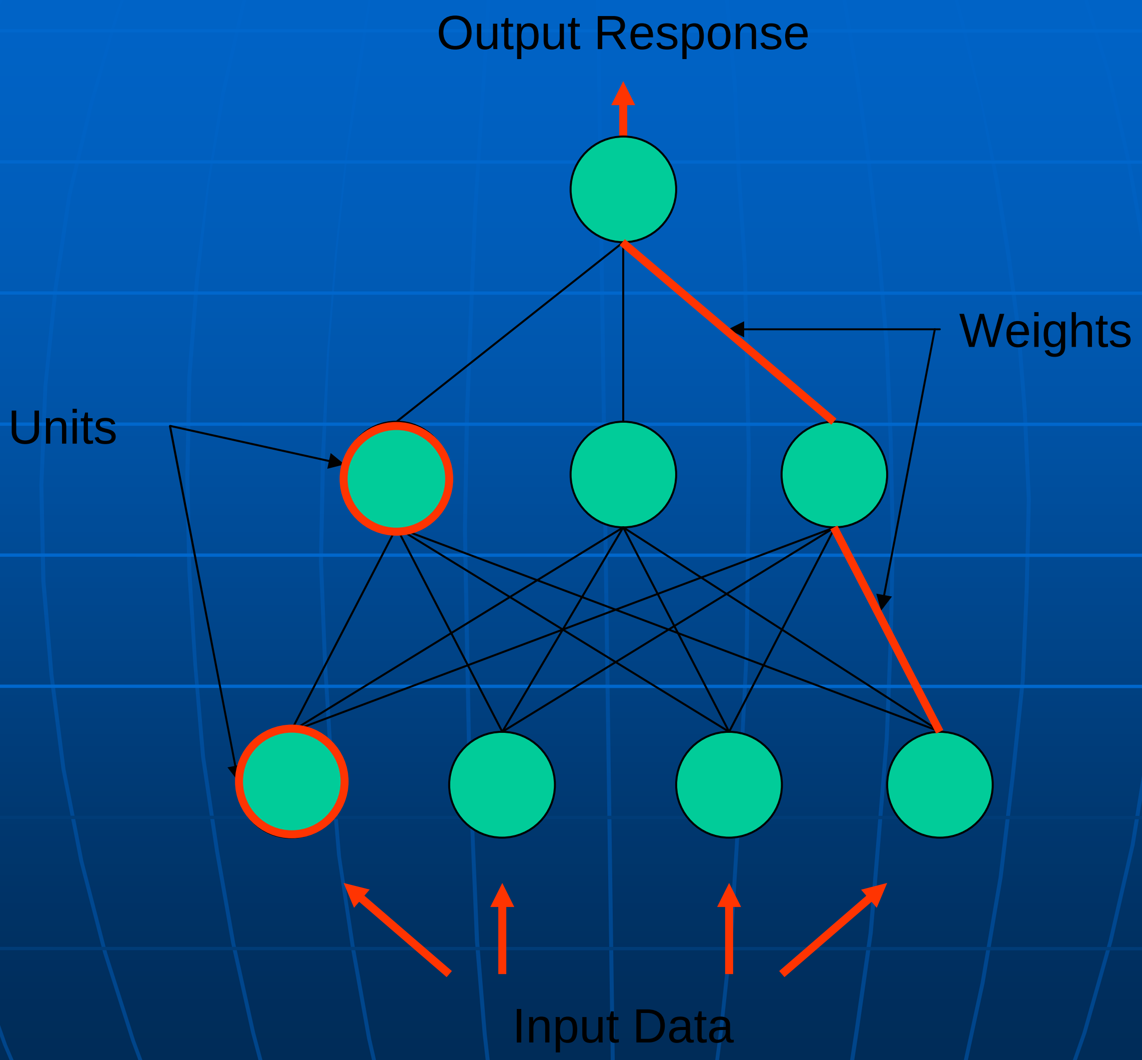


Sigmoid Function

$$P(z) = \frac{1}{1 + e^{-z}}$$

- This gives a smoother response
- The steepness of the curve is changed by z
- The derivative can be easily computed

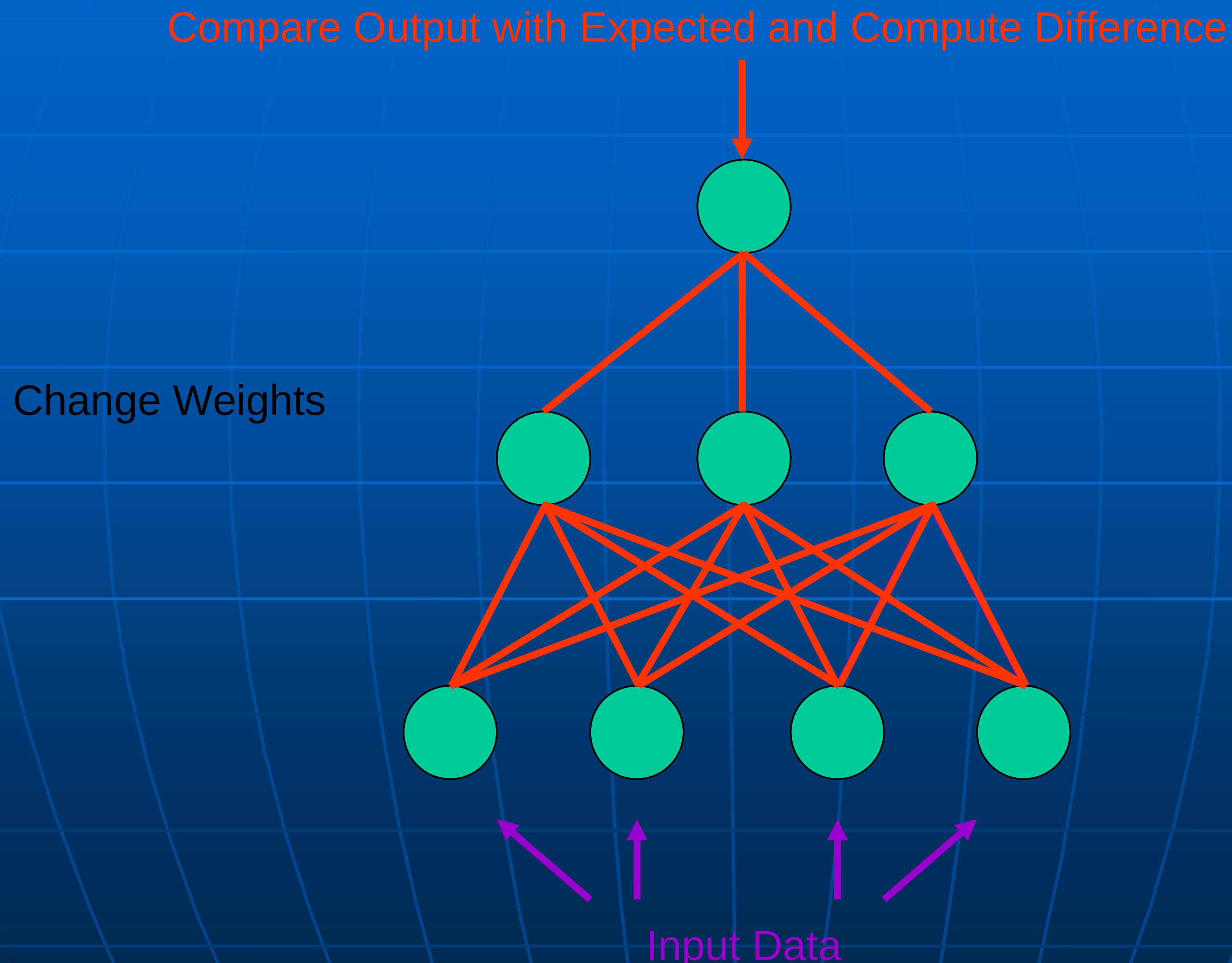
MultiLayer Perceptron



Weights

- Weights are variable strength connections between units
- Propagate signals from one unit to the next
- Main learning component
  - Weights are the main component changed during learning

Supervised Learning



Learning Algorithm - FeedForward

- Initialise weights and thresholds to small random values
- Present Input and Desired Output
- Calculate actual output
  - Multiply incoming signal by weight
  - Pass this through sigmoid activation function
  - Pass on this output to units in the next layer

$$y_{pj} = f\left[\sum_{i=0}^{n-1} w_i x_i\right]$$

Learning Algorithm – Backpropagation 1

- Adapt the weights
- Start from the output layer and work backwards:
  - New weight (t+1) = old weight, plus a learning rate*error for pattern *p* on node *j**output signal for *p* on *j*

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_{pj} o_{pj}$$

Learning Algorithm – Backpropagation 2

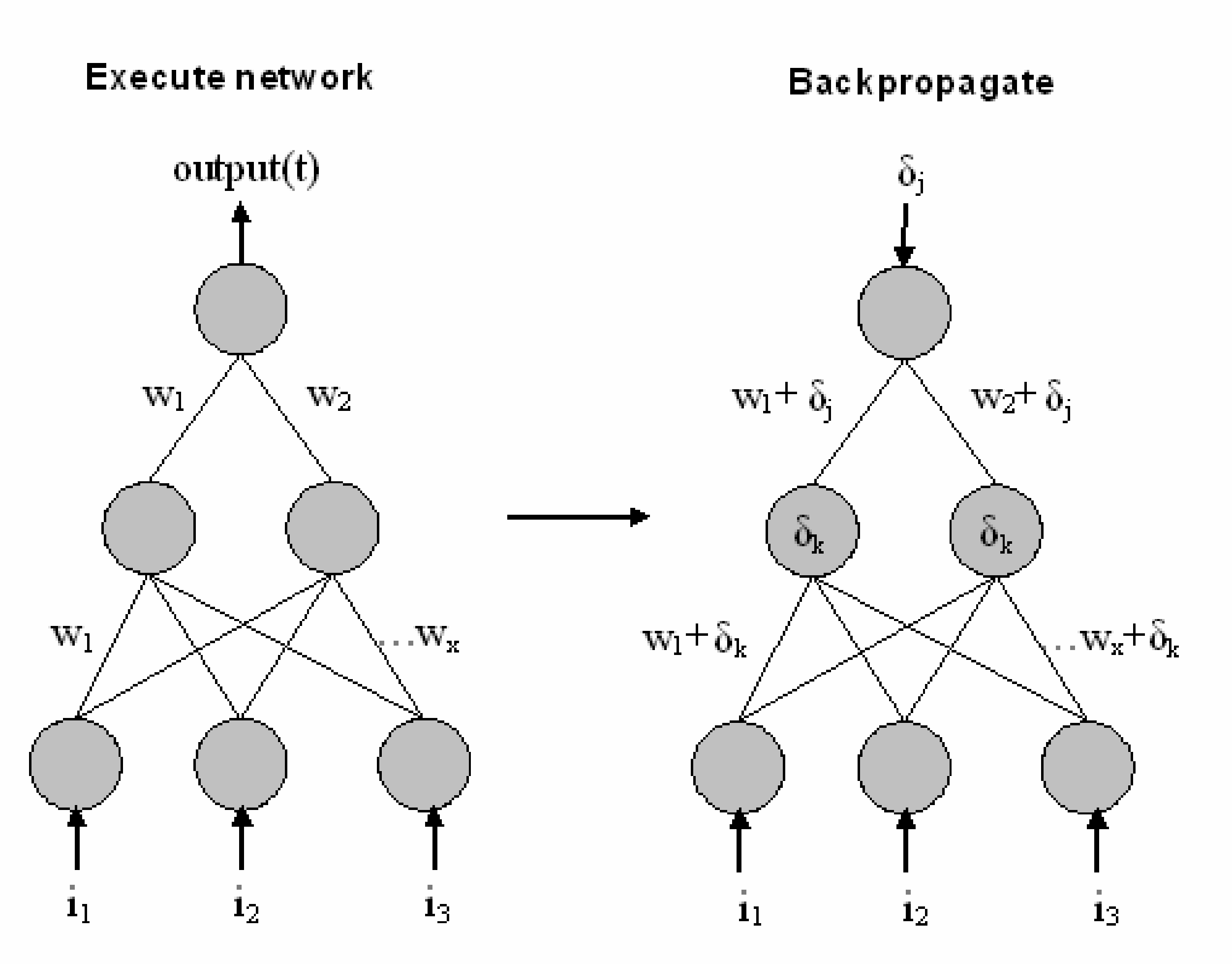
- Compute error as follows:
- For output units
  - Compute error sigmoid derivative*target output – actual output

$$\delta_{pj} = zo_{pj}(1 - o_{pj})(t_{pj} - o_{pj})$$

- For hidden units
  - Use the sigmoid derivative*weighted error of the k units in the layer above

$$\delta_{pj} = zo_{pj}(1 - o_{pj})\sum_{k} \delta_{pk} w_{jk}$$

Learning Illustration

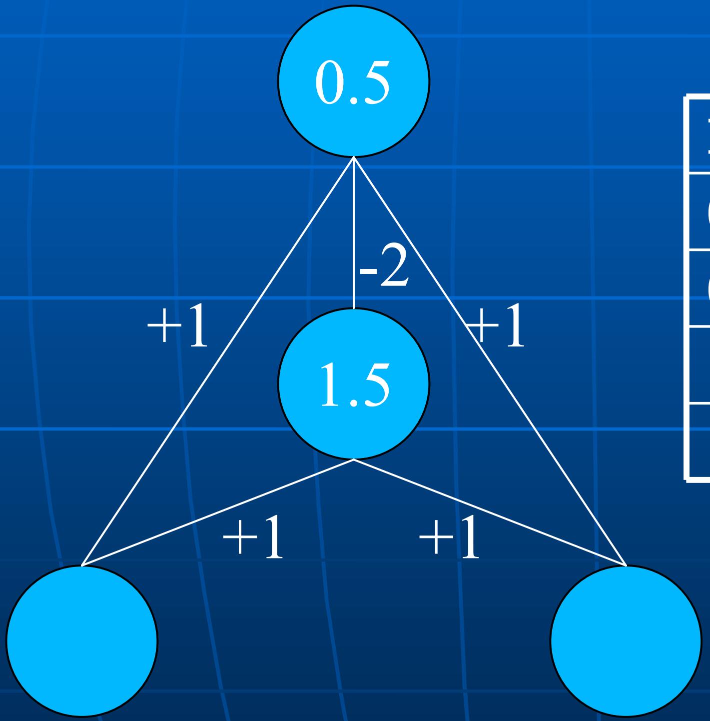


Two Types of Weight Updating

- Batch Updating
  - All patterns are presented, errors are calculated, then the weights are updated

- Online Updating
  - The weights are updated after the presentation of each pattern

XOR Problem – A MultiLayer Solution



Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Data Mining Demo

- Demo

Neural Network Properties

- Able to relate input variables to required output e.g.
  - Input car attributes and predict MPG
  - Predict stock market based on historical information
  - Classify individuals as ‘cancerous’ and ‘non-cancerous’ based on their genes
  - Many other control and learning tasks
- Is able to generalise between samples
- Shows ‘graceful degradation’ – removing one or more units results in reduced performance, not complete failure

Next Time….

- Applications of ANNs
- Issues in running neural networks
  - Input/output representations
  - Choosing architectures
  - Testing
  - Overfitting