Multi-layer Perceptron

Network structure

- Feed Forward Networks

  Represents any arbitrary function, no internal states

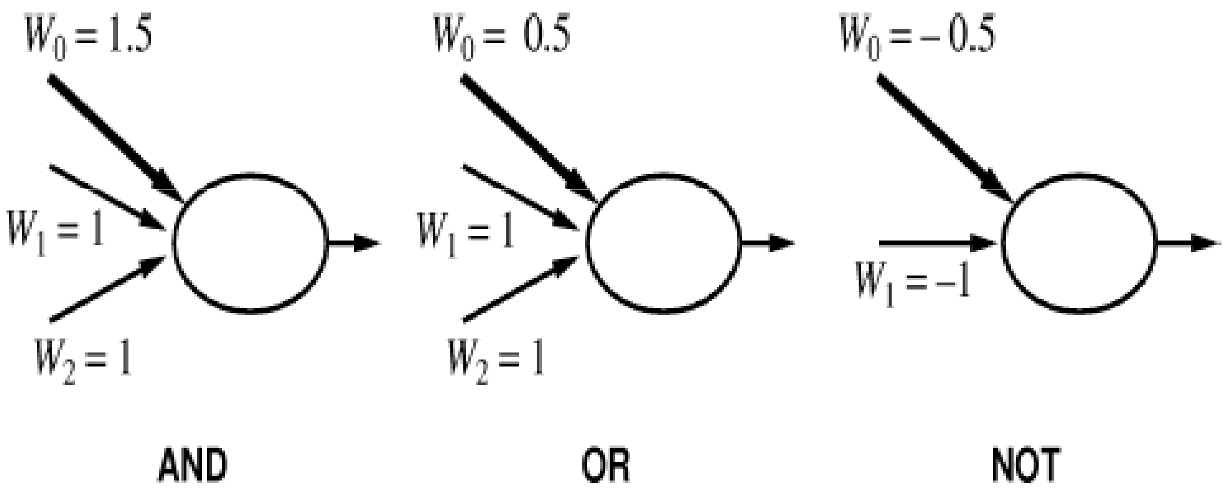
  - Single Layer Perceptron
  - Multi Layer Perceptron

- Recurrent Networks

  Output is fed back with delay, has internal states, can oscillate

  - Hopfield Network
  - Boltzmann Machine

Example – Boolean function



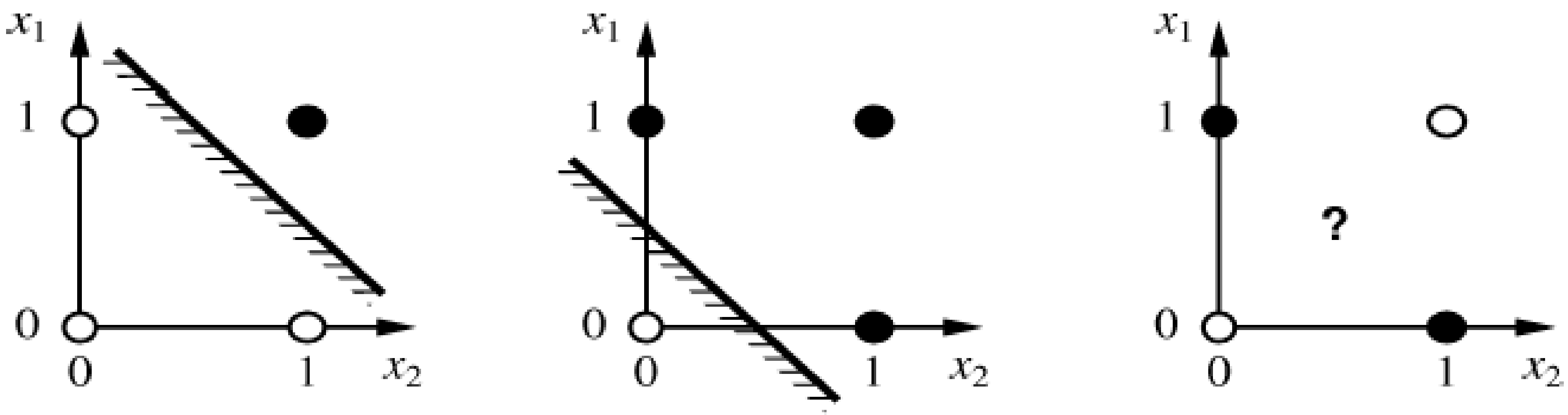
Every boolean function can be represented by a neural network

Contd..

- Thus, single layer network represents a linear separator (line, plane, hyperplane) in the input space

$\sum_j W_j x_j = 0$ or $\mathbf{W} \bullet \mathbf{x} = 0$

- This is not adequate for many pattern recognition problems!



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Contd..

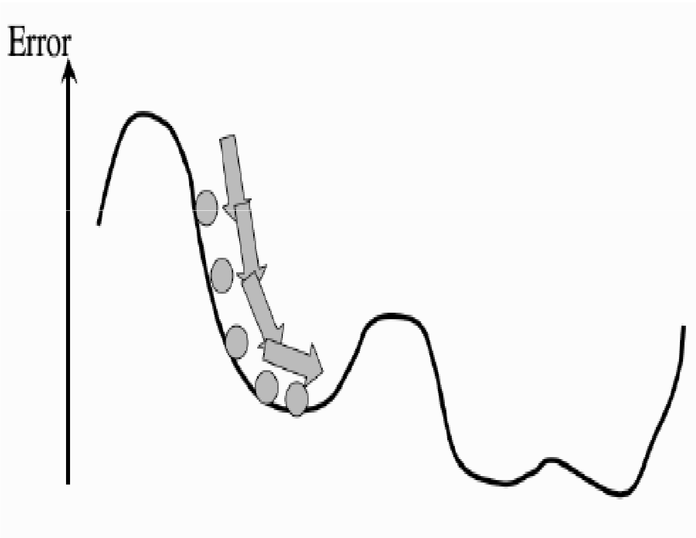
- Any pattern classification problem that can be solved by finding a linear decision boundary is called as linearly separable
- We need an algorithm to learn the weights, and thus a decision boundary, from given examples

Contd..

- Learn by adjusting weights to reduce error on training set
- The squared error for an example with input x and output y is

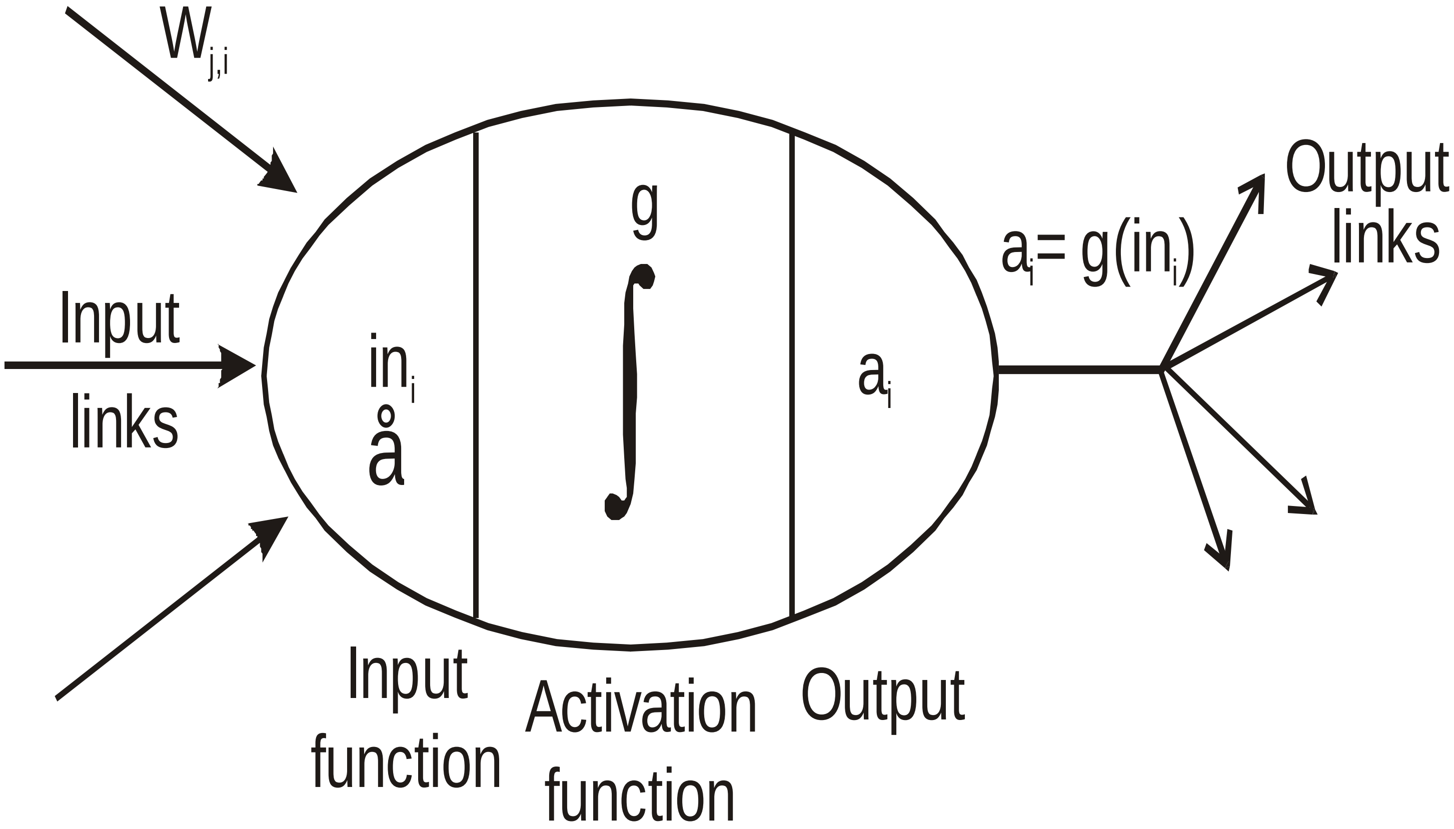
$$E = \frac{1}{2} Err^2 = \frac{1}{2}\left(y - h_W(x)\right)^2$$

Gradient descent



- weights of the network are trained so that the error goes downhill until it reaches a local minimum, just like a ball rolling under gravity.
- back-propagation of error, which makes it clear that the errors are sent backwards through the network.

How to construct a neural network?



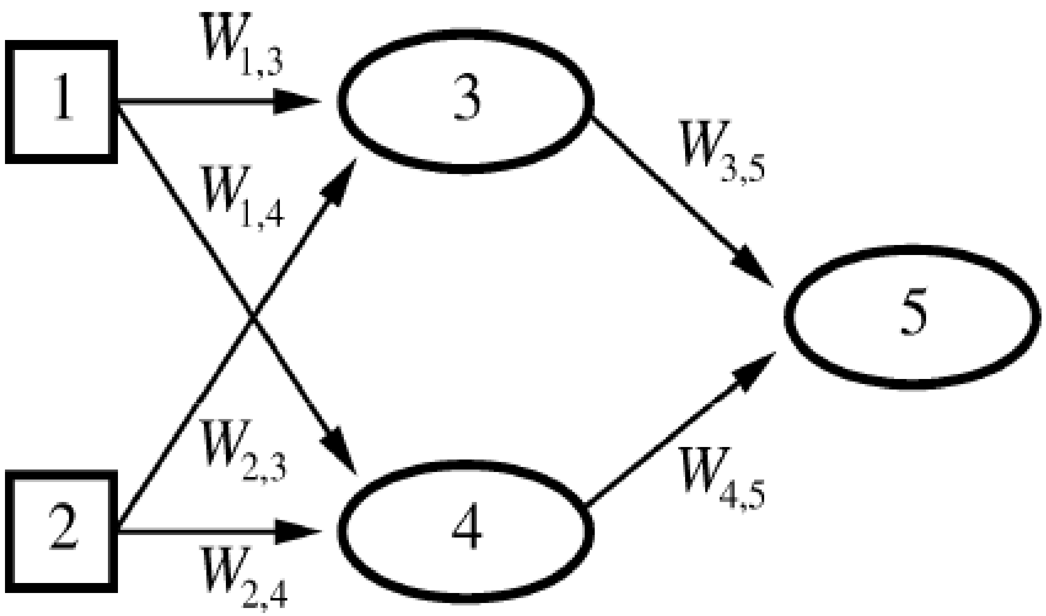
Contd…

- Each unit performs computation using three steps:
- **Input function** is computed by summing the weights and input values - *Linear function.*
- **Activation function** computes the actual output using any one of the activation function from the available three functions (step, sign and sigmoid) - *Nonlinear function*.
- A fixed *threshold value* is introduced in each level instead of having it in each unit.

  ini = åj wj,i

  ai = g(ini)

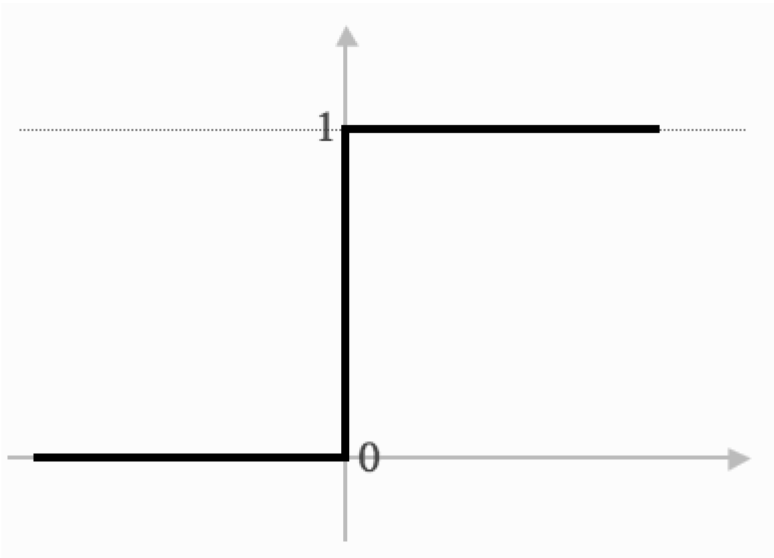
Feed forward network - example



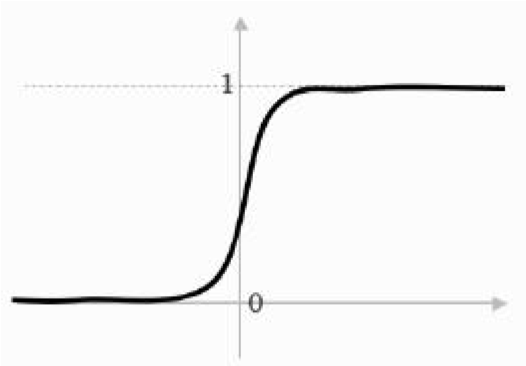
$$
\begin{aligned}
a_5 &= g(\, w_{3,5} * a_3 + w_{4,5} * a_4\,) \\
&= g(\, w_{3,5} * g(w_{1,3} * a_1 + w_{1,4} * a_2) \\
&\qquad + w_{4,5} * g(w_{1,4} * a_1 + w_{2,4} * a_2)\,)
\end{aligned}
$$

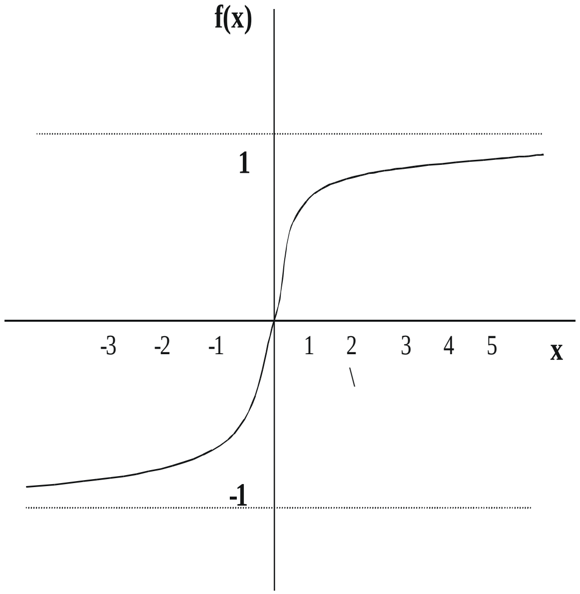
Activation Functions

- Binary step function
- Binary sigmoid function
- Bipolar sigmoid function



Contd…





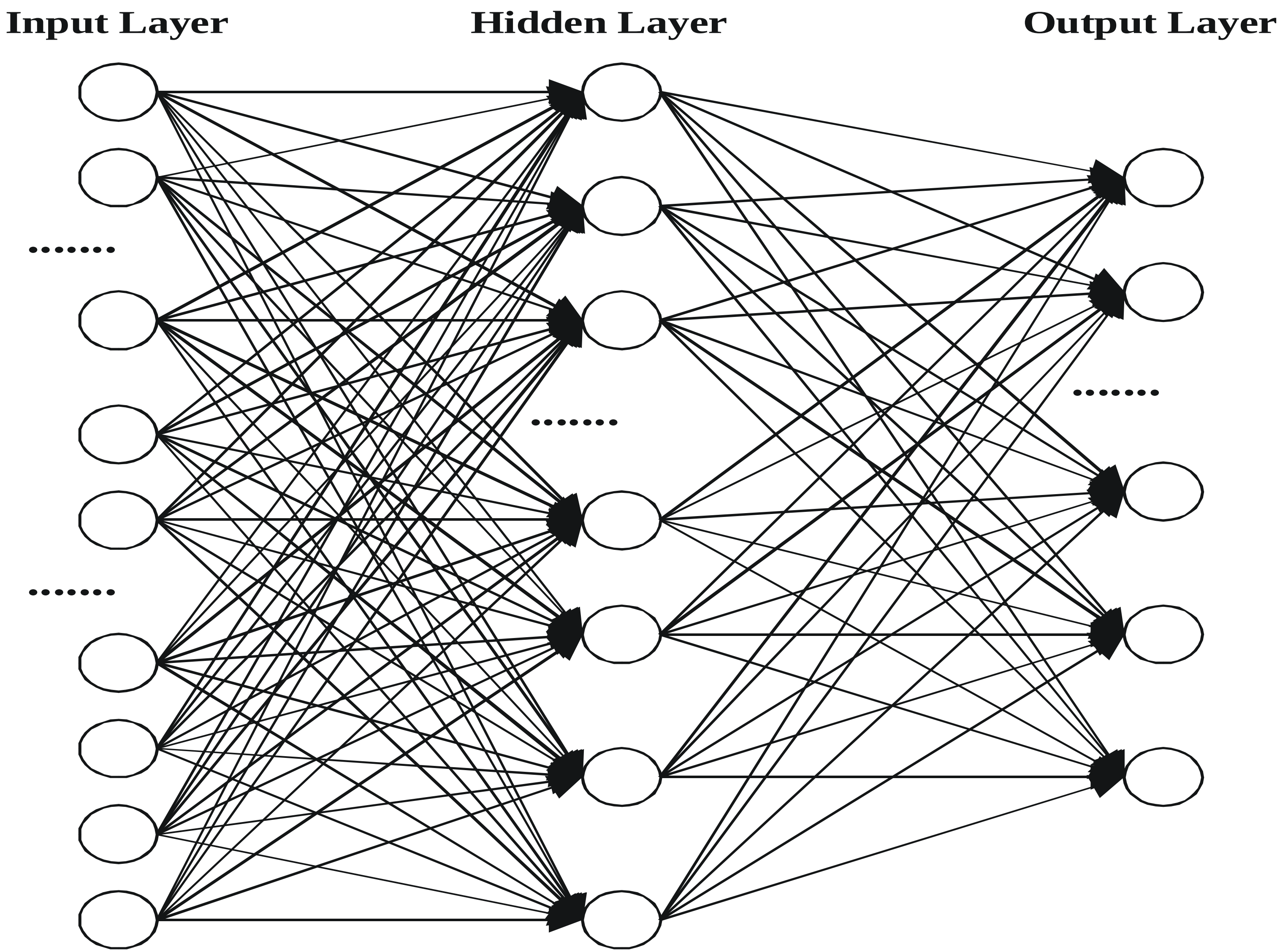
The sigmoid function, which looks qualitatively fairly similar, but varies smoothly and differentiably

Back propagation network

Training a network by back propagation involves 3 stages:

- Feed forward of the input training pattern
- Back propagation of the associated error
- Adjustment of weights

Multi Layer Network



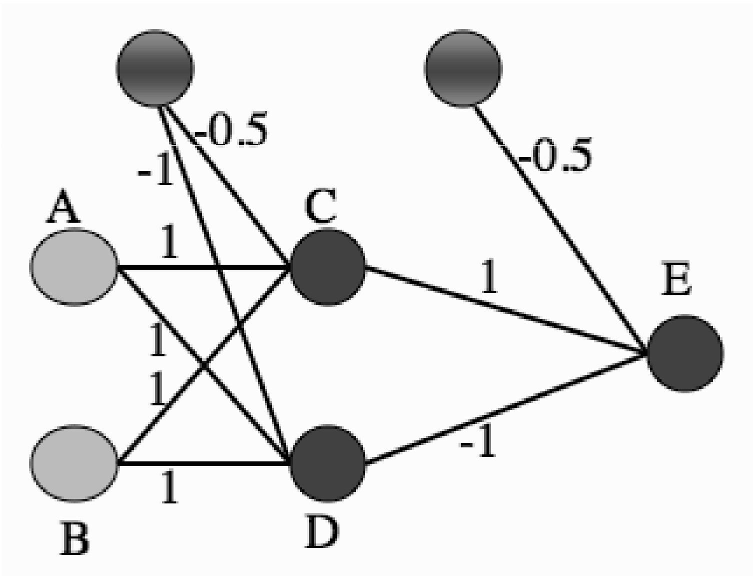
MLP training algorithm using back-propagation

- an input vector is put into the input nodes
- the inputs are fed *forward through the network*
- the inputs and the first-layer weights (here labelled as *v) are used to decide* whether the hidden nodes fire or not. The activation function *g(·) is the sigmoid* function
- the outputs of these neurons and the second-layer weights (labelled as *w) are* used to decide if the output neurons fire or not

Contd…

- the *error is computed as the sum-of-squares difference between the network outputs* and the targets
- this error is fed *backwards through the network in order to* first update the second-layer weights and then afterwards, the first-layer weights

MLP – XOR problem



- Multi-layer Perceptron network showing a set of weights that solve the XOR problem.

Multi-layer Perceptron Algorithm

- Initialisation
  - initialise all weights to small (positive and negative) random values
- Training
  - repeat:
    - for each input vector:

      Forwards phase:

      - compute the activation of each neuron j in the hidden layer(s) using:

$$h_\zeta = \sum_{i=0}^{L} x_i v_{i\zeta} \tag{4.4}$$

$$a_\zeta = g(h_\zeta) = \frac{1}{1+\exp(-\beta h_\zeta)} \tag{4.5}$$

      - work through the network until you get to the output layer neurons, which have activations (although see also Section 4.2.3):

$$h_\kappa = \sum_{j} a_j w_{j\kappa} \tag{4.6}$$

$$y_\kappa = g(h_\kappa) = \frac{1}{1+\exp(-\beta h_\kappa)} \tag{4.7}$$

Contd…

Backwards phase:

- compute the error at the output using:

$$\delta_o(\kappa) = (y_\kappa - t_\kappa)\, y_\kappa (1 - y_\kappa) \tag{4.8}$$

- compute the error in the hidden layer(s) using:

$$\delta_h(\zeta) = a_\zeta (1 - a_\zeta) \sum_{k=1}^{N} w_\zeta \delta_o(k) \tag{4.9}$$

- update the output layer weights using:

$$w_{\zeta\kappa} \leftarrow w_{\zeta\kappa} - \eta \delta_o(\kappa) a_\zeta^{\text{hidden}} \tag{4.10}$$

- update the hidden layer weights using:

$$v_\iota \leftarrow v_\iota - \eta \delta_h(\kappa) x_\iota \tag{4.11}$$

* (if using sequential updating) randomise the order of the input vectors so that you don't train in exactly the same order each iteration

– until learning stops (see Section 4.3.3)

- Recall
  – use the Forwards phase in the training section above