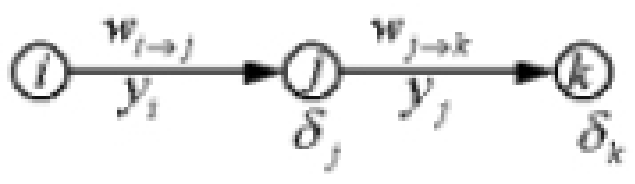
Backpropagation

An efficient method of implementing gradient descent for neural networks

$$w_{i\to j} = w_{i\to j} - r\delta_j y_i$$ **Descent rule**

$$\delta_j = \frac{ds(z_j)}{dz_j}\sum_k \delta_k w_{j\to k}$$ **Backprop rule**

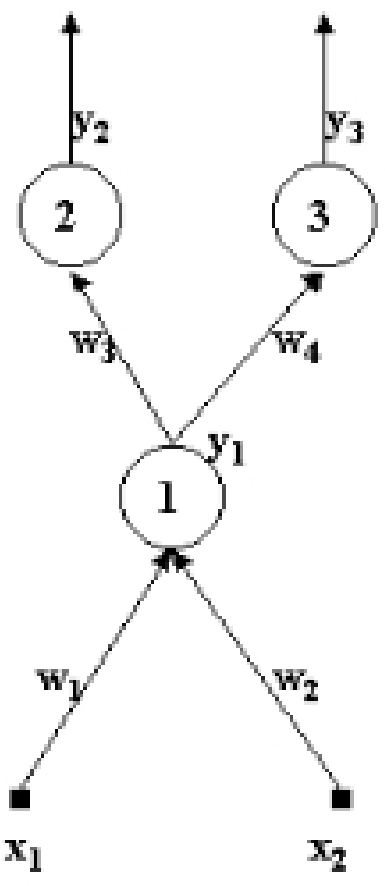


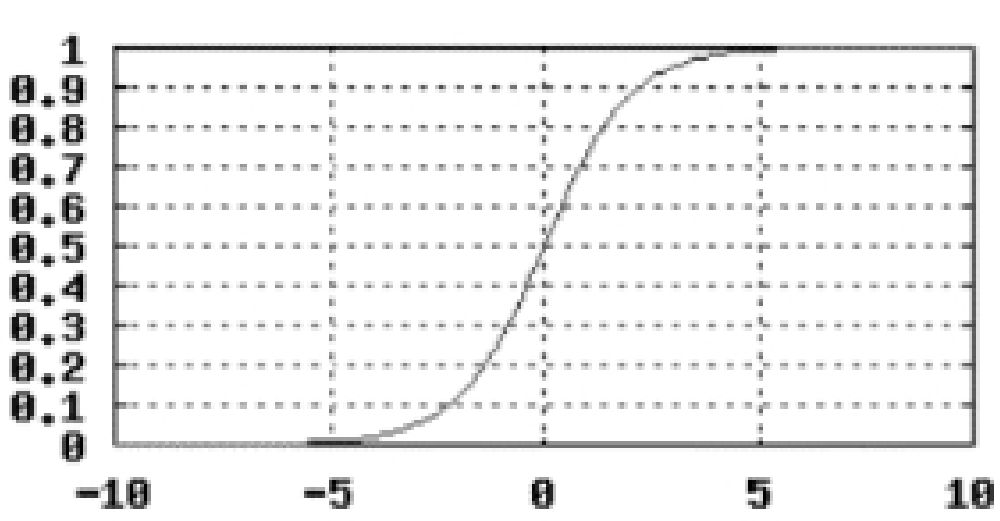
y_i is x_i for input layer

1. Initialize weights to small random values
2. Choose a random sample input feature vector
3. Compute total input (z_j) and output (y_j) for each unit (forward prop)
4. Compute δ_n for output layer $\delta_n = \frac{ds(z_n)}{dz_n}(y_n - y_n^*) = y_n(1-y_n)(y_n - y_n^*)$
5. Compute δ_j for preceding layer by backprop rule (repeat for all layers)
6. Compute weight change by descent rule (repeat for all weights)

Notation in Winston's book $\delta_j = o_j(1-o_j)\beta_j, y_j = o_j, y_n^* = d_n$

9p - Nov 00 - 15





$x_1 = x_2 = 2$
$w_1 = 2$, $w_2 = -2$, $w_3 = 4$, and $w_4 = 0$.

B.2 (8 points)

You are confronted with two situations, and you are to determine whether the weights will go up or down. In both situations, you are to use the network of part B.1, which uses sigmoid neurons.

In **situation 1**, you are to assume that the desired outputs for inputs

$x_1 = 2$,
$x_2 = 2$

are

$y_2 = 0$,
$y_3 = 1$,

that the learning rate is 1, and that the weights are as in B.1.

In **situation 2**, you are to assume that the desired outputs, for the same inputs, are

$y_2 = 1$,
$y_3 = 0$,

Fill in the cells of the table below with **up** and **down,** as appropriate.

	Situation 1	**Situation 2**
w_1		
w_2		
w_3		
w_4		

We provide a following page with helpful information that you can tear off and use for reference.