Given k^{th} training data pair:
$\{ (x_1(k_1), x_2(k_2), \ldots)^T, t(k) \}$

$t_i(k) - o_i^{(L)} \sim \text{error}$

Objective function of online training:

$$E(k) = \frac{1}{2}\sum_{i=1}^{q}\left(t_i(k) - o_i^{(L)}(k)\right)^2 \quad ; \quad k = 1, 2, \ldots, n$$

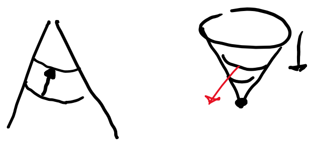
$n =$ the total number of training data pairs

Objective function of offline training:

$$E_c = \sum_{k=1}^{n} E(k) = \sum_{k=1}^{n}\sum_{i=1}^{q}\frac{1}{2}\left(t_i(k) - o_i^{(L)}\right)^2$$

$\vec{W}^{(L)}(k+1) = \vec{W}^{(L)}(k) + \Delta\vec{W}^{(L)}(k)$

$\Delta\vec{W}^{(L)}(k) = -\eta \frac{\partial E(k)}{\partial \vec{W}^{(L)}(k)}$



$\Delta\vec{W}_{ij}^{(\ell)} = -\eta \frac{\partial E}{\partial o_i^{(\ell)}} \cdot \frac{\partial o_i^{(\ell)}}{\partial tot_i^{(\ell)}}$

$tot_i^{(\ell)} = w_{i1}^{(\ell)} o_1^{(\ell-1)} + \cdots + w_{ij}^{(\ell)} o_j^{(\ell-1)} + \cdots + w_{i(m_1)}^{(\ell)} o_{m_1}^{(\ell-1)}$

(See previous neural network node map)

Output layer L:

$$E(k) = \frac{1}{2}\sum_{i=1}^{q}\left(t_i^{(L)} - o_i^{(L)}\right)^2$$

$E(k) = \left(t_1^{(L)} - o_1^{(L)}\right) + \cdots + \left(t_i^{(L)} - o_i^{(L)}\right) + \cdots + (t_q^{(L)} - o_q^{(L)})$

$tot_i^{(L)} = \cdots + w_{ij}^{(L)} o_j^{(L-1)} + \cdots$

$\Delta W_{ij}^{(L)} = -\eta \frac{\partial E}{\partial o_i^{(L)}} \cdot \frac{\partial o_i^{(L)}}{\partial tot_i^{(L)}} \cdot \frac{\partial tot_i^{(L)}}{\partial W_{ij}^{(L)}}$

$$\Delta W_{ij}^{(L)} = -\eta \frac{1}{2} * (2)\left(t_1^{(L)} - o_1^{(L)}\right)(-1) * f'\left(tot_i^{(L)}\right) * o_j^{(L-1)}$$

$$o_i^{(L-1)} = f\left(tot_i^{(L)}\right)$$

- If a sigmoid AF (activation function) is used:

$$f(x) = \frac{1}{1+e^{-x}}$$

$x = tot_i^{(L)}$
$f(x) = o_i^{(L)}$

$$f = \frac{1}{1+e^{-x}}$$
$$f' = [(1+e^{-x})^{-1}]' = -1(1+e^{-x})^{-2}e^{-x}(-1)$$
$$= \frac{e^{-x}}{(1+e^{-x})^2}$$
$$= \overset{f}{\frac{1}{1+e^{-x}}} * \left(1 - \overset{f}{\frac{1}{1+e^{-x}}}\right)$$

Then,

$$f' = f(1-f)$$

- If $o_i^{(L)}$ is a sigmoid function:

$$\Delta W_{ij}^{(L)} = \eta\left(t_i^{(L)} - o_i^{(L)}\right) f(1-f) o_j^{(L-1)}$$
$$= \eta\left(t_i^{(L)} - o_i^{(L)}\right) o_i^{(L)}\left(1 - o_i^{(L)}\right) o_j^{(L-1)}$$

$\delta_i^{(L)} = \left(t_i^{(L)} - o_i^{(L)}\right) f'\left(tot_i^{(L)}\right)$
$\delta_i^{(L)} = \left(t_i^{(L)} - o_i^{(L)}\right) o_i^{(L)}\left(1 - o_i^{(L)}\right)$

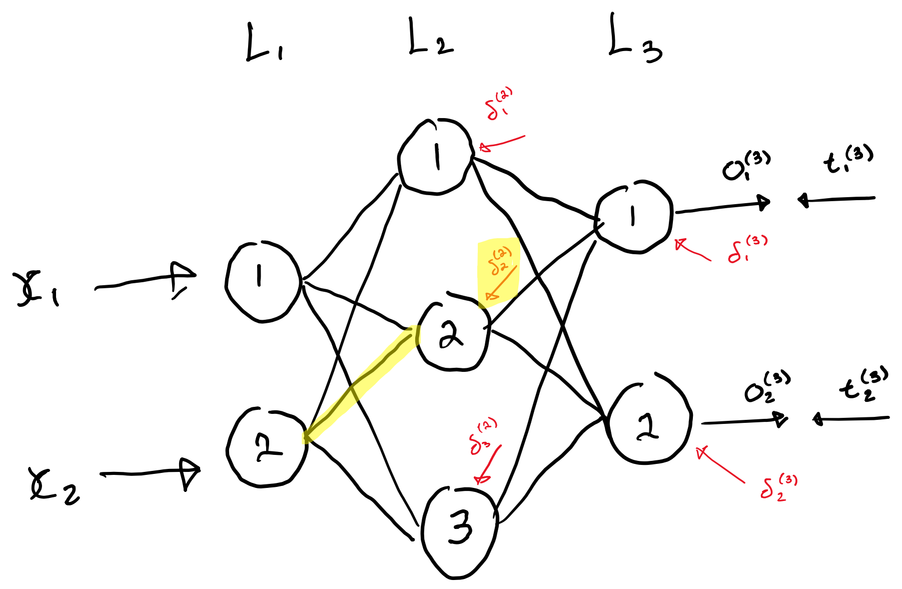
$W_{ij}^{(L)} = \eta \delta_i^{(L)} * o_j^{(L-1)}$

For $W_{ij}^{(\ell)}$:

$$\boxed{\Delta W_{ij}^{(\ell)} = \eta \delta_i^{(\ell)} o_j^{(\ell-1)}}$$

$$\left(t_i^{(\ell)} - o_i^{(\ell)}\right) f'\left(tot_i^{(\ell)}\right)$$

General structure:



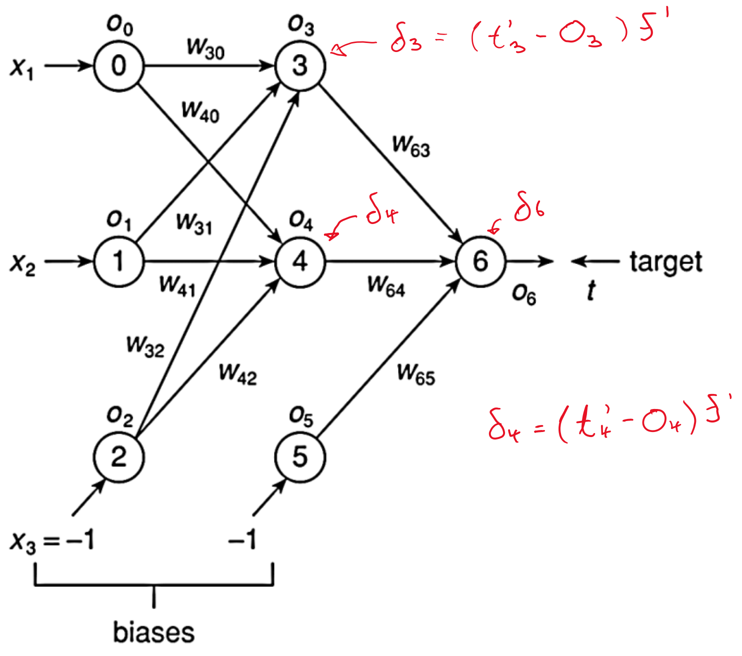
Forward pass → o (calculate output)

Backward pass → update $W_{ij}^{(L)}$

Example 5.1

To illustrate this powerful algorithm, we apply it for the training of the following network, shown in Figure 5.4. The following htree training pattern pairs are used, with $\boldsymbol{x}$ and $\boldsymbol{t}$ being the input and the output data respectively:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \quad \mathbf{t}^{(1)} = (0.88)$$
$$\mathbf{x}^{(2)} = (0.1, 0.6), \quad \mathbf{t}^{(2)} = (0.82)$$
$$\mathbf{x}^{(3)} = (0.9, 0.4), \quad \mathbf{t}^{(3)} = (0.57)$$



$$O = f(\Sigma wo - \underline{\underline{\theta}})$$

Biases are treated here as connection weights that are always multiplied by (-1) through a neuron to avoid special case calculation for biases. Each neuron uses a unipolar sigmoid activation function given by:

$$o = f(tot) = \frac{1}{1 + e^{-\lambda tot}}, \text{using } \lambda = 1, \text{then } f'(tot) = o(1 - o)$$

Solution

Example 5.1

To illustrate this powerful algorithm, we apply it for the training of the following network shown in Figure 5.4. The following three training pattern pairs are used, with **x** and **t** being the input and the output data respectively:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \quad \mathbf{t}^{(1)} = (0.88),$$
$$\mathbf{x}^{(2)} = (0.1, 0.6), \quad \mathbf{t}^{(2)} = (0.82),$$
$$\mathbf{x}^{(3)} = (0.9, 0.4), \quad \mathbf{t}^{(3)} = (0.57),$$

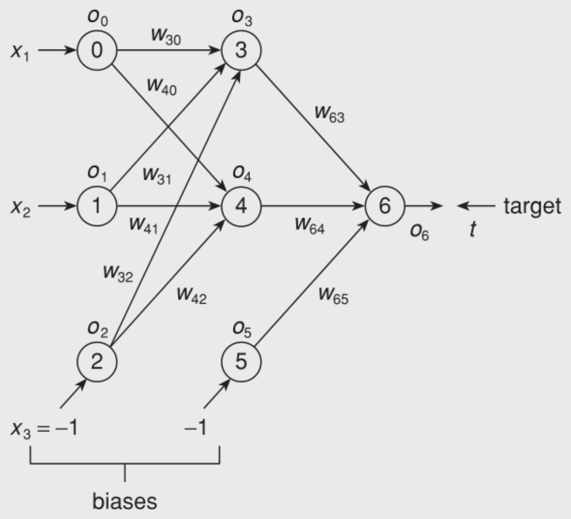


Figure 5.4: Structure of the neural network of Example 5.1

Biases are treated here as connection weights that are always multiplied by -1 through a neuron to avoid special case calculation for biases. Each neuron uses a unipolar sigmoid activation function given by:

$$o = f(\text{tot}) = \frac{1}{1 + e^{-\lambda \text{tot}}}, \text{ using } \lambda = 1, \text{ then } f'(\text{tot}) = o(1 - o)$$

Step (1) – Initialization

- Initialize the weights to small random values. We assume all weights are initialized to 0.2; set learning rate to $\eta = 0.2$; set maximum tolerable error to $E_{\max} = 0.01$ (i.e., 1% error); set current error value to $E = 0$; set current training pattern to $k = 1$.

Training Loop – Loop (1)

Step (2) – Apply input pattern

- Apply the 1st input pattern to the input layer:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \mathbf{t}^{(1)} = (0.88), \text{ then, } o_0 = \mathbf{x}_1 = 0.3;\ o_1 = \mathbf{x}_2 = 0.4;\ o_2 = \mathbf{x}_3 = -1$$

Step (3) – Forward propagation

- Propagate the signal forward through the network:

$$o_3 = f(w_{30}o_0 + w_{31}o_1 + w_{32}o_2) = 0.4850$$
$$o_4 = f(w_{40}o_0 + w_{41}o_1 + w_{42}o_2) = 0.4850$$
$$o_5 = -1$$
$$o_6 = f(w_{63}o_3 + w_{64}o_4 + w_{65}o_5) = 0.4985$$

Step (4) – Output error measure

- Compute the error value E and the error signal δ_6 of the output layer:

$$E = \tfrac{1}{2}(t - o_6)^2 + E = 0.0728$$
$$\begin{aligned}\delta_6 &= f'(\text{tot}_6)(t - o_6)\\ &= o_6(1 - o_6)(t - o_6)\\ &= 0.0954\end{aligned}$$

Step (5) – Error backpropagation

- Propagate the errors backward to update the weights and compute the error signals of the preceding layers.

Third layer weight updates:

$$\Delta w_{63} = \eta\delta_6 o_3 = 0.0093 \qquad w_{63}^{\text{new}} = w_{63}^{\text{old}} + \Delta w_{63} = 0.2093$$
$$\Delta w_{64} = \eta\delta_6 o_4 = 0.0093 \qquad w_{64}^{\text{new}} = w_{64}^{\text{old}} + \Delta w_{64} = 0.2093$$
$$\Delta w_{65} = \eta\delta_6 o_5 = -0.0191 \qquad w_{65}^{\text{new}} = w_{65}^{\text{old}} + \Delta w_{65} = 0.1809$$

First ~~Second~~ layer error signals:

$$\delta_3 = f_3'(\text{tot}_3)\sum_{i=6}^{6} w_{i3}\delta_i = o_3(1-o_3)w_{63}\delta_6 = 0.0048$$
$$\delta_4 = f_4'(\text{tot}_4)\sum_{i=6}^{6} w_{i4}\delta_i = o_4(1-o_4)w_{64}\delta_6 = 0.0048$$

Second layer weight updates:

$$\Delta w_{30} = \eta\delta_3 o_0 = 0.00028586 \qquad w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2003$$
$$\Delta w_{31} = \eta\delta_3 o_1 = 0.00038115 \qquad w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2004$$
$$\Delta w_{32} = \eta\delta_3 o_2 = -0.00095288 \qquad w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.1990$$

$$\Delta w_{40} = \eta\delta_4 o_0 = 0.00028586 \qquad w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2003$$
$$\Delta w_{41} = \eta\delta_4 o_1 = 0.00038115 \qquad w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2004$$
$$\Delta w_{42} = \eta\delta_4 o_2 = -0.00095288 \qquad w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.1990$$

Training Loop – Loop (2)

Step (2) – Apply the 2nd input pattern

Apply the 2nd input pattern to the input layer:

$\mathbf{x}^{(2)} = (0.1, 0.6)$, $\mathbf{t}^{(2)} = (0.82)$, then, $o_0 = 0.1$, $o_1 = 0.6$, $o_2 = -1$

Step (3) – Forward propagation

$$o_3 = f(w_{30}o_0 + w_{31}o_1 + w_{32}o_2) = 0.4853$$
$$o_4 = f(w_{40}o_0 + w_{41}o_1 + w_{42}o_2) = 0.4853$$
$$o_5 = -1$$
$$o_6 = f(w_{63}o_3 + w_{64}o_4 + w_{65}o_5) = 0.5055$$

Step (4) – Output error measure

$$E = \tfrac{1}{2}(t - o_6)^2 + E = 0.1222$$
$$\delta_6 = o_6(1 - o_6)(t - o_6) = 0.0786$$

Step (5) – Error backpropagation

Third layer weight updates:

$$\Delta w_{63} = \eta\delta_6 o_3 = 0.0076 \qquad w_{63}^{\text{new}} = w_{63}^{\text{old}} + \Delta w_{63} = 0.2169$$
$$\Delta w_{64} = \eta\delta_6 o_4 = 0.0076 \qquad w_{64}^{\text{new}} = w_{64}^{\text{old}} + \Delta w_{64} = 0.2169$$
$$\Delta w_{65} = \eta\delta_6 o_5 = -0.0157 \qquad w_{65}^{\text{new}} = w_{65}^{\text{old}} + \Delta w_{65} = 0.1652$$

Second layer error signals:

$$\delta_3 = f_3'(\text{tot}_3)\sum_{i=6}^{6} w_{i3}\delta_i = o_3(1-o_3)w_{63}\delta_6 = 0.0041$$
$$\delta_4 = f_4'(\text{tot}_4)\sum_{i=6}^{6} w_{i4}\delta_i = o_4(1-o_4)w_{64}\delta_6 = 0.0041$$

~~Second~~ First layer weight updates:

$$\Delta w_{30} = \eta\delta_3 o_0 = 0.000082169 \qquad w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2004$$
$$\Delta w_{31} = \eta\delta_3 o_1 = 0.00049302 \qquad w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2009$$
$$\Delta w_{32} = \eta\delta_3 o_2 = -0.00082169 \qquad w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.1982$$

$$\Delta w_{40} = \eta\delta_4 o_0 = 0.000082169 \qquad w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2004$$
$$\Delta w_{41} = \eta\delta_4 o_1 = 0.00049302 \qquad w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2009$$
$$\Delta w_{42} = \eta\delta_4 o_2 = -0.00082169 \qquad w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.1982$$

Training Loop – Loop (3)

Step (2) – Apply the 3rd input pattern to the input layer

$\mathbf{x}^{(2)} = (0.9,\ 0.4)$, $\mathbf{t}^{(2)} = (0.57)$, then, $o_0 = 0.9$, $o_1 = 0.4$, $o_2 = -1$

Step (3) – Forward propagation

$$o_3 = f(w_{30}o_0 + w_{31}o_1 + w_{32}o_2) = 0.5156$$
$$o_4 = f(w_{40}o_0 + w_{41}o_1 + w_{42}o_2) = 0.5156$$
$$o_5 = -1$$
$$o_6 = f(w_{63}o_3 + w_{64}o_4 + w_{65}o_5) = 0.5146$$

Step (4) – Output error measure

$$E = \tfrac{1}{2}(t - o_6)^2 + E = 0.1237$$
$$\delta_6 = o_6(1 - o_6)(t - o_6) = 0.0138$$

Required Steps for Backpropagation Learning Algorithm

- **Step 1:** Initialize weights and thresholds to small random values.
- **Step 2:** Choose an input-output pattern form the training input-output data set:

$$(x(k), t(k))$$

- **Step 3:** Propagate the k^{th} signal forward through the network and compute the output values or all i neurons at every layer (ℓ) using:

$$o_i^{\ell}(k) = f\left(\sum_{p=0}^{n_{\ell-1}} w_{ip}^{(\ell)} o_p^{(\ell-1)}\right)$$

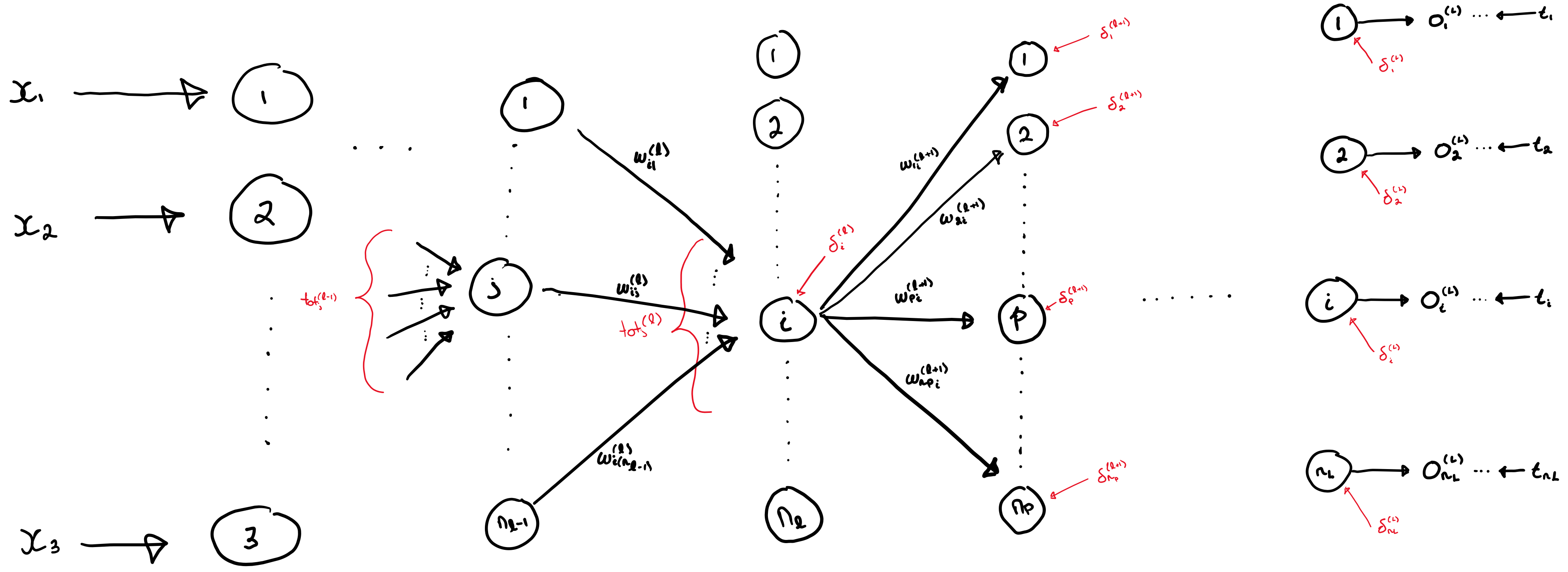
- **Step 4:** Compute the total error value $E = E(k) + E$ and the error signal $\delta_i^{(L)}$ using formulae:

$$\delta_i^{(L)} = \left[t_i - o_i^{(L)}\right]\left[(tot)_i^{(L)}\right]$$

- **Step 5:** Update the weights according to:

$$\Delta w_{ij}^{(\ell)} = -\eta \delta_i^{(\ell)} o_j^{(\ell-1)}, \quad \text{for } \ell = \mathrm{L}, \dots, 1 \quad \text{using}$$
$$\delta_i^{(L)} = \left[t_i - o_i^{(L)}\right]\left[f'(tot)_i^{(L)}\right] \quad \text{and proceeding backward using}$$
$$\delta_i^{(\ell)} = o_i^{(\ell)}\left(1 - o_i^{(\ell)}\right)\sum_{p=1}^{n_\ell} \delta_p^{(\ell+1)} w_{pi}^{(\ell+1)} \quad \text{for } \ell < L$$

- **Step 6:** Repeat the process starting from step 2 using another exemplar. Once all exemplars have been used, we then reach what is known as one epoch training.
- **Step 7:** Check is the cumulative error E in the output layer has become less than a predetermined value. If so, we say the network has been trained. If not, repeat the whole process for one more epoch.



x_1
x_2
x_3
1
2
3
1
j
n_{l-1}
$tot_j^{(l-1)}$
$tot_j^{(l)}$
$w_{i1}^{(l)}$
$w_{ij}^{(l)}$
$w_{i(n_{l-1})}^{(l)}$
1
2
i
n_l
$\delta_i^{(l)}$
$w_{1i}^{(l+1)}$
$w_{2i}^{(l+1)}$
$w_{pi}^{(l+1)}$
$w_{n_p i}^{(l+1)}$
1
2
p
n_p
$\delta_1^{(l+1)}$
$\delta_2^{(l+1)}$
$\delta_p^{(l+1)}$
$\delta_{n_p}^{(l+1)}$
1
2
i
n_L
$O_1^{(L)} \cdots t_1$
$O_2^{(L)} \cdots t_2$
$O_i^{(L)} \cdots t_i$
$O_{n_L}^{(L)} \cdots t_{n_L}$
$\delta_1^{(L)}$
$\delta_2^{(L)}$
$\delta_i^{(L)}$
$\delta_{n_L}^{(L)}$

$$\Delta w_{ij}^{(\ell)} = \eta \delta_i^{(\ell)} o_i^{(\ell-1)}$$

Error signal:

$$\delta_i^{(\ell)} = f'\left(tot_i^{(\ell)}\right)\left[\delta_1^{(\ell+1)} w_{1i}^{(\ell+1)} + \delta_2^{(\ell+1)} w_{2i}^{(\ell+1)} + \cdots + \delta_p^{(\ell+1)} w_{pi}^{(\ell+1)} + \cdots + \delta_{n_p}^{(\ell+1)} w_{n_p i}^{(\ell+1)}\right]$$

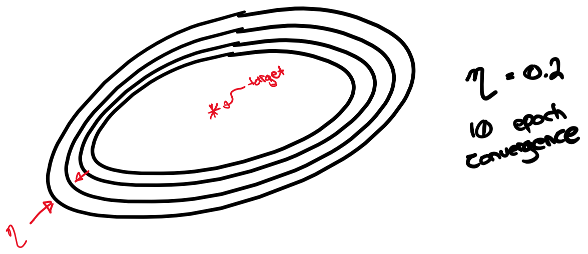
$$\delta_i^{(\ell)} = f'\left(tot_i^{(\ell)}\right)\sum_{p=1}^{n_p} \delta_p^{(\ell+1)} w_{p_i}^{(\ell+1)}$$

For a sigmoid AF, there is a special case:

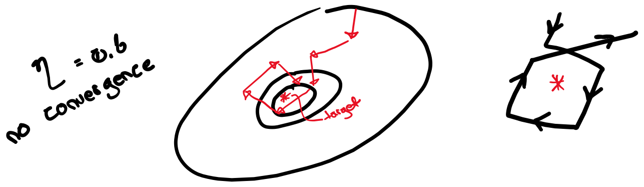
$$f' = f(1-f) \rightarrow o^{(\ell)}(1-o^{(\ell)})$$

4.6 Momentum

When η is small, the convergence towards the target is slow:



Conversely, when η is large, it can miss the target (convergence not met)



$E_{min} = 0.05$

$$\Delta\vec{w}^{(\ell)}(k+1) = -\eta \frac{\partial E(k)}{\partial \vec{w}^{(\ell)}} \underline{\nu \Delta \vec{w}_{(k)}^{(\ell)}} \xrightarrow{} \text{momentum}$$

$\Delta\vec{w}^{(\ell)}$
$\nu \in [0,1]$
$\nu = 0.8, 0.9$

Example 5.2

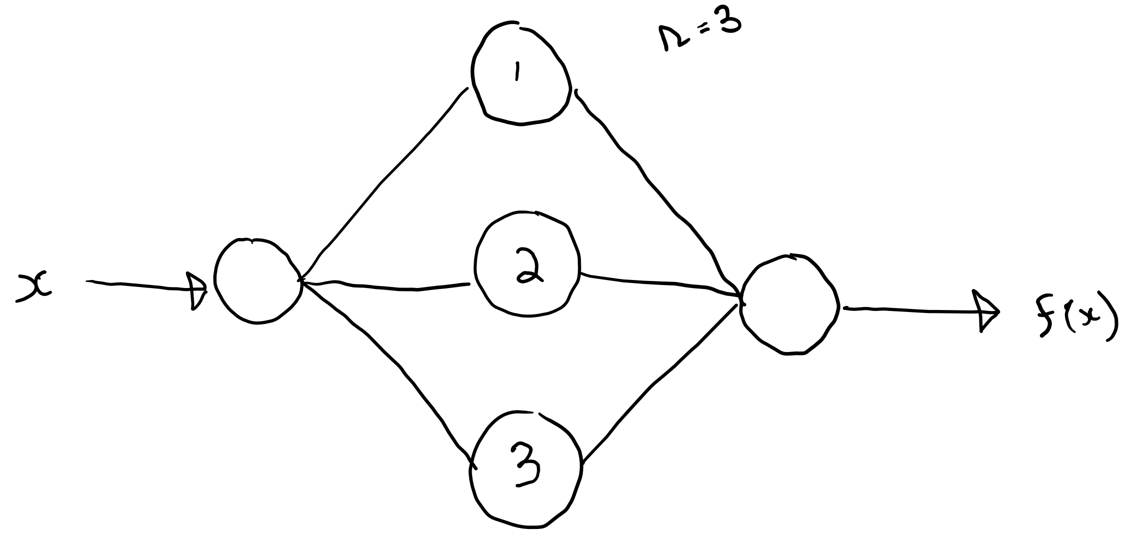
Effect of hidden nodes on function approximation

To illustrate the effects of the number of hidden neurons on the approximation capabilities of the MLP, we use here the simple function $f(x)$ given by:

$$f(x) = x\sin(x)$$

Six input/output samples were selected from the range [0:10] of the variable x. The first run was made for a network with three hidden nodes. The results are shown in Figure 5.6(a). Another run was made for a network with five (Figure 5.6(b)) and 20 (Figure 5.6(c)) nodes respectively. From the result of the simulation, one may conclude that a higher number of nodes is not always better as is seen in Figure 5.6(c). This is mostly due to the fact that a network with this structure has overinterpolated in between the samples and we say that the network was overtrained. This happens when the network starts to memorize the patterns instead of interpolating between them. In this series of simulations the best match with the original curve was obtained with a network having five hidden nodes. It seems here that this network (with five nodes) was able to interpolate quite well the nonlinear behavior of the curve. A smaller number of nodes didn't permit a faithful approximation of the function given that the nonlinearities induced by the network were not enough to allow for adequate interpolation between samples.

$$f(x) = x\sin(x)$$
$$x = 0\sim 10$$



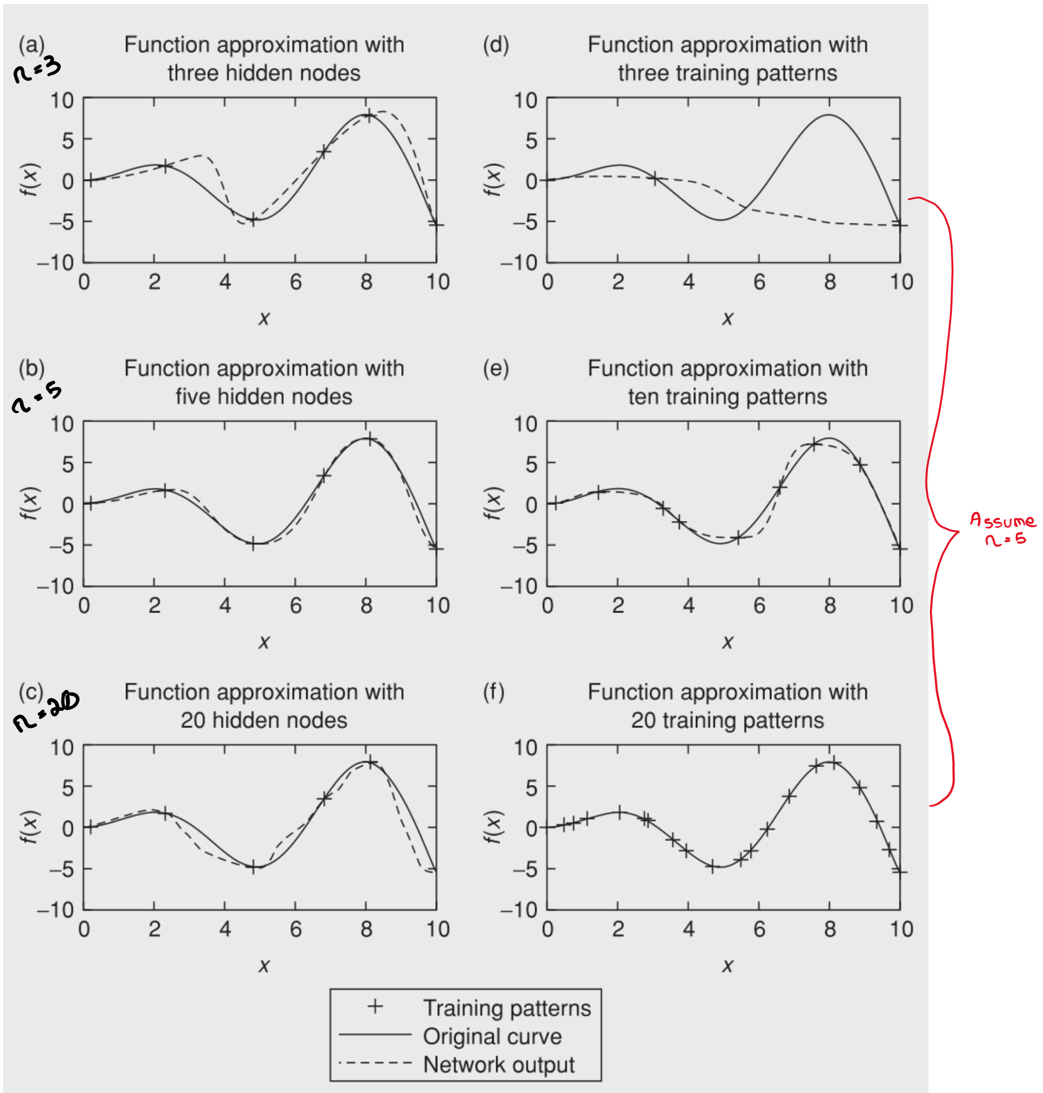


Figure 5.6 a, b, c: Effect of the number of hidden nodes on MLP approximation of $f(x)$

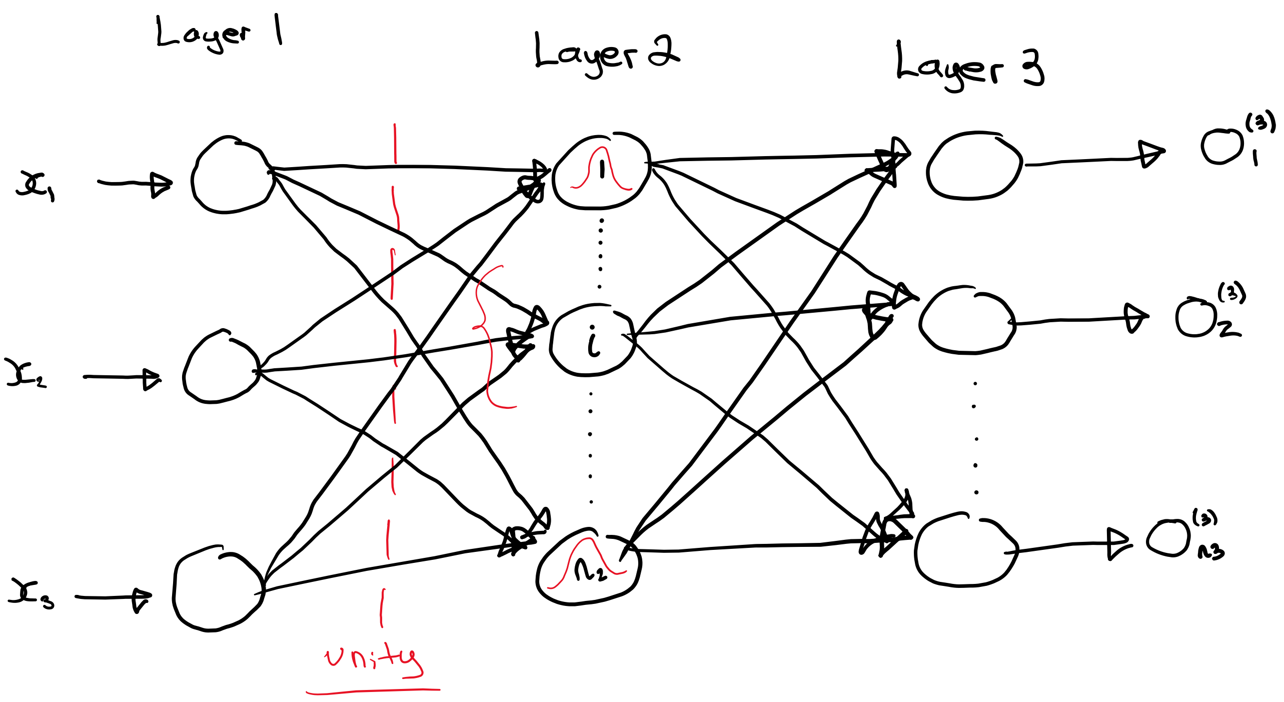
Figure 5.6 d, e, f: Effect of the number of training patterns on MLP approximation of $f(x)$

Using more neurons in the hidden layer doesn't necessarily improve the performance of the system, but using more training data pairs improves system performance.

4.7 Radial Basis Function Neural Network (RBF NN)

- Special case of a feedforward neural network

1. 3 Layer FF NN



2. Unity line weights between (neurons) layer 1 and layer 2 (they have the same value).

3. AFs in the neurons in hidden layer are kernel functions.

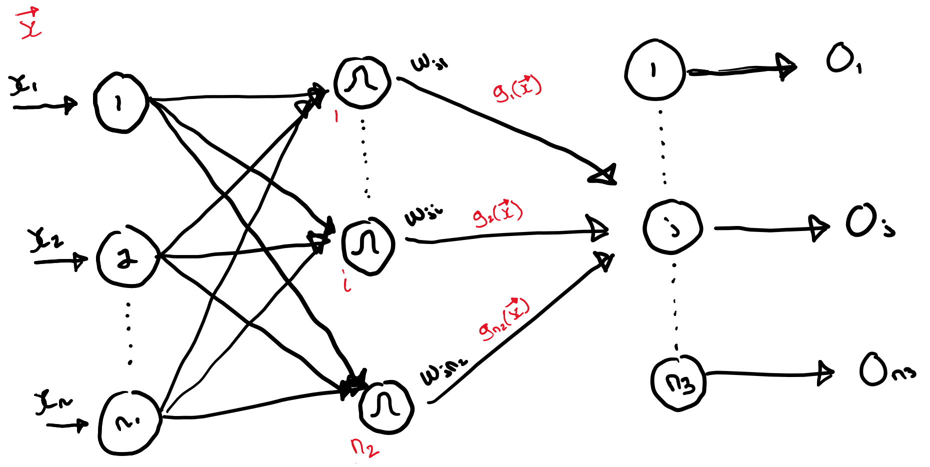
- Gaussian function:

$$g_i(\vec{x}) = e^{\frac{-||\vec{x}-\vec{v}_i||^2}{2\sigma_i^2}}$$

$\vec{x}$ = input vector
$\vec{v}_i$ = center vector
σ_i = spread parameter

- Logic function:

$$g_i(\vec{x}) = \frac{1}{1 + e^{\frac{-||\vec{x}-\vec{v}_i||^2}{2\sigma_i^2}}}$$



Output:

$$o_j(\vec{x}) = g_1(\vec{x})w_{j1} + \cdots + g_i(\vec{x})w_{ji} + \cdots + g_{n_2}(\vec{x})w_{jn_2} \quad ; \quad j = 1, 2, \ldots, n_3$$

$$o_j(\vec{x}) = \sum_{i=1}^{n_2} w_{ji} * g_i(\vec{x})$$

Training:

- Parameters in the hidden neuron AFs (centers and spreads)
- Link weights between the hidden layer & output layer

Note:
A Radial Basis Function (RBF) neural network is a neuro-fuzzy system

Chapter 5: Neuro-Fuzzy Systems

5.1 Introduction

	Fuzzy logic	**Neural networks**
Representation	Linguistic description of knowledge	Knowledge distributed within computational units
Adaption	Some adaptation	Adaptive
Knowledge Representation	Explicit and easy to interpret	Implicit and difficult to interpret
Learning	Non-existent	Excellent tools for imparting learning
Verification	Easy and efficient	Not straightforward (“black box” reasoning)

Integrated systems of fuzzy logic (FL) and neural networks (NN)

1. Neuro-fuzzy (NF) system
   FL parameters can be trained by using NN training methods (back propagation, etc.)
2. Fuzzy-neuro system (RBF)
   Neural network, but some neurons are fuzzified
3. Neural fuzzy systems
   Just a simple combination of FL and NN (separate systems utilized in series)