Multi-layer Feedforward Neural Network
(Back Propagation Algorithm)

- Error corrects the weights

$$Net = w_i^T x \ , \quad f(net) = \frac{1}{1+e^{-x}}$$

$$\frac{\partial ou}{\partial net} = f'(net) = out(1 - out)$$

- Training pattern $(\overline{x_p}, \overline{t_p})$

$$\overline{x_p} = \begin{bmatrix} x_{p1} \\ \vdots \\ x_{pn} \end{bmatrix} ; \ \overline{t_p} = \begin{bmatrix} t_{p1} \\ \vdots \\ t_{pn} \end{bmatrix}$$

- For node in Hidden layer j, the net input is

$$Net_j = \sum_i w_{ij} o_i$$

Where, o_i= Output of node i

w_{ij} = Connection weight from neuron i to neuron j

- The output of node j is

$$O_j = f(net_j) \qquad (1)$$

Where f is the activation function

- For sigmoidal activation function

$$O_j = \frac{1}{1+e^{-net_j}} \ ; \ O_j = \frac{1}{1+e^{\left(\frac{-net_j - \theta_j}{\theta_o}\right)}} \ ; \ O_j = \frac{2}{1+e^{-net_j}} - 1$$

This is for Bipolar sigmoidal function

- The output of node k is

$$O_k = f(net_k) \qquad (2)$$

Where f is the activation function

- For sigmoidal activation function

$$O_k = \frac{1}{1+e^{-net_k}} \ ; \ O_k = \frac{1}{1+e^{\left(\frac{-net_k - \theta_k}{\theta_o}\right)}} \ ; \ O_k = \frac{2}{1+e^{-net_k}} - 1$$

This is for Bipolar sigmoidal function

- For nodes in output layer k,
  the net input is

$$Net_k = \sum_j w_{jk} o_j \qquad (3)$$

Where, o_j = Output of node j

w_{jk} = Connection weight from neuron j to neuron k

- Compare the final output signals with a target signals then adjust weights between unit so that the difference between the final and target signals is minimized. The weights are changed to minimize the following cost function

$$E_p = \frac{1}{2} \sum_j (t_{pj} - O_{pj})^2 \qquad (4)$$

Where, t_{pj} = target signal at unit j for input pattern p

O_{pj} = output signal of unit j

- Error may be +ve or -ve
- Weights between neurons are determined as follows:
  The **Steepest Decent Method** is utilized in order to minimize Eq. (4)
- Back Propagation Algorithm, attempts to minimize error by adjusting each weight of the network proportional to the derivative of the error with respect to that weight.
- The change in weights from unit (i) to (j) for input pattern ' p ' is given by

$$\Delta_p w_{ij} \propto -\left(\frac{\partial E_p}{\partial w_{ij}}\right)$$

- The change in proportional to gradient error

$$w^{(k+1)} = w^{(k)} + \Delta w$$

$$\Delta w = \eta (t - o)\, x$$ ← input

[Widrow-Hoff Rule]

η → Learning Rate

(modifies the rate of convergence)

Error +ve → ADD
Error -ve → SUBTRACT

- The change in weights from unit (k) to (j)

$$\Delta w_{kj} \propto -\left(\frac{\partial E}{\partial w_{kj}}\right)$$

(-ve indicates minimization of error)

$$\Delta w_{kj} = -\eta\left(\frac{\partial E}{\partial w_{kj}}\right) \qquad (5)$$

From Eq.(5)

$$-\left(\frac{\partial E}{\partial w_{kj}}\right) = -\left(\frac{\partial E}{\partial w_{kj}}\right).\left(\frac{\partial net_k}{\partial net_k}\right)$$

$$= -\left(\frac{\partial E}{\partial net_k}\right).\left(\frac{\partial net_k}{\partial w_{kj}}\right) \qquad (6)$$

$$E_p = \frac{1}{2}\sum_k (t_{pk} - O_{pk})^2$$

system error

$$E = \frac{1}{2}\sum_{p=1}^{p}\left[\sum_k (t_{pk} - O_{pk})^2\right]$$

$$\frac{\partial E}{\partial O_k} = -(t_k - O_k) \qquad (7)$$

From Eq. (3)

$$net_k = w_k^t O_j$$

$$\frac{\partial net_k}{\partial w_{kj}} = O_j \qquad (8)$$

From Eq. (6)

$$\delta_k = -\left(\frac{\partial E}{\partial net_k}\right) \qquad (9)$$

$$= -\left(\frac{\partial E}{\partial net_k}\right).\left(\frac{\partial O_k}{\partial O_k}\right)$$

$$= -\left(\frac{\partial E}{\partial O_k}\right).\left(\frac{\partial O_k}{\partial net_k}\right)$$

From Eq. (6) and derived Eq. (2)

$$\delta_k = (t_k - O_k).f'(net_k) \qquad (10)$$

Put Eq. (8) and (9) in Eq. (6)

$$= [(t_k - O_k).f'(net_k)].O_j$$

- Delta rule, it is valid for continuous activation function

Put Eq. (6) in Eq. (5) $\Delta w_{kj} = \eta[(t_k - O_k).f'(net_k)].O_j$

$$\Delta w_{kj} = \eta.\delta_k.O_j$$

Internal Layer

$$\Delta w_{ji} = -\eta\left(\frac{\partial E}{\partial w_{ji}}\right) \qquad (11)$$

$$= -\eta\left(\frac{\partial E}{\partial w_{ji}}\right).\left(\frac{\partial net_j}{\partial net_j}\right)$$

$$= -\eta\left(\frac{\partial E}{\partial net_j}\right).\left(\frac{\partial net_j}{\partial w_{ji}}\right)$$

Similarly, as Eq. (8)

$$= -\eta\left(\frac{\partial E}{\partial net_j}\right).O_i$$

$$= -\eta\left(\frac{\partial E}{\partial net_j}\right).\left(\frac{\partial O_j}{\partial O_j}\right).O_i$$

$$= -\eta\left(\frac{\partial E}{\partial O_j}\right).\left(\frac{\partial O_j}{\partial net_j}\right).O_i$$

Similarly, as Eq. (10)

$$= -\eta\left(\frac{\partial E}{\partial O_j}\right).f'(net_j).O_i \qquad (12)$$

From Eq. (12)

$$\delta_j = -\left(\frac{\partial E}{\partial O_j}\right).f'(net_j) \qquad (13)$$

From Eq. (13)

$$-\left(\frac{\partial E}{\partial O_j}\right) = -\sum_k \left(\frac{\partial E}{\partial O_j}\right).\left(\frac{\partial net_k}{\partial net_k}\right)$$

$$= -\sum_k \left(\frac{\partial E}{\partial net_k}\right).\left(\frac{\partial net_k}{\partial O_j}\right)$$

Where, $net_k = w_{kj}o_j$

$$= \sum_k \left(-\frac{\partial E}{\partial net_k}\right).\left(\frac{\partial \sum w_{kj}o_j}{\partial O_j}\right)$$

$$= \sum_k \left(-\frac{\partial E}{\partial net_k}\right).w_{kj}$$

From Eq. (9)

$$= \sum_k (-(-\delta_k)).w_{kj} \qquad (14)$$

- Direction δ_k has been reversed in this case, i.e., $\delta_k \rightarrow -\delta_k$

Put Eq. (14) in Eq. (13)

$$\delta_j = \sum_k \delta_k \,.\, w_{kj} \,.\, f'(net_j) \qquad (15)$$

From Eq. (12), Eq. (13) and Eq. (15)

$$\Delta w_{ji} = \eta \left[\sum_k \delta_k \,.\, w_{kj} \,.\, f'(net_j) \right] . O_i$$

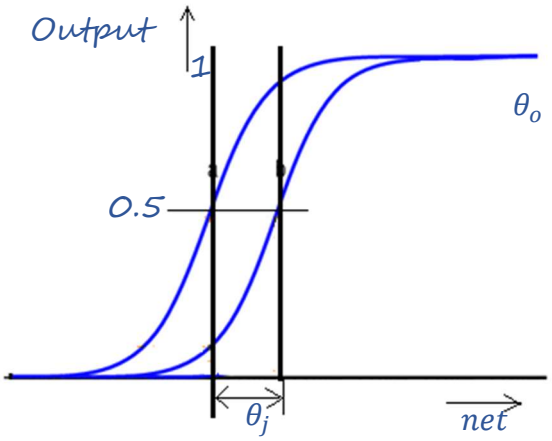
$$\Delta w_{ji} = \eta \left[\sum_k \delta_k \,.\, w_{kj} \,.\, O_j(1 - O_j) \right] . O_i$$

- BPA → Propagating the output layer through the networks layer by layer, adjusting weight at each layer
- However, for hidden layers δ must be generalised without benefit of a target vector.
- First δ for each neuron in the output layer. It is used to adjust the weight feeding into the output layer, then it is propagated back through the same weights to generate a value. For δ, for each neuron in the first hidden layer. There values of δ are used, in turn to adjust the weight of this hidden layer and in a similar way, are propagate δ back to all the preceding layers.
- Consider a single neuron in the hidden layer just before the output layer. In the forward pass, this neuron propagates in output value to neurons in the output layer through the inter connecting weights.
- During training these weights operate in reverse, passing the value of δ from the output layer back to the hidden layer. Each of these weights multiplied by the δ value of the neuron to which it connects in the output layer.
- The value of δ needed for the hidden layer neuron is produced by summing all such products and multiplying by the derivative of the squashing function.

$$\delta_{pj} = out_{p,j}(1 - out_{p,j}) \left(\sum_q \delta_{q,k} \,.\, w_{pq,k} \right)$$

Comments or Observations

1. The sigmoidal function never becomes 1 or 0 unless function goes $+\alpha$ to $-\alpha$. Therefore, the weights should be $0 < w < 1$
2. Stubbing values chosen randomly
3. If η is very large, it is oscillatory
4. Add a momentum term (α) such that it will not exchange too fast
   Large α → Convergence is slow
   $$\Delta w_{ji}(n+1) = \eta\, \delta_j O_i + \alpha\, \Delta w_{ji}(n)$$
5. Local minima
6. θ_j could be viewed as equivalent weight



θ_j → Threshold

θ_0 → Slope

7. $$f'(net_j) = \frac{\partial O_j}{\partial net_j} = O_j(1 - O_j)$$