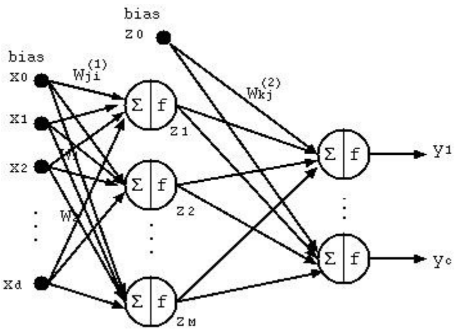
BP Training Algorithm I

1

Multilayer Perceptron



Such a two-layer perceptron has universal approximation ability.

SSE Cost function:

$$E = \frac{1}{2}\sum_{t=1}^{N}\sum_{k=1}^{c}(d_k(t) - y_k(t))^2$$

where

$$\begin{aligned} y_k(t) &= f_2(net_k(t)) \\ &= f_2(\sum_{j=0}^{M} w_{kj}^{(2)} z_j(t)) \\ &= f_2(\sum_{j=0}^{M} w_{kj}^{(2)} f_1(net_j(t))) \\ &= f_2(\sum_{j=0}^{M} w_{kj}^{(2)} f_1(\sum_{i=0}^{d} w_{ji}^{(1)} x_i(t)))) \end{aligned}$$

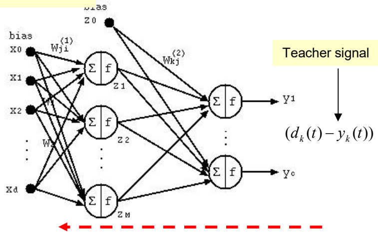
2

Neural Network Training: BP Algorithm

Weights are updated using a gradient descent method:

$$w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} - \mu \frac{\partial E}{\partial w_{ji}^{(1)}}$$

$$w_{kj}^{(2)} \leftarrow w_{kj}^{(2)} - \mu \frac{\partial E}{\partial w_{kj}^{(2)}}$$



The gradient is computed simply as a negative multiplication of the perceptron input and the error signal introduced for that perceptron.

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = -\sum_{t=1}^{N} \delta_{2k}(t) z_j(t) \ , \qquad \delta_{2k}(t) = (d_k(t) - y_k(t)) f_2'(net_k(t))$$

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = -\sum_{t=1}^{N} \delta_{1j}(t) x_i(t) \ , \qquad \delta_{1j}(t) = f_1'(net_j(t)) \sum_{k=1}^{c} w_{kj}^{(2)} \delta_{2k}(t)$$

Feed-forward Network Mapping

Compute outputs of 1st-layer perceptron:

$$net_j^{(1)} = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^{d} w_{ji}^{(1)} x_i$$

$$z_j = f_1(net_j^{(1)})$$

Compute outputs of 2nd-layer perceptron:

$$net_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f_2(net_k^{(2)})$$

In summary, the output of network:

$$y_k = f_2\left(\sum_{j=0}^{M} w_{kj}^{(2)} f_1\left(\sum_{i=0}^{d} w_{ji}^{(1)} x_i\right)\right)$$

Gradient Computing

(1) Gradient for the weights of output-layer perceptron:

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \sum_{t=1}^{N} \frac{\partial E}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial net_k^{(2)}(t)} \frac{\partial net_k^{(2)}(t)}{\partial w_{kj}^{(2)}}$$

$$= -\sum_{t=1}^{N} \boxed{(d_k(t) - y_k(t)) f_2'(net_k^{(2)}(t))} z_j(t)$$

$$= -\sum_{t=1}^{N} \delta_{2k}(t) z_j(t)$$

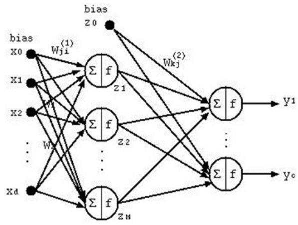
where $\delta_{2k}(t) = (d_k(t) - y_k(t)) f_2'(net_k^{(2)}(t))$

Reminder:

$$E = \frac{1}{2}\sum_{t=1}^{N}\sum_{k=1}^{c}(d_k(t) - y_k(t))^2$$

$$net_k^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f_2(net_k^{(2)})$$



Gradient Computing (cnt'd)

(2) Gradient for weights of hidden or input layer perceptron:

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \sum_{t=1}^{N} \frac{\partial E}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial net_k^{(2)}(t)} \frac{\partial net_k^{(2)}(t)}{\partial z_j(t)} \frac{\partial z_j(t)}{\partial net_j^{(1)}(t)} \frac{\partial net_j^{(1)}(t)}{\partial w_{ji}^{(1)}}$$

$$= -\sum_{t=1}^{N}\sum_{k=1}^{c} \boxed{(d_k(t) - y_k(t)) f_2'(net_k^{(2)}(t))} w_{kj}^{(2)} f_1'(net_j^{(1)}(t)) x_i(t)$$

$$= -\sum_{t=1}^{N} \boxed{\sum_{k=1}^{c} \delta_{2k}(t) w_{kj}^{(2)} f_1'(net_j^{(1)}(t))} x_i(t)$$

$$= -\sum_{t=1}^{N} \delta_{1j}(t) x_i(t)$$

where $\delta_{1j}(t) = f_1'(net_j^{(1)}(t)) \sum_{k=1}^{c} w_{kj}^{(2)} \delta_{2k}(t)$

Reminder:

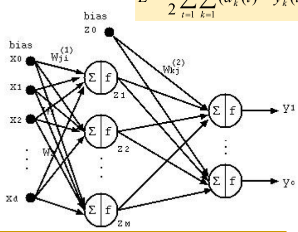
$$net_j^{(1)} = \sum_{i=0}^{d} w_{ji}^{(1)} x_i$$

$$z_j = f_1(net_j^{(1)})$$

$$net_k^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f_2(net_k^{(2)})$$

$$E = \frac{1}{2}\sum_{t=1}^{N}\sum_{k=1}^{c}(d_k(t) - y_k(t))^2$$



Error Backward Propagation

(3) Summary of the gradient computing:

For weights of output layer perceptron:

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = -\sum_{t=1}^{N} \delta_{2k}(t) z_j(t)$$

$$\delta_{2k}(t) = (d_k(t) - y_k(t)) f_2^{'}(net_k^{(2)}(t))$$

For weights of hidden or input layer perceptron

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = -\sum_{t=1}^{N} \delta_{1j}(t) x_i(t)$$

$$\delta_{1j}(t) = f_1^{'}(net_j^{(1)}(t)) \sum_{k=1}^{c} w_{kj}^{(2)} \delta_{2k}(t)$$

7

Implementing BP Algorithm

To implement BP algorithm, we need to create variables for each layer of perceptron:

- the weight matrix $w_{ji}^{(1)}$ and $w_{kj}^{(2)}$,
- the net input vectors: $\text{net}_j^{(1)}$ and $\text{net}_k^{(2)}$
- the output vectors of perceptron,

  $z_j = f(\text{net}_j^{(1)})$ and $y_k = f(\text{net}_k^{(2)})$
- the "error" vectors, δ_{1j} and δ_{2k}

8

(Cont'd)

Step 1: Forward Propagation

Compute the activation for each hidden note, z_j, $i=1,\ldots,M$:

$$net_j^{(1)} = \sum_{i=0}^{d} w_{ji}^{(1)} x_i \quad \text{and} \quad z_j = f_1(net_j^{(1)})$$

Compute the activation for each output node, y_k, $k=1,\ldots,c$:

$$net_k^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \quad \text{and} \quad y_k = f_2(net_k^{(2)})$$

(Cont'd)

Step 2: Backward Propagation

Compute error signal for each output node, δ_{2k}, $k=1,\ldots,c$:

$$\delta_{2k} = (d_k - y_k) f_2'(net_k^{(2)})$$

Compute error signal for each hidden node, δ_{1j}, $j=1,\ldots,M$:

$$\delta_{1j} = f_1'(net_j^{(1)}) \sum_{k=1}^{c} w_{kj}^{(2)} \delta_{2k}$$

(Cont'd)

Step 3: Accumulate gradients over the input patterns

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = -\sum_{t=1}^{N} \delta_{2k}(t) z_j(t)$$

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = -\sum_{t=1}^{N} \delta_{1j}(t) x_i(t)$$

Step 4: After repeat Step 1 to 3 for all patterns, update the weights:

$$w_{kj}^{(2)} \leftarrow w_{kj}^{(2)} - \mu \frac{\partial E}{\partial w_{kj}^{(2)}}$$

$$w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} - \mu \frac{\partial E}{\partial w_{ji}^{(1)}}$$