Back Propagation Networks

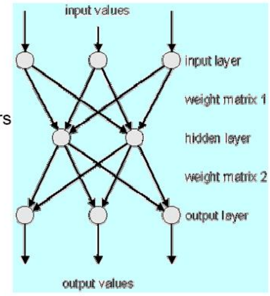
MLP Architecture

The Multi-Layer-Perceptron was first introduced by M. Minsky and S. Papert in 1969

Type:
Feedforward

Neuron layers:
1 input layer
1 or more hidden layers
1 output layer

Learning Method:
Supervised

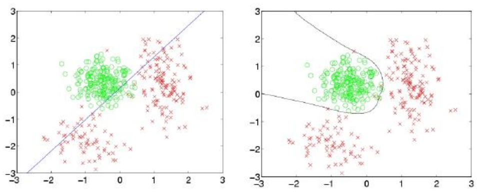


Terminology/Conventions

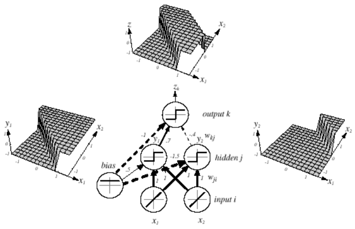
- Arrows indicate the direction of data flow.
- The first layer, termed **input layer**, just contains the input vector and does not perform any computations.
- The second layer, termed **hidden layer**, receives input from the input layer and sends its output to the **output layer**.
- After applying their activation function, the neurons in the output layer contain the output vector.

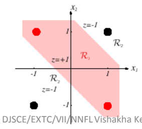
From perceptrons to multilayer perceptrons

Why?



MLP





MLP

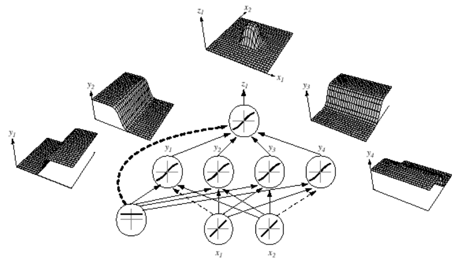
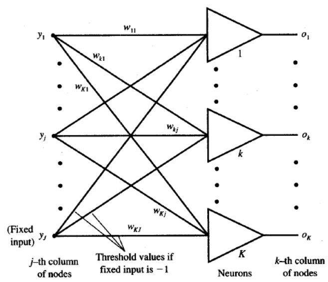


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Why the MLP?

- The single-layer perceptron classifiers discussed previously can only deal with linearly separable sets of patterns.
- The multilayer networks to be introduced here are the most widespread neural network architecture
  - Made useful until the 1980s, because of lack of efficient training algorithms (McClelland and Rumelhart 1986)
  - The introduction of the backpropagation training algorithm.

Single layer network



Single-layer network with continuous perceptrons.

Input ; Output ;Weight Vectors and Target O/p Vector

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_J \end{bmatrix}, \quad \mathbf{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_K \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1J} \\ w_{21} & w_{22} & \cdots & w_{2J} \\ \vdots & \vdots & \cdots & \vdots \\ w_{K1} & w_{K2} & \cdots & w_{KJ} \end{bmatrix} \qquad \mathbf{d} \triangleq \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_K \end{bmatrix}$$

- Generalized Error Expression for pattern p is

$$E_p = \frac{1}{2} \sum_{k=1}^{K} (d_{pk} - o_{pk})^2$$

- Required weight adjustment is:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}$$

each node in layer k, $k = 1, 2, \ldots, K$, we can write using

$$net_k = \sum_{j=1}^{J} w_{kj} y_j$$

the neuron's output is

$$o_k = f(net_k)$$

The *error signal term* δ called *delta* produced by the k'th neuron is defined for this layer as follows

$$\delta_{ok} \triangleq -\frac{\partial E}{\partial(net_k)}$$

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial(net_k)} \cdot \frac{\partial(net_k)}{\partial w_{kj}}$$

This is because error contribution in E is only affect W_{kj} for k=1,2 ,3.....j

$$\frac{\partial(net_k)}{\partial w_{kj}} = y_j \qquad \frac{\partial E}{\partial w_{kj}} = -\delta_{ok} y_j \qquad \Delta w_{kj} = \eta \delta_{ok} y_j,$$

- Error signal δ_{ok} is given by

$$\delta_{ok} = -\frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial(net_k)}$$

$$f'_k(net_k) \triangleq \frac{\partial o_k}{\partial(net_k)}$$

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k)$$

$$\delta_{ok} = (d_k - o_k) f_k'(net_k), \quad \text{for } k = 1, 2, \ldots, K$$

$$\Delta w_{kj} = \eta (d_k - o_k) f_k'(net_k) y_j$$

$$w'_{kj} = w_{kj} + \Delta w_{kj}$$

For the unipolar continuous activation function

$$f'(net) = \frac{\exp(-net)}{[1+\exp(-net)]^2}$$

$$f'(net) = \frac{1}{1+\exp(-net)} \cdot \frac{1+\exp(-net)-1}{1+\exp(-net)}$$

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

$$\delta_{ok} = (d_k - o_k)o_k(1-o_k)$$

The delta value for the bipolar continuous activation function :

$$\delta_{ok} = \frac{1}{2}(d_k - o_k)(1-o_k^2)$$

$$f'(net) = \frac{1}{2}(1-o^2)$$

$$w'_{kj} = w_{kj} + \eta(d_k - o_k)o_k(1-o_k)y_j$$

for

$$o_k = \frac{1}{1+\exp(-net_k)}$$

$$w'_{kj} = w_{kj} + \frac{1}{2}\eta(d_k - o_k)(1-o_k^2)y_j$$

for

$$o_k = 2\left(\frac{1}{1+\exp(-net_k)} - \frac{1}{2}\right)$$

What is Backpropagation?

- Supervised Error Back-propagation Training
  - The mechanism of backward error transmission (delta learning rule) is used to modify the synaptic weights of the internal (hidden) and output layers
    - The mapping error can be propagated into hidden layers

Reference: Clara Boyd

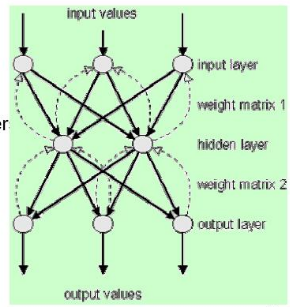
Architecture: Backpropagation Network

The Backpropagation Net was first introduced by G.E. Hinton, E. Rumelhart and R.J. Williams in 1986

Type:
Feedforward

Neuron layers:
1 input layer
1 or more hidden layer
1 output layer

Learning Method:
Supervised

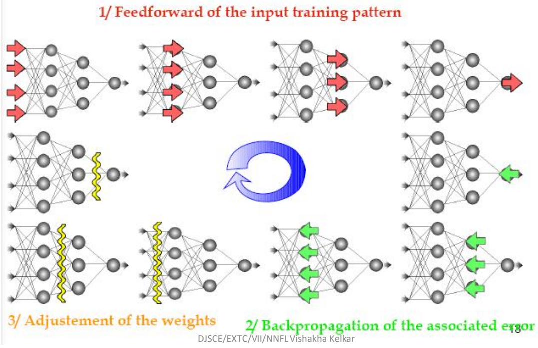


11

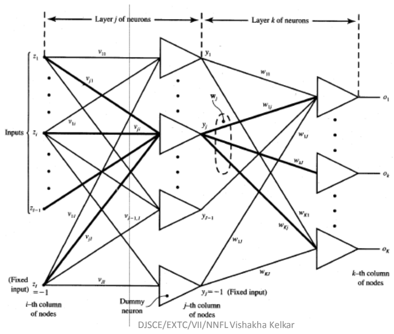
Backpropagation Preparation

- **Training Set**
  A collection of input-output patterns that are used to train the network
- **Testing Set**
  A collection of input-output patterns that are used to assess network performance
- **Learning Rate-α**
  A scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments

Backpropagation training cycle

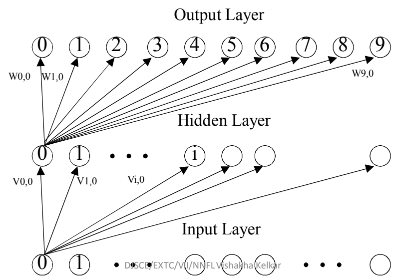


MLP



Backpropagation

- General multi-layered neural network



EBP Algorithm (Generalised Delta Rule)

derive a general expression for the weight increment

Δv_{ji} for any layer of neurons that is not an output layer.

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}}, \quad \text{for } j = 1, 2, \ldots, J \text{ and } i = 1, 2, \ldots, I$$

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial (net_j)} \cdot \frac{\partial (net_j)}{\partial v_{ji}}$$

$$\Delta v_{ji} = \eta \delta_{yj} z_i$$

where δ_{yj} is the error signal term of the hidden layer having output $\mathbf{y}$.

$$\delta_{yj} \triangleq -\frac{\partial E}{\partial (net_j)}, \quad \text{for } j = 1, 2, \ldots, J$$

In contrast to the output layer neurons' excitation net_k, which affected the k'th neuron output only, the net_j contributes now to *every* error component in the error sum containing K terms:

$$\delta_{yj} = -\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial (net_j)}$$

where

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k=1}^{K}\{d_k - f[net_k(\mathbf{y})]\}^2\right)$$

$$\frac{\partial y_j}{\partial (net_j)} = f_j'(net_j)$$

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}(d_k - o_k)\frac{\partial}{\partial y_j}\{f[net_k(\mathbf{y})]\}$$

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}(d_k - o_k) f'(net_k)\frac{\partial (net_k)}{\partial y_j}$$

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}\delta_{ok} w_{kj}$$

$$\frac{\partial E}{\partial y_j} = -\sum_{k=1}^{K}\delta_{ok} w_{kj}$$

$$\delta_{yj} = f_j'(net_j)\sum_{k=1}^{K}\delta_{ok} w_{kj},$$

$$\Delta v_{ji} = \eta f_j'(net_j) z_i \sum_{k=1}^{K}\delta_{ok} w_{kj}, \quad \text{for } j = 1, 2, \ldots, J \text{ and } i = 1, 2, \ldots, I$$

EBP Algorithm

Step 1: $\eta > 0$, E_{max} chosen.
Weights $\mathbf{W}$ and $\mathbf{V}$ are initialized at small random values; $\mathbf{W}$ is $(K \times J)$, $\mathbf{V}$ is $(J \times I)$.

$$q \leftarrow 1, \; p \leftarrow 1, \; E \leftarrow 0$$

Step 2: Training step starts here (See Note 1 at end of list.)
Input is presented and the layers' outputs computed [$f(net)$ as in (2.3a) is used]:

$$\mathbf{z} \leftarrow \mathbf{z}_p, \; \mathbf{d} \leftarrow \mathbf{d}_p$$

$$y_j \leftarrow f(\mathbf{v}_j^t \mathbf{z}), \quad \text{for } j = 1, 2, \ldots, J$$

where $\mathbf{v}_j$, a column vector, is the j'th row of $\mathbf{V}$, and

$$o_k \leftarrow f(\mathbf{w}_k^t \mathbf{y}), \quad \text{for } k = 1, 2, \ldots, K$$

where $\mathbf{w}_k$, a column vector, is the k'th row of $\mathbf{W}$.

EBP Algorithm

Step 3: Error value is computed:

$$E \leftarrow \frac{1}{2}(d_k - o_k)^2 + E, \quad \text{for } k = 1, 2, \ldots, K$$

Step 4: Error signal vectors $\boldsymbol{\delta}_o$ and $\boldsymbol{\delta}_y$ of both layers are computed. Vector $\boldsymbol{\delta}_o$ is $(K \times 1)$, $\boldsymbol{\delta}_y$ is $(J \times 1)$. (See Note 2 at end of list.)
The error signal terms of the output layer in this step are

$\delta_{ok} = (d_k - o_k) f_k'(net_k)$

$$\delta_{ok} = \frac{1}{2}(d_k - o_k)(1 - o_k^2), \quad \text{for } k = 1, 2, \ldots, K$$

The error signal terms of the hidden layer in this step are

$\delta_{yj} = f_j'(net_j) \sum_{k=1}^{K} \delta_{ok} w_{kj},$

$$\delta_{yj} = \frac{1}{2}(1 - y_j^2) \sum_{k=1}^{K} \delta_{ok} w_{kj}, \quad \text{for } j = 1, 2, \ldots, J$$

$$\delta_{ok} = (d_k - o_k)(1 - o_k)o_k, \quad \text{for } k = 1, 2, \ldots, K$$

$$\delta_{yj} = y_j(1 - y_j) \sum_{k=1}^{K} \delta_{ok} w_{kj}, \quad \text{for } j = 1, 2, \ldots, J$$

EBP Algorithm

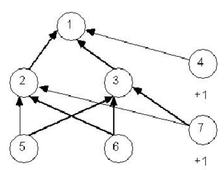
Step 5: Output layer weights are adjusted:

$$w_{kj} \leftarrow w_{kj} + \eta \delta_{ok} y_j, \quad \text{for } k = 1, 2, \ldots, K \text{ and } j = 1, 2, \ldots, J$$

Step 6: Hidden layer weights are adjusted:

$$v_{ji} \leftarrow v_{ji} + \eta \delta_{yj} z_i, \quad \text{for } j = 1, 2, \ldots, J \text{ and } i = 1, 2, \ldots, I$$

EBP Example



a) All weights initially 1.0

Training Patterns

1) 0 0 -> 1
2) 0 1 -> 0

a learning constant of 1.

Nodes 4, 5, 6, and 7 just input nodes and do not have a sigmoidal output.

BP-1)

net2 = $\sum w_i x_i$ = (1*0 + 1*0 + 1*1) = 1

net3 = 1

o2 = $1/(1+e^{-net}) = 1/(1+e^{-1}) = 1/(1+.368) = .731$

o3 = .731

o4 = 1

net1 = (1*.731 + 1*.731 + 1) = 2.462

o1 = $1/(1+e^{-2.462})$ = .921

$\delta 1$ = (t1 - o1) o1 (1 - o1) = (1 - .921) .921 (1 - .921) = .00575

$\Delta w21 = \eta\, \delta_j\, o_i = \eta\, \delta_1\, o_2$ = 1 * .00575 * .731 = .00420

$\Delta w31$ = 1 * .00575 * .731 = .00420

$\Delta w41$ = 1 * .00575 * 1 = .00575

$\delta 2 = o_j (1 - o_j) \sum \delta_k w_{jk} = o_2 (1 - o_2)\ \delta_1 w_{21} =$
.731 (1 - .731) (.00575 * 1) = .00113

$\delta 3$ = .00113

$\Delta w52 = \eta\ \delta_j\ o_i = \eta\ \delta 2\ o_5$ = 1 * .00113 * 0 = 0

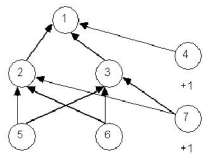
$\Delta w62$ = 0

$\Delta w72$ = 1 * .00113 * 1 = .00113

$\Delta w53$ = 0

$\Delta w63$ = 0

$\Delta w73$ = 1 * .00113 * 1 = .00113



Second pass for 0 1 -> 0

Modifi ed Weights:

w21 = 1.004 2 w31 = 1.004 2 w41 = 1.00575

w52 = 1 w62 = 1 w72 = 1.00113

w53 = 1 w63 = 1 w73 = 1.00113

net2 = $\sum w_i x_i$ = (1*0 + 1*1 + 1*1.00113) = 2.00113

net3 = 2.00113

o2 = 1/(1+e $^{-net}$) = 1/(1+e $^{-2.00113}$) = .881

o3 = .881

o4 = 1

net1 = (1.0042*.88 1 + 1.0042*.881 + 1.00575*1) = 2.775

o1 = 1/(1+e $^{-2.775}$)= .941

$\delta 1$ = (t1 - o1) o1 (1 - o1) = (0 - .941) .941 (1 - .941) = -.0522

$\Delta w21 = \eta\ \delta_j\ o_i = \eta\ \delta_1\ o_2$ = 1 * -.0522 * .881 = -.0460

$\Delta w31$ = 1 * -.0522 * .881 = -.0460

$\Delta w41$ = 1 * -.0522 * 1 = -.0522

$\delta 2 = o_j (1 - o_j) \sum \delta_k w_{jk} = o_2 (1 - o_2)\ \delta_1 w_{21} =$
.881 (1 - .881) (-.0522 * 1.0042) = -.00547

$\delta 3$ = -.00547

$\Delta w52 = \eta\ \delta_j\ o_i = \eta\ \delta 2\ o_5$ = 1 * -.00547 * 0 = 0

$\Delta w62$ = 1 * (-.00547) * 1 = -.00547

$\Delta w72$ = 1 * (-.00547) * 1 = -.00547

$\Delta w53$ = 0

$\Delta w63$ = -.00547

$\Delta w73$ = 1 * (-.00547) * 1 = -.00547

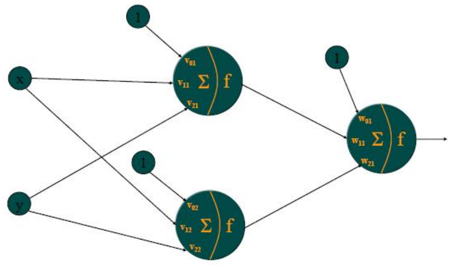
w21 = 1.0042 - .0460 =	.958
w31 = 1.0042 - .0460 =	.958
w41 = 1.00575 - .0522 =	.954
w52 = 1 + 0 =	1
w62 = 1 - .00547 =	.995
w72 = 1.00113 - .00547 =	.996
w53 = 1 + 0 =	1
w63 = 1 - .00547 =	.995
w73 = 1.00113 - .00547 =	.996

Example 1

- The XOR function could not be solved by a single layer perceptron network
- The function is:

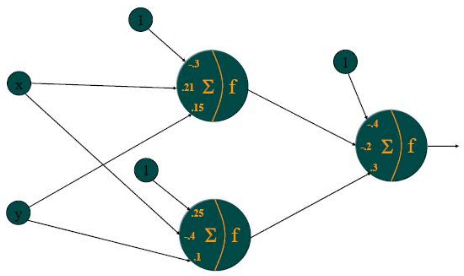
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

XOR Architecture

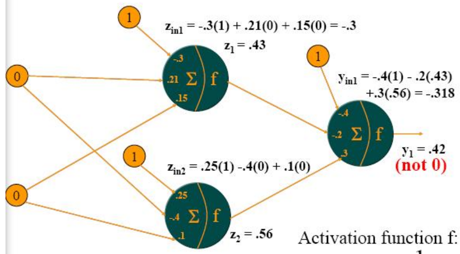


Initial Weights

Randomly assign small weight values:



Feedfoward – 1st Pass

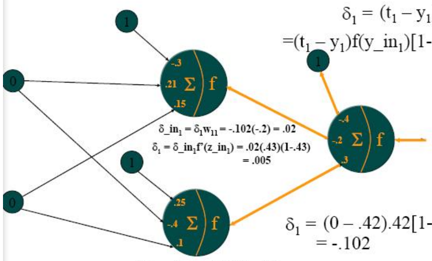


Training Case: (0 0 0)

Activation function f:

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

Backpropagate



$\delta_1 = (t_1 - y_1)f'(y\_in_1)$
$=(t_1 - y_1)f(y\_in_1)[1- f(y\_in_1)]$

$\delta\_in_1 = \delta_1 w_{11} = -.102(-.2) = .02$
$\delta_1 = \delta\_in_1 f'(z\_in_1) = .02(.43)(1-.43)$
$= .005$

$\delta_1 = (0 - .42).42[1-.42]$
$= -.102$

$\delta\_in_2 = \delta_1 w_{21} = -.102(.3) = -.03$
$\delta_2 = \delta\_in_2 f'(z\_in_2) = -.03(.56)(1-.56)$
$= -.007$

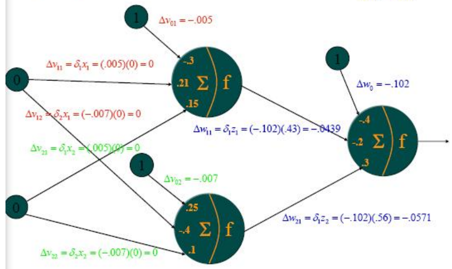
32

DJSCE/EXTC/VII/NNFL Vishakha Kelkar

Calculate the Weights – First Pass

$\Delta v_{ij} = \alpha \delta_j x_i \quad j = 1,2$
$\Delta v_{0j} = \alpha \delta_j$

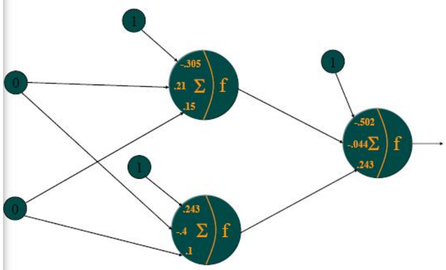
$\Delta w_{j1} = \alpha \delta_1 z_j \quad j = 1,2$
$\Delta w_0 = \alpha \delta_1$



$\Delta v_{01} = -.005$
$\Delta v_{11} = \delta_1 x_1 = (.005)(0) = 0$
$\Delta v_{12} = \delta_2 x_1 = (-.007)(0) = 0$
$\Delta v_{21} = \delta_1 x_2 = (.005)(0) = 0$
$\Delta v_{02} = -.007$
$\Delta v_{22} = \delta_2 x_2 = (-.007)(0) = 0$
$\Delta w_0 = -.102$
$\Delta w_{11} = \delta_1 z_1 = (-.102)(.43) = -.0439$
$\Delta w_{21} = \delta_1 z_2 = (-.102)(.56) = -.0571$

DJSCE/EXTC/VII/NNFL Vishakha Kelkar

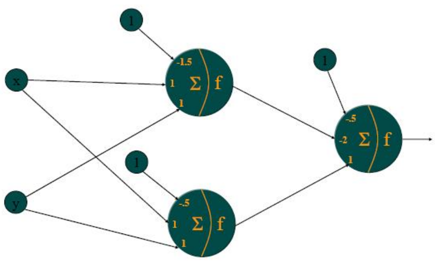
Update the Weights – First Pass



DJSCE/EXTC/VII/NNFL Vishakha Kelkar

Final Result

- After about 500 iterations:



DJSCE/EXTC/VII/NNFL Vishakha Kelkar