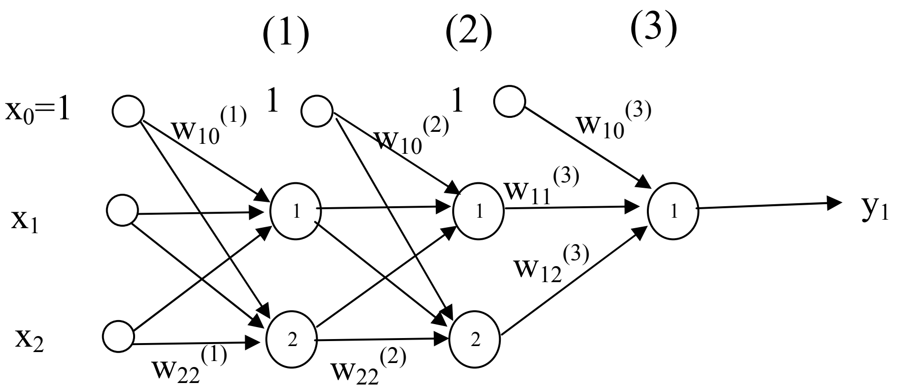
Example: Backpropagation Algorithm

Consider the following Multilayer Perceptrons:



Each neuron has an unipolar sigmoid activation fn:

$$\varphi(v) = \frac{1}{1+e^{-av}}$$

Let a=1, $\varphi'(v) = \varphi(v)(1-\varphi(v))$

For a given training sample, $\{\mathbf{x}, d_1\}$:

Forward pass:

Compute $v_j^{(s)} = \sum_{i=0}^{n_{s-1}} x_i^{(s-1)} w_{ji}^{(s)}$

$x_{out,j}^{(s)} = \varphi(v_j^{(s)})$

for layer 1, 2, 3 (i.e. s=1,2,3)

Backward pass:

For output layer,

$$\delta_1^{(3)} = (d_1 - y_1)\varphi'(v_1^{(3)})$$
$$= (d_1 - y_1)y_1(1 - y_1)$$

where d_1 is the desired value corresponding to the input **x**

For hidden layers,

$$\delta_1^{(2)} = \varphi'(v_1^{(2)})\sum_{h=1}^{1} w_{h1}^{(3)}\delta_h^{(3)}$$
$$= x_{out,1}^{(2)}(1 - x_{out,1}^{(2)})w_{11}^{(3)}\delta_1^{(3)}$$
$$\delta_2^{(2)} = \varphi'(v_2^{(2)})\sum_{h=1}^{1} w_{h2}^{(3)}\delta_h^{(3)}$$
$$= x_{out,2}^{(2)}(1 - x_{out,2}^{(2)})w_{12}^{(3)}\delta_1^{(3)}$$
$$\delta_1^{(1)} = \varphi'(v_1^{(1)})\sum_{h=1}^{2} w_{h1}^{(2)}\delta_h^{(2)}$$
$$= x_{out,1}^{(1)}(1 - x_{out,1}^{(1)})(w_{11}^{(2)}\delta_1^{(2)} + w_{21}^{(2)}\delta_2^{(2)})$$
$$\delta_2^{(1)} = \varphi'(v_2^{(1)})\sum_{h=1}^{2} w_{h2}^{(2)}\delta_h^{(2)}$$
$$= x_{out,2}^{(1)}(1 - x_{out,2}^{(1)})(w_{12}^{(2)}\delta_1^{(2)} + w_{22}^{(2)}\delta_2^{(2)})$$

Then wt update rules are:

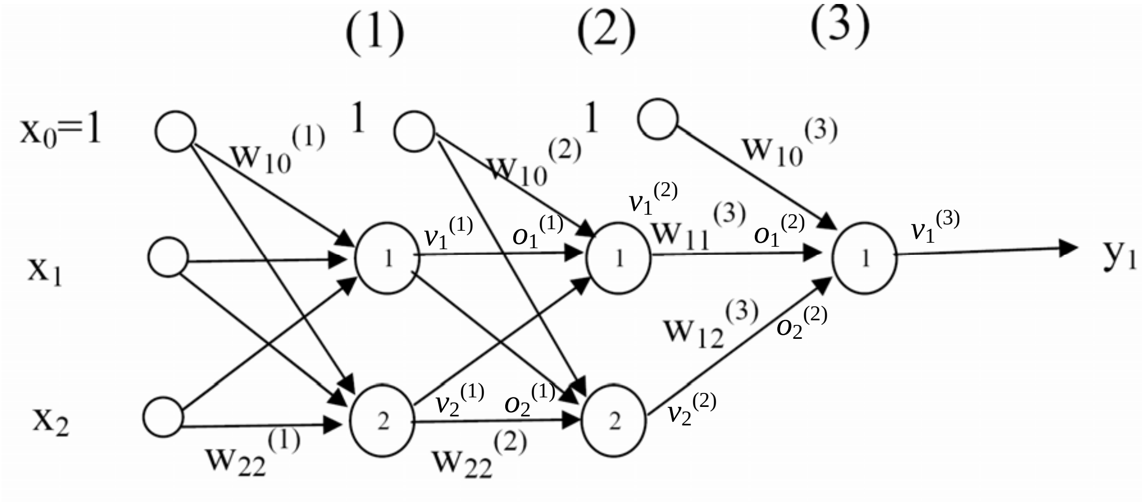
$$\Delta w_{10}^{(3)} = \eta\delta_1^{(3)}, \Delta w_{11}^{(3)} = \eta\delta_1^{(3)}x_{out,1}^{(2)}, \ \Delta w_{12}^{(3)} = \eta\delta_1^{(3)}x_{out,2}^{(2)}$$
$$\Delta w_{10}^{(2)} = \eta\delta_1^{(2)}, \Delta w_{11}^{(2)} = \eta\delta_1^{(2)}x_{out,1}^{(1)}, \ \Delta w_{12}^{(2)} = \eta\delta_1^{(2)}x_{out,2}^{(1)}$$
$$\Delta w_{20}^{(2)} = \eta\delta_2^{(2)}, \Delta w_{21}^{(2)} = \eta\delta_2^{(2)}x_{out,1}^{(1)}, \ \Delta w_{22}^{(2)} = \eta\delta_2^{(2)}x_{out,2}^{(1)}$$
$$\Delta w_{10}^{(1)} = \eta\delta_1^{(1)}, \Delta w_{11}^{(1)} = \eta\delta_1^{(1)}x_1, \ \Delta w_{12}^{(1)} = \eta\delta_1^{(1)}x_2$$
$$\Delta w_{20}^{(1)} = \eta\delta_2^{(1)}, \Delta w_{21}^{(1)} = \eta\delta_2^{(1)}x_1, \ \Delta w_{22}^{(1)} = \eta\delta_2^{(1)}x_2$$

Above wt update is for a single training pattern (corresponding to a learning step).

After a learning step is finished, next training pattern is submitted and the learning step is repeated.

Until all the patterns in the training set have been exhausted
→ One epoch

Example: Solve the XOR problem using the following network architecture:

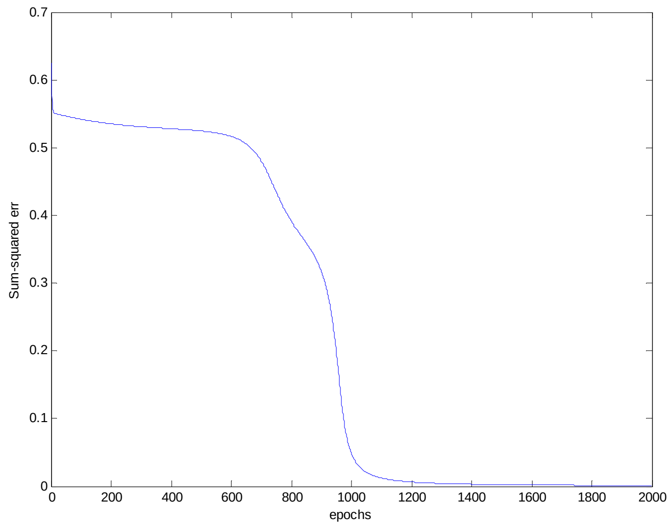


The input vector consists of 4 patterns,

$\boldsymbol{x}_1 = [0\ 0]^T$, $\boldsymbol{x}_2 = [1\ 0]^T$, $\boldsymbol{x}_3 = [1\ 1]^T$, $\boldsymbol{x}_4 = [0, 1]^T$;

which corresponds to the desired vector of: $\boldsymbol{d} = [0, 1, 0, 1]^T$;

For randomly initialized weights and a learning rate of 0.8, the learning stops when the sum-squared err is less than 0.001 or the number of epochs reaches 2000. The convergence is shown in the figure below.



In this case, the output obtained is given as: $\boldsymbol{y} = [0.0219, 0.9769, 0.0239, 0.9768]^T$

After training, the weight matrix of each layer (1, 2, 3) is as follows,

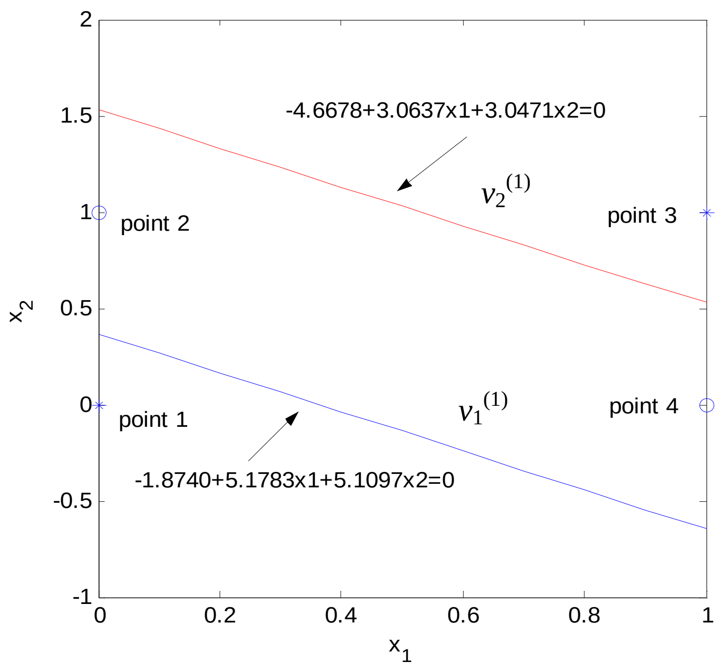
$$\begin{array}{ll} & \quad\text{bias} \\ \boldsymbol{w}^{(1)} = & \begin{bmatrix} -1.87 & 5.17 & 5.10 \\ -4.66 & 3.06 & 3.04 \end{bmatrix} \end{array}$$

$$\begin{array}{ll} & \quad\text{bias} \\ \boldsymbol{w}^{(2)} = & \begin{bmatrix} -1.06 & 3.69 & -5.29 \\ 2.53 & -4.67 & 3.54 \end{bmatrix} \end{array}$$

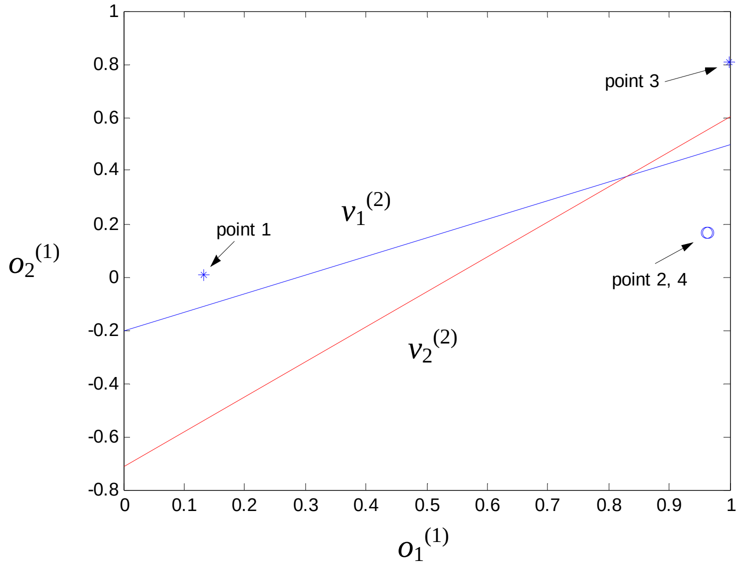
$$\begin{array}{ll} & \quad\text{bias} \\ \boldsymbol{w}^{(3)} = & [-0.36, 6.49, -6.5147] \end{array}$$

Therefore the decision lines for the first hidden layer are:

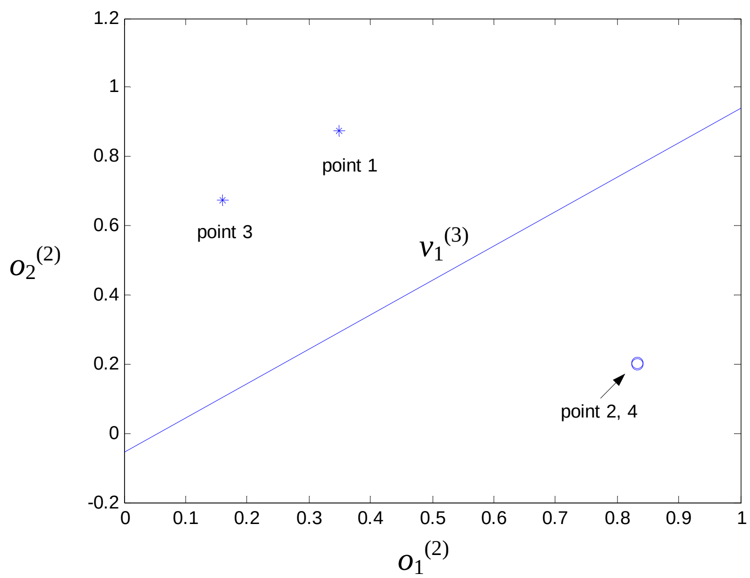
$-1.87 + 5.17 \cdot x_1 + 5.10 \cdot x_2 = 0$ and $-4.66 + 3.06 \cdot x_1 + 3.04 \cdot x_2 = 0$.



The decision lines for the second hidden layer are:



And the decision line for the output layer is:



****
Source codes in Matlab
****

```
clear all;
close all;

X=[0 1 1 0;0 0 1 1];
```

```
X=[ones(1,4);X];
d=[0 1 0 1];


H1=2;H2=2;
No=1;
W1=2*(rand(H1,3)-rand(H1,3));
W2=2*(rand(H2,H1+1)-rand(H2,H1+1));
W3=2*(rand(No,H2+1)-rand(No,H2+1));
err=1;
epoch=1;
yit=0.96;

while (err>0.001) & (epoch<2000)
   err=0;
for iter=1:4

%% begin forward process
for i=1:H1
   V1(i)=W1(i,:)*X(:,iter);
   O1(i)=logsig(V1(i));
end

for i=1:H2
   V2(i)=W2(i,:)*[1;O1'];
   O2(i)=logsig(V2(i));
end

%%output layer
for i=1:No
   V3(i)=W3(i,:)*[1;O2'];
   Y(i,iter)=logsig(V3(i));
end

%%end forward process

%% begin backward pass
for i=1:No
   De3(i)=(d(iter)-Y(i,iter))*Y(i,iter)*(1-Y(i,iter));
end
for i=1:H2
   De2(i)=O2(i)*(1-O2(i))*W3(:,i+1)*De3;
end
for i=1:H1
   De1(i)=O1(i)*(1-O1(i))*(De2*W2(:,i+1));
end
%% end backward pass
```

```
%% weights update
W3=W3+yit*De3*[1;O2']';
for i=1:H2
   W2(i,:)=W2(i,:)+yit*De2(i)*[1;O1']';
end
for i=1:H1
   W1(i,:)=W1(i,:)+yit*De1(i)*X(:,iter)';
end
err=err+0.5*(d(iter)-Y(iter))^2;
end %% end of one epoch
Err(epoch)=err;
epoch=epoch+1;
end
Y
plot(Err)
```