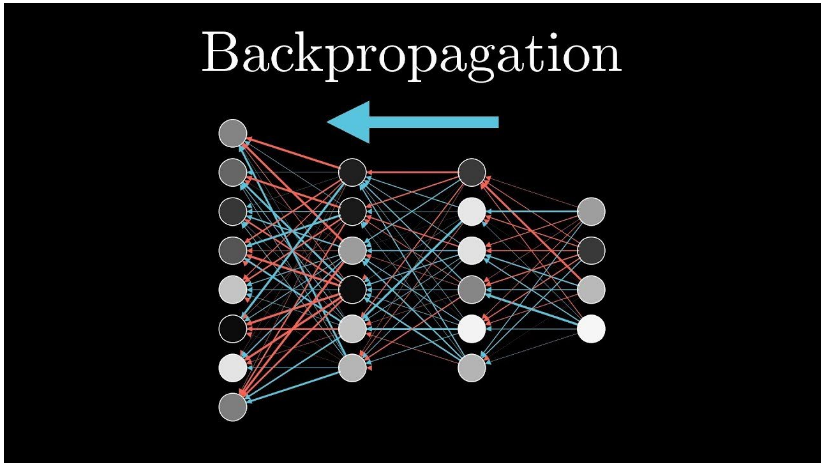
Machine intelligence

Delta learning rule and backpropagation



Dr. Ahmad Al-Mahasneh

In the previous lecture

- Widrow-Hoff Algorithm

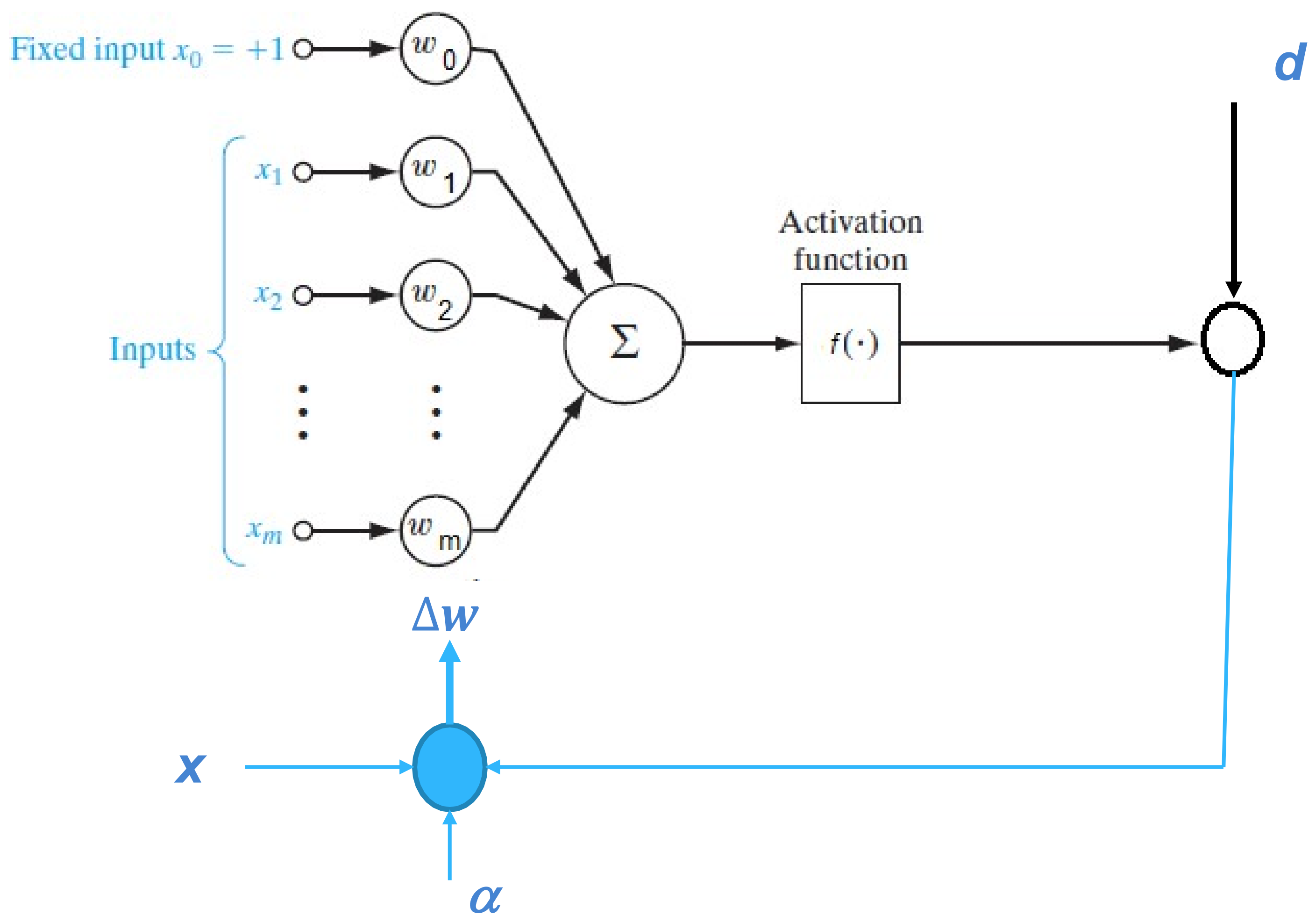
- Perceptron example

- Delta Rule Learning (one neuron)

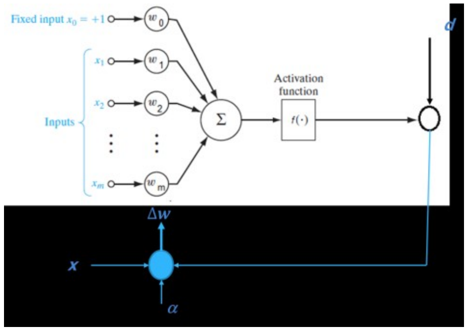
Outline

- Delta Rule Learning (one neuron)
- Example
- MATLAB example
- Delta Rule Learning (multi-neurons)
- Backpropagation

Delta Rule Learning



Delta Rule Learning: single neuron



$\boldsymbol{net} = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}$

Using a linear activation function $\boldsymbol{y} = \boldsymbol{f}(\text{net})$

The error between the network output and the desired output is

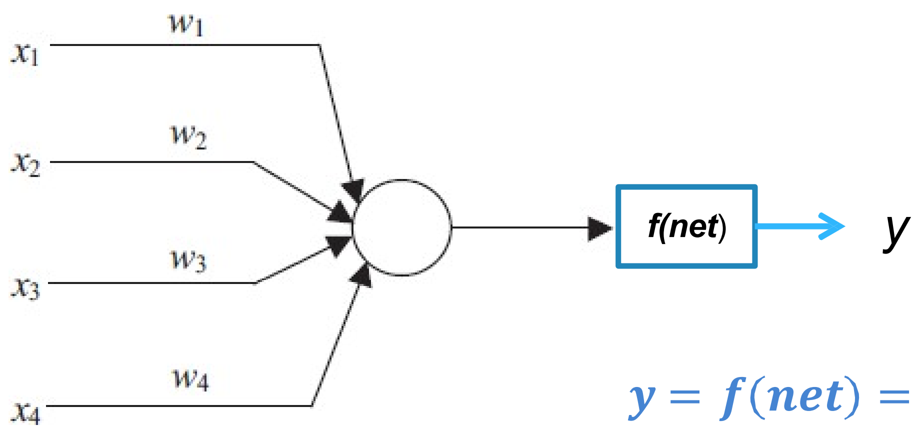
$$E = \frac{1}{2}(d - y)^2$$

The derivative w.r.t. weights is $\frac{dE}{dw_i} = \frac{dE}{dy}\frac{dy}{dw_i} = -(d-y)f'(net)\,x_i$

Update the weighs using delta rule $w_i = w_i - \alpha \frac{dE}{dw_i}$

In vector format: $W = W - \alpha\, \nabla E$

Example



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

$$Note: y' = 0.5(1 - y^2)$$

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}, x^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

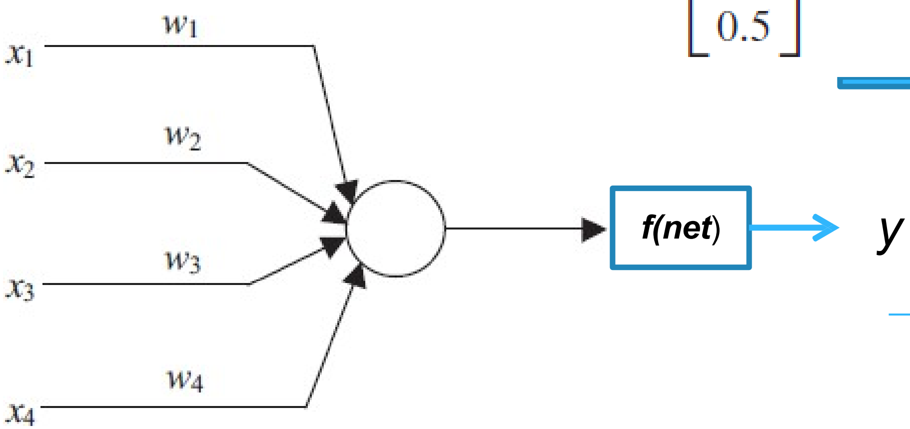
Desired output, d = [-1 -1 1]

Use **Delta Rule** learning to update the weights

with $\alpha = 0.1$

Example

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}, x^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$



Desired output, d = [-1 -1 1]

Iteration One
Pattern One

$$net = w^T x = [1 \quad -1 \quad 0 \quad 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

$$y = \frac{1 - e^{-net}}{1 + e^{-net}} \Rightarrow y = 0.848$$

$$y' = 0.5\,(1 - y^2) = 0.140$$

$$\nabla E = -\alpha\,(d - y)\, y' x$$

$$= -0.1\,(-1 - 0.848)(0.14) \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= 0.0259 \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.0259 \\ -0.0518 \\ 0 \\ -0.0259 \end{bmatrix}$$

$$w = w - \nabla E = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0.0259 \\ -0.0518 \\ 0 \\ -0.0259 \end{bmatrix} = \begin{bmatrix} 0.9741 \\ -0.9482 \\ 0 \\ 0.5259 \end{bmatrix}$$

Example

$$w^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}, x^1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \text{ and } x^3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

Desired output, d = [-1 -1 1]

Pattern Two

$$net = w^T x$$

$$= [\,0.974 \quad -0.948 \quad 0 \quad 0.5259] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= -1.948$$

$$y = \frac{1 - e^{-net}}{1 + e^{-net}} \Rightarrow y = -0.7505$$

$$y' = 0.5\,(1 - y^2\,) = 0.2184$$

$$\nabla E = -\alpha\,(d - y\,)\,y'x$$

$$= -0.1\,(-1 + 0.7505)(0.2184) \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0.0082 \\ -0.0027 \\ -0.0054 \end{bmatrix}$$

$$w = w - \nabla E = \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.5259 \end{bmatrix} - \begin{bmatrix} 0 \\ 0.0082 \\ -0.0027 \\ -0.0054 \end{bmatrix} = \begin{bmatrix} 0.9741 \\ -0.9563 \\ 0.0027 \\ 0.5314 \end{bmatrix}$$

Example

Iteration One

Pattern One

Pattern Two

Pattern Three

Iteration Two

Pattern One

Pattern Two

Pattern Three

Iteration Three

Pattern One

Pattern Two

Pattern Three

…

Two Loops

Iteration Loop

Pattern Loop

Calculate the Error

MATLAB CODE

- **% Delta Rule Example for single neuron**

- w=[1 -1 0 0.5]';
- x1=[1 -2 0 -1]'; x2=[0 1.5 -0.5 -1]'; x3=[-1 1 0.5-1]'; d1 = -1; d2=-1; d3=1;
- a=0.1;

- **for iter = 1:100**

- **% Pattern 1**
- net = w'*x1;
- y1 = (1 - exp(-net)) / (1 + exp(-net));
  yp = 0.5 * (1 - y1^2);
- dE = -a * (d1 - y1)*yp*x1;
- w = w - dE;

- **% Pattern 2**
- net = w'*x2;
- y2 = (1 - exp(-net)) / (1 + exp(-net));
  yp = 0.5 * (1 - y2^2);
- dE = -a * (d2 - y2)*yp*x2;
- w = w - dE;

% Pattern 3

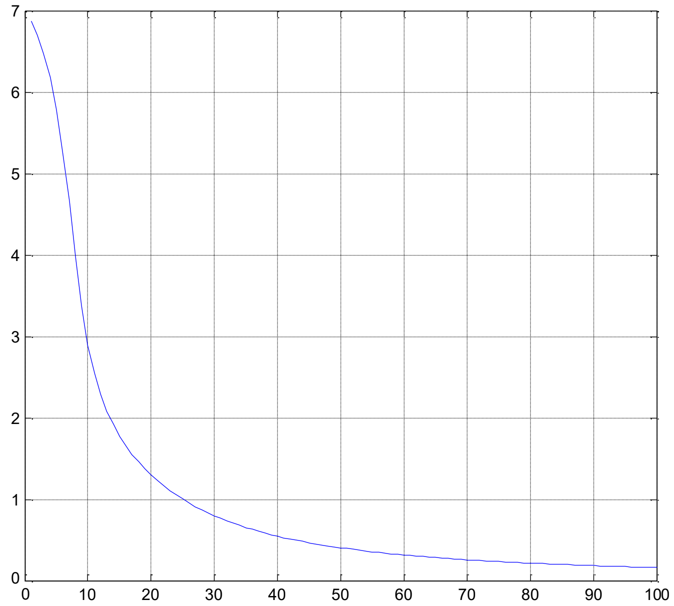
```
    net = w'*x3;
    y3 = ( 1 - exp(-net) ) / ( 1 + exp(-net) );
    yp = 0.5 * ( 1 - y3^2);
    dE = -a * (d3 - y3)*yp*x3;
    w = w - dE;

    Err(iter) = (d1-y1)^2 + (d2-y2)^2 + (d3-y3)^2;
```

End

plot(Err); grid

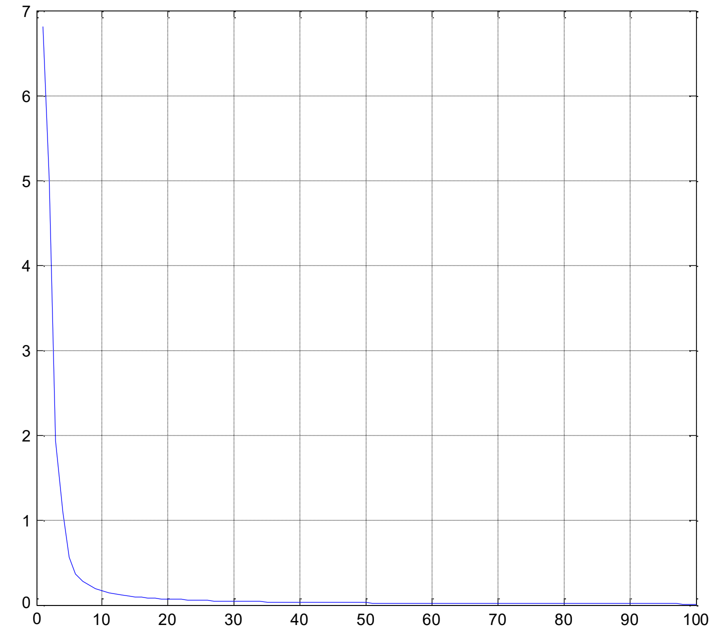
MATLAB Results

$\alpha = 0.1$



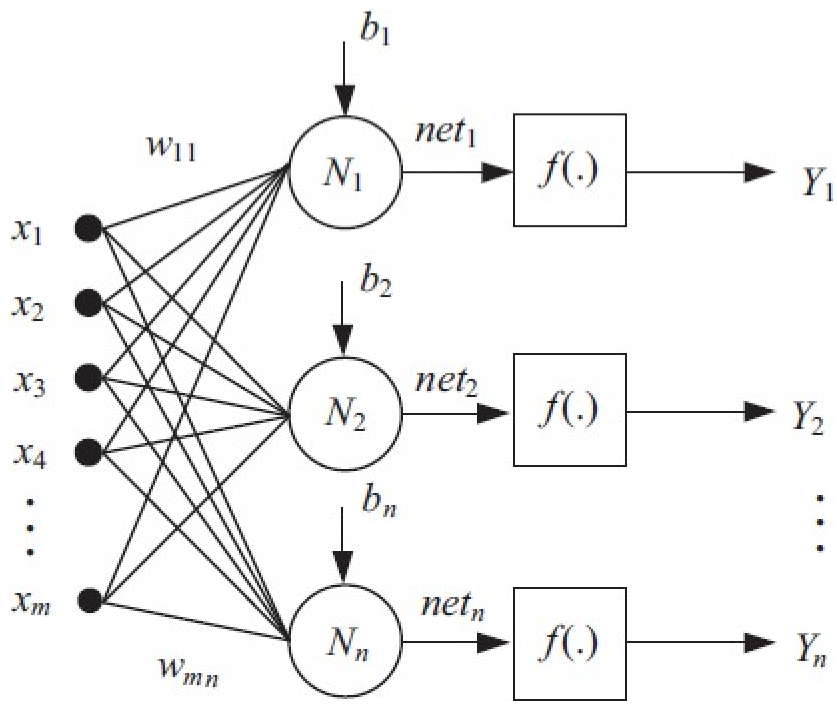
y1 = -0.8897
y2 = -0.7191
y3 = 0.7319

$\alpha = 0.9$



y1 = -0.9669
y2 = -0.9240
y3 = 0.9278

Multi-Neurons



$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \qquad Y = f(W^{\mathsf{T}} \cdot x + b)$$

Delta Rule Learning: Multi-Neurons

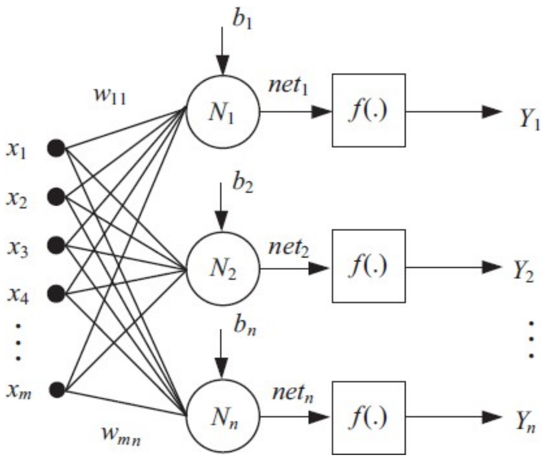
For each output neuron j = 1 : m

$$net_j = \sum_{i=1}^{m} w_{ij}\, x_i$$

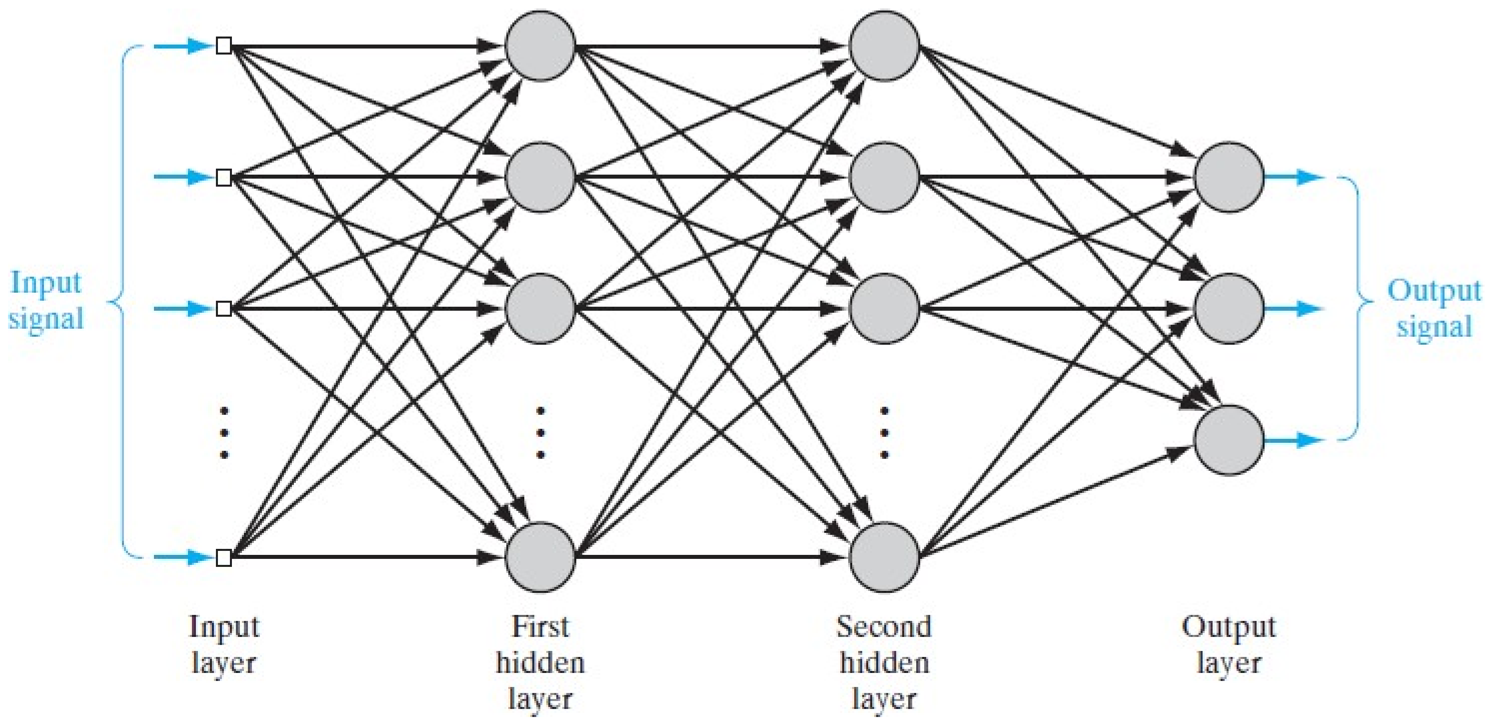
$$y_j = f\left(net_j\right)$$

$$\frac{dE}{dw_{ij}} = -\left(d_j - y_j\right) f'(net_j)\, x_i$$

$$w_{ij} = w_{ij} - \alpha \frac{dE}{dw_{ij}}$$



Multilayer Feedforward Network



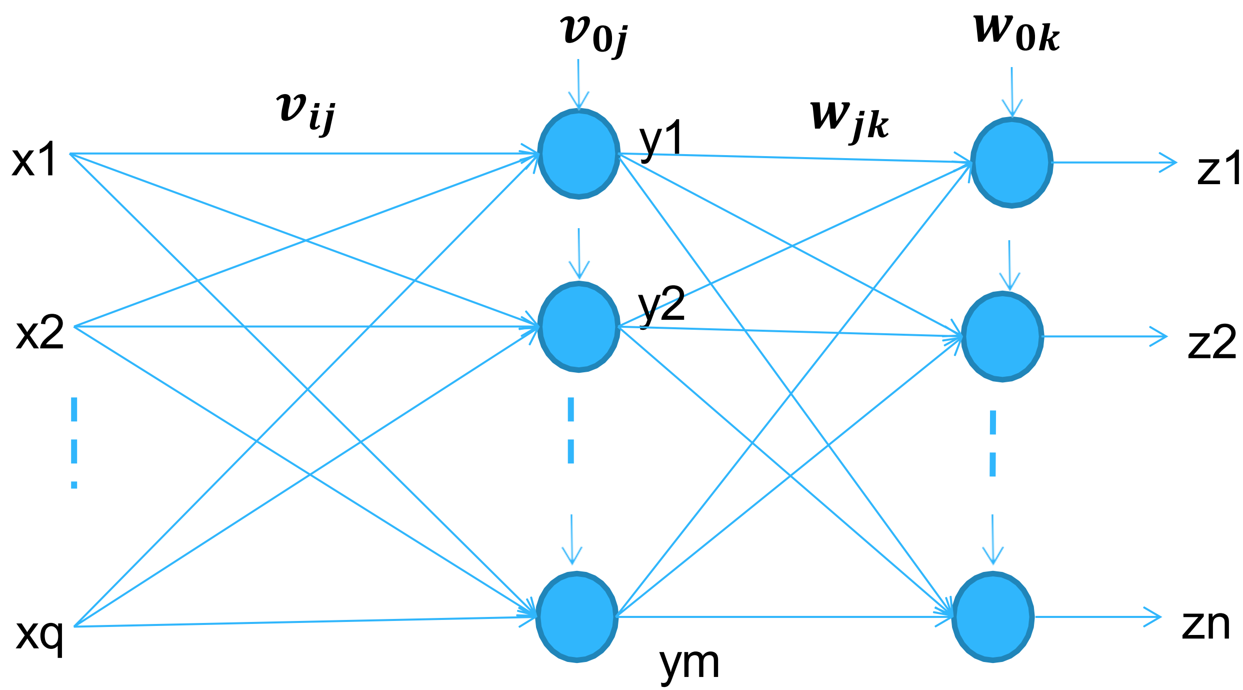
Backpropagation

- Backpropagation is an algorithm for supervised learning of artificial neural networks using gradient descent.

- Backpropagation is short for propagation of errors

- Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It then back-propagates the error to the inner layers.

Three-layer Network



Network Weights

$$V = \begin{bmatrix} v_{11} & \dots & v_{1m} \\ \dots & \dots & \dots \\ v_{q1} & \dots & v_{qm} \end{bmatrix}, W = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ \dots & \dots & \dots \\ w_{m1} & \dots & w_{mn} \end{bmatrix}$$

Input, Hidden, and Output Vectors

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_q \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{bmatrix}, Z = \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_n \end{bmatrix},$$

Bias Weights

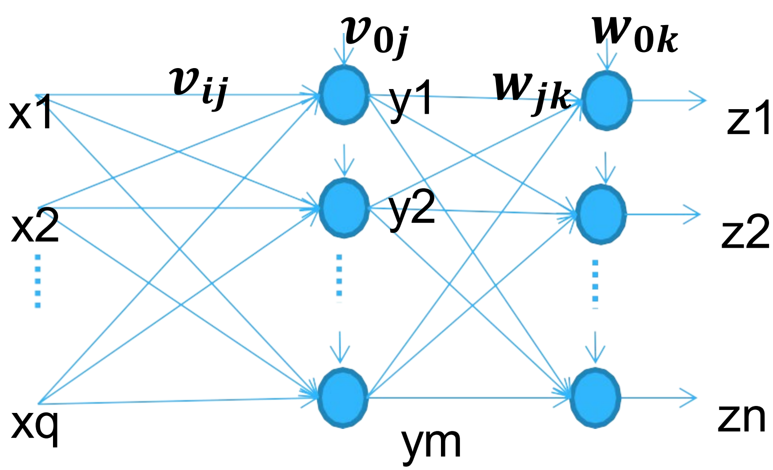
$$V_0 = \begin{bmatrix} v_{01} \\ v_{02} \\ \dots \\ v_{0m} \end{bmatrix}, W_0 = \begin{bmatrix} w_{01} \\ w_{02} \\ \dots \\ w_{0n} \end{bmatrix},$$

Backpropagation Theory

Error for each pattern: $E = \frac{1}{2}\sum_{k=1}^{n}(z_k - d_k)^2$

$$\boldsymbol{Y = f(V^T X + V_o)}$$

$$\boldsymbol{Z = f(W^T Y + W_o)}$$



Step One: Output Weights

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial w_{jk}} = (z_k - d_k)z'_k\, y_j$$

$$\frac{\partial E}{\partial w_{jk}} = \delta_k\, y_j$$ where $$\delta_k = (z_k - d_k)z'_k$$

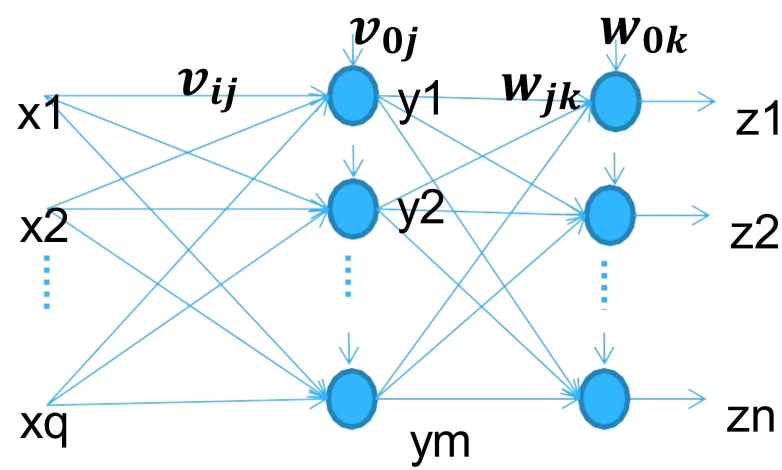
Then, the weight update is: $w_{jk} = w_{jk} - \alpha\delta_k y_j$

Backpropagation Theory

$$E = \frac{1}{2}\sum_{k=1}^{n}(z_k - d_k)^2$$

$$\boldsymbol{Y = f(V^T X + V_o)}$$

$$\boldsymbol{Z = f(W^T Y + W_o)}$$



Step Two: Hidden Weights

$$\frac{\partial E}{\partial v_{ij}} = \sum_{k=1}^{n}\left(\frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial y_j}\right)\frac{\partial y_j}{\partial v_{ij}}$$

$$\frac{\partial E}{\partial v_{ij}} = \sum_{k=1}^{n}(\delta_k w_{jk})\, y'_j x_i$$

$$\frac{\partial E}{\partial v_{ij}} = \delta_j x_i$$

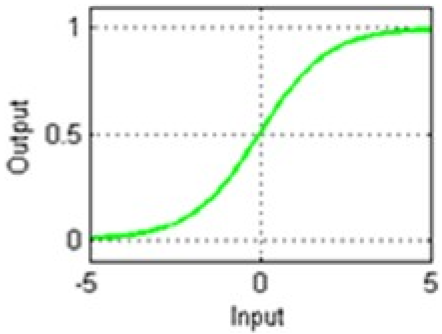
where

$$\delta_j = y'_j \sum_{k=1}^{n}\delta_k w_{jk}$$

Then, the weight update is: $v_{ij} = v_{ij} - \alpha\delta_j x_i$

Function Derivatives: Review

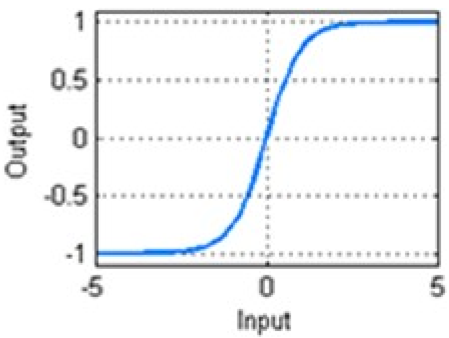
Sigmoidal



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

$$y' = y(1-y)$$

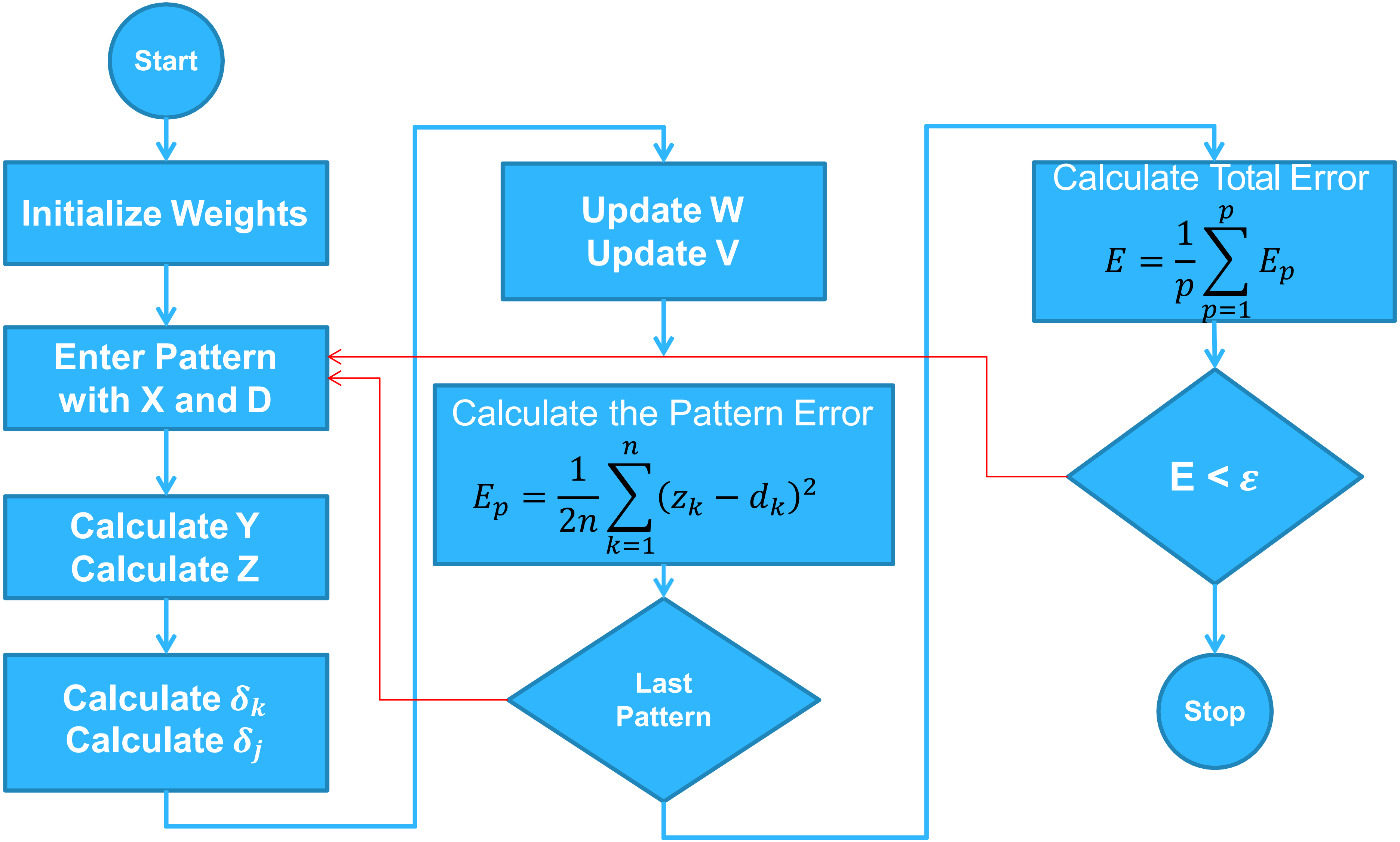
Hyperbolic Tangent



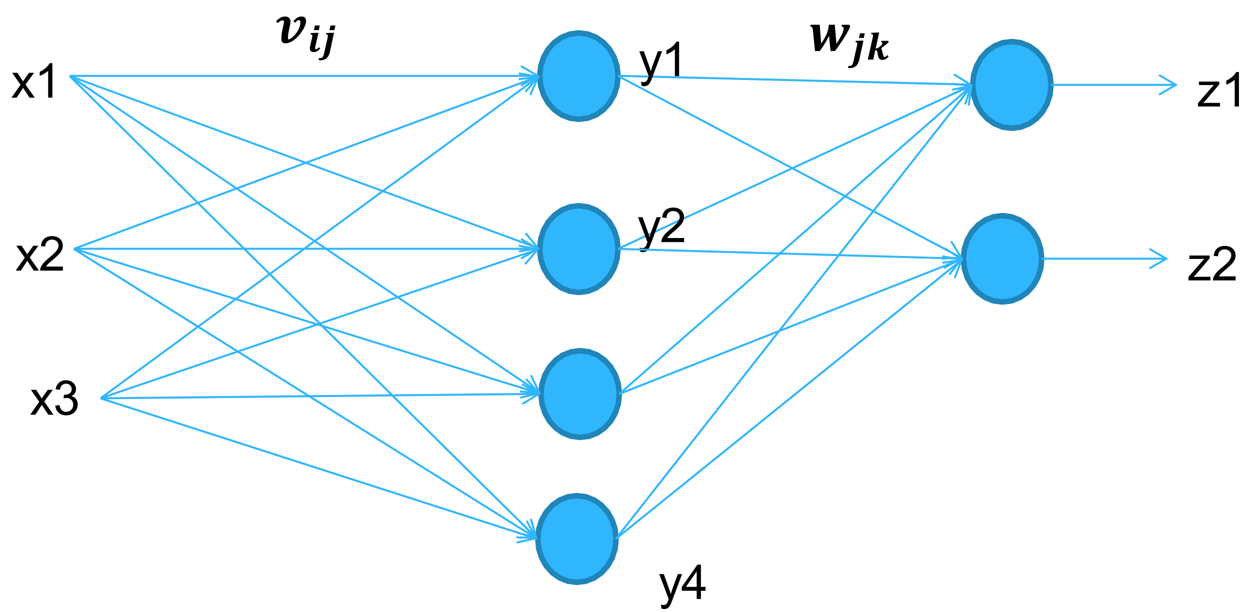
$$y = f(net) = \frac{1-e^{-net}}{1+e^{-net}}$$

$$y' = 0.5(1-y^2)$$

Backpropagation Algorithm



Example: BP for 3-layerNetwork



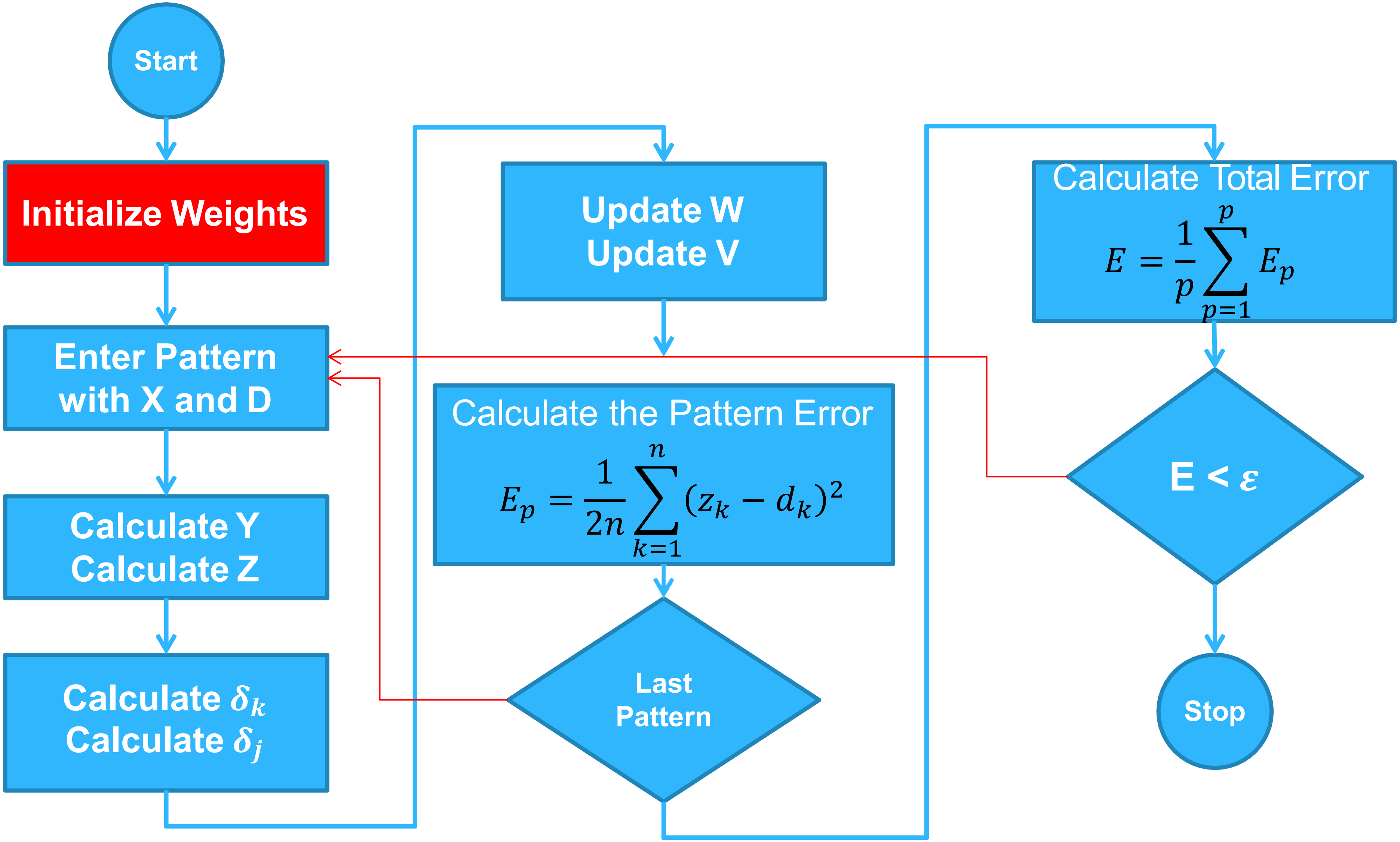
Input			Output	
x3	x2	x1	d2	d1
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Step length $\alpha = 0.1$

Activation function at hidden layer is Sigmoidal Function

Activation function at output layer is identity function

Backpropagation Algorithm



Initialize Weights

Weights are initialized randomly using normal distribution

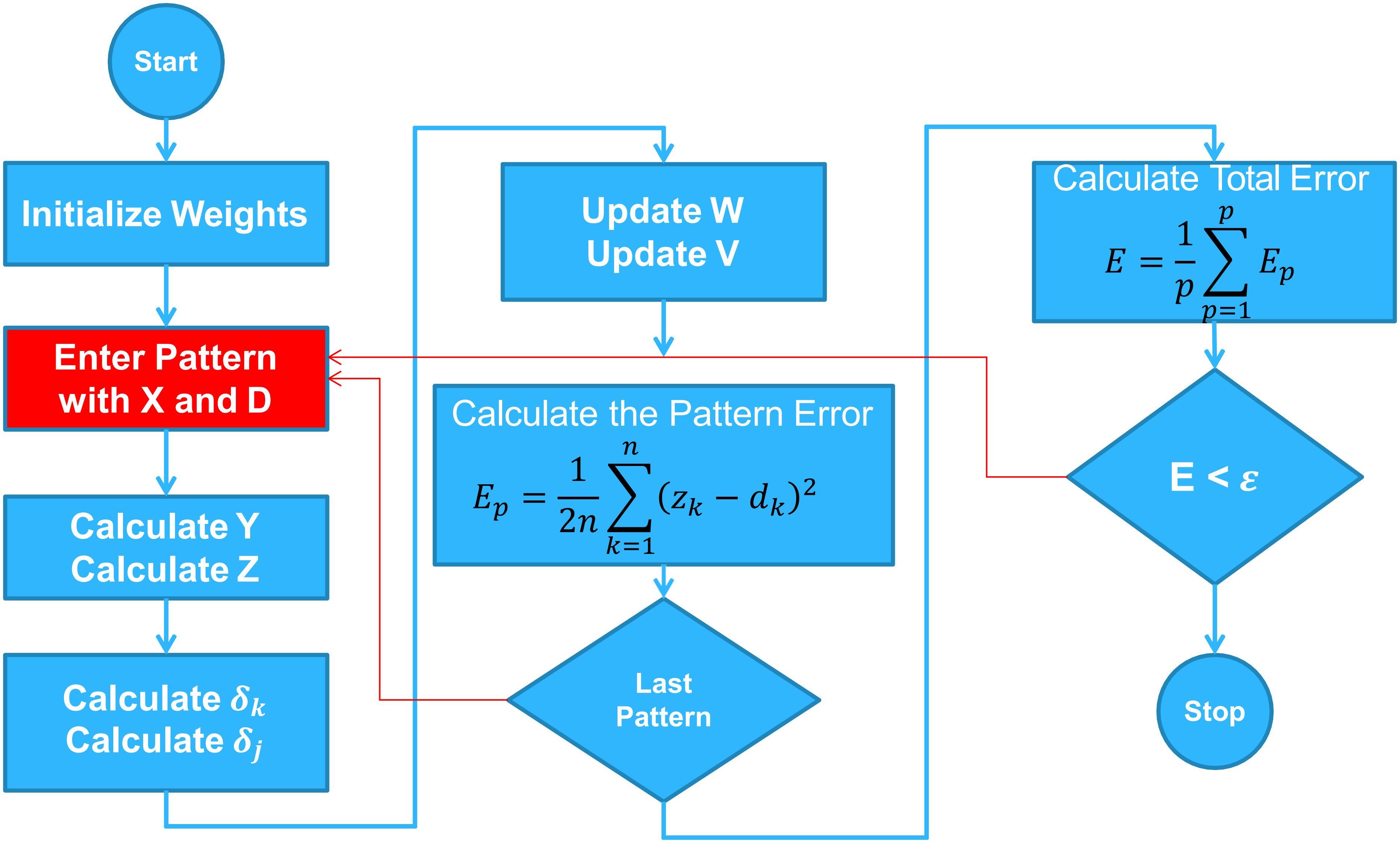
V =

-0.4677	-0.8608	-0.2339	-0.0867
-0.1249	0.7847	-1.0570	-1.4694
1.4790	0.3086	-0.2841	0.1922

W =

-0.8223	-0.2883
-0.0942	0.3501
0.3362	-1.8359
-0.9047	1.0360

Backpropagation Algorithm



Enter patterns

iteration = **1**

pattern = **1**

X =

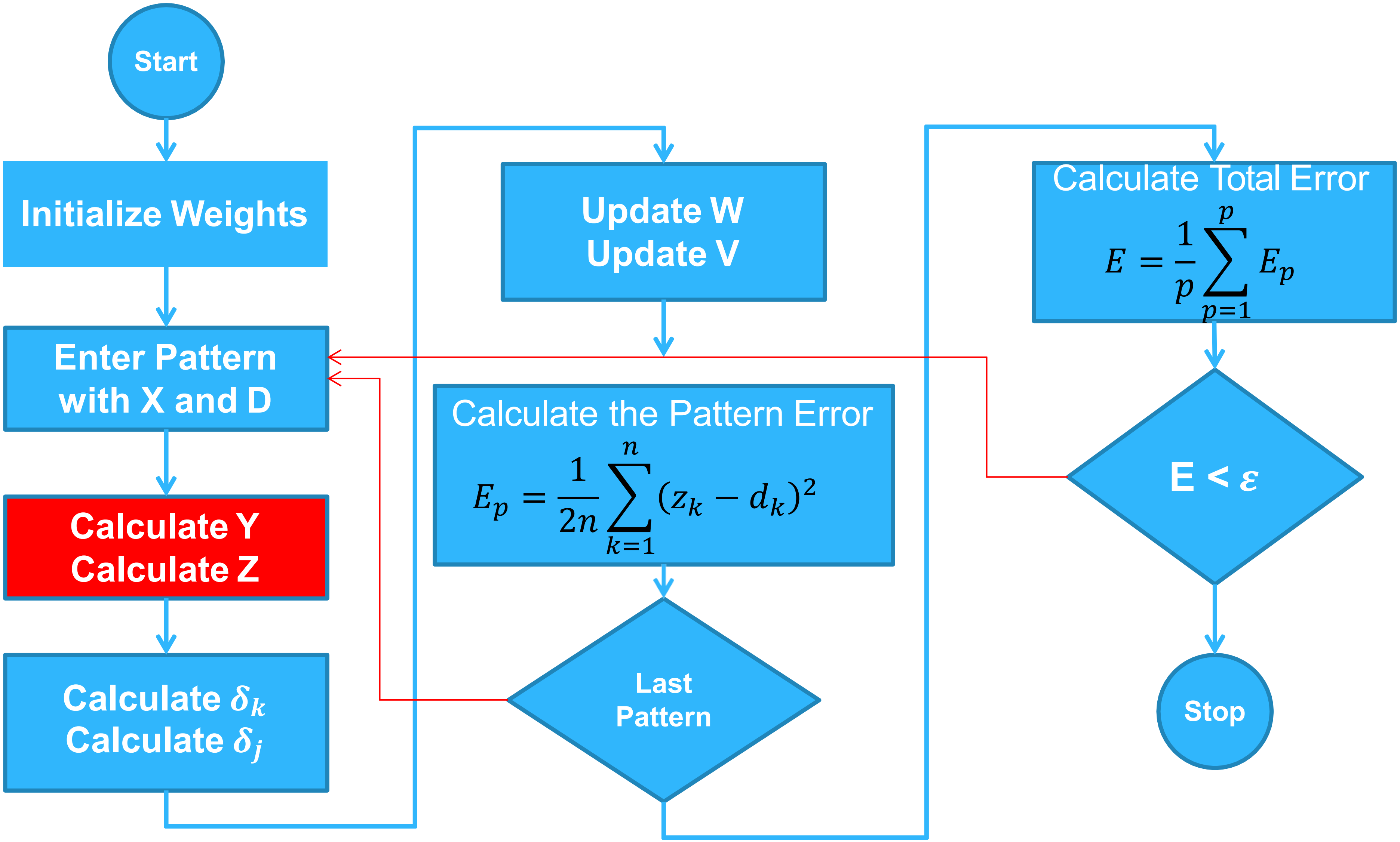
0
0
0

D =

0
1

Input x3	x2	x1	Output d2	d1
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Backpropagation Algorithm



Calculate Y and Z

$$Y = f\left(V^T X\right)$$

$$Y = f\left(\begin{bmatrix} -0.4677 & -0.1249 & 1.4790 \\ -0.8608 & 0.7847 & 0.3086 \\ -0.2339 & -1.0570 & -0.2841 \\ -0.0867 & -1.4694 & 0.1922 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right)$$

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

$$Z = f\left(W^T Y\right) = W^T Y$$

$$Z = \begin{bmatrix} -0.8223 & -0.0942 & 0.3362 & -0.9047 \\ -0.2883 & 0.3501 & -1.8359 & 1.0360 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

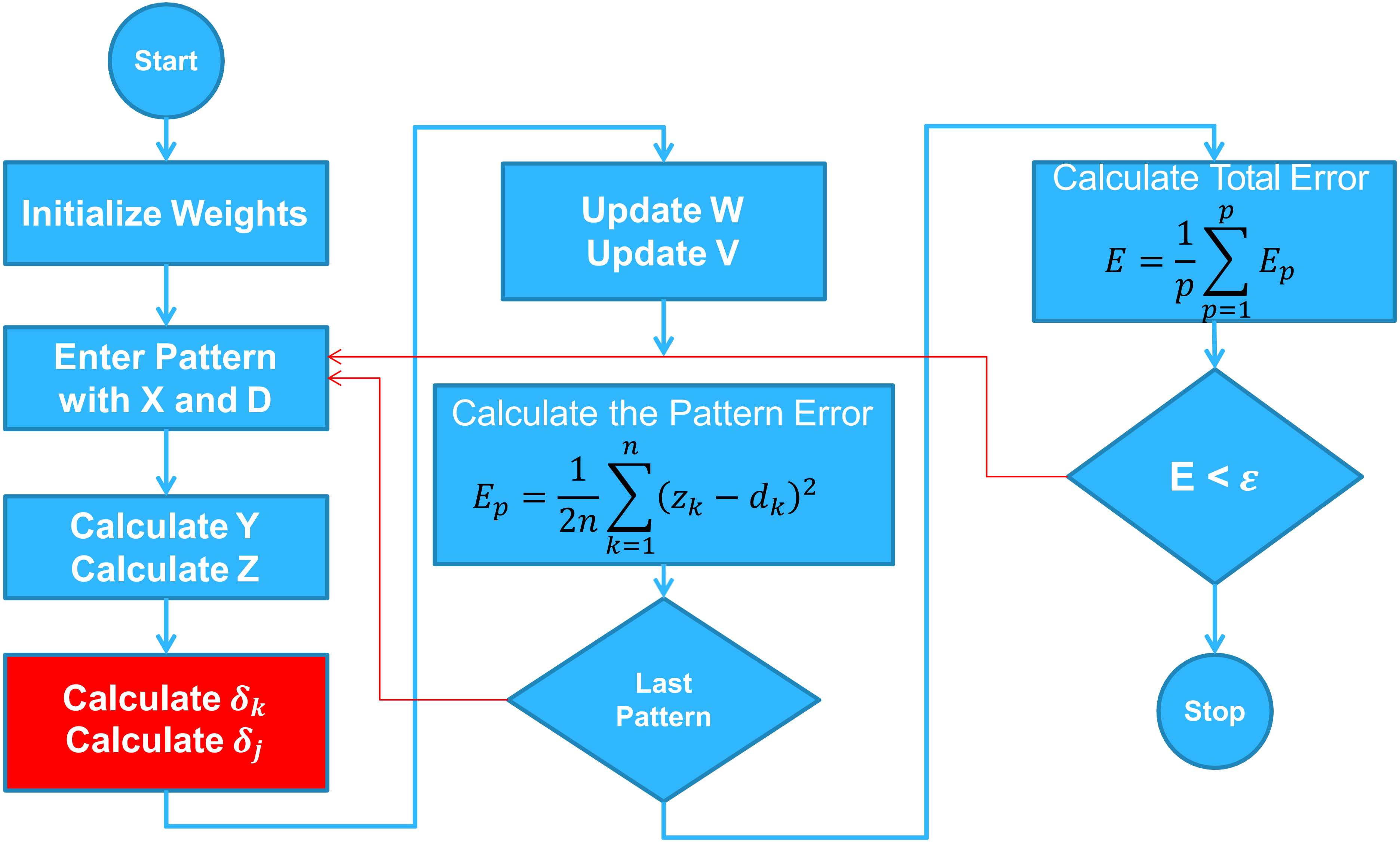
Y =

0.5000
0.5000
0.5000
0.5000

Z =

-0.7425
-0.3690

Backpropagation Algorithm



Calculate the gradient components

Scalar equation

$$\delta_k = (z_k - d_k)z'_k$$

$$z'_k = 1 \Rightarrow \delta_k = z_k - d_k$$

Vector format equation

$$\delta Z = Z - D$$

$$\delta Z = \begin{bmatrix} -0.7425 \\ -0.3690 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

dZ' =

-0.7425   -1.3690

Calculate the gradient components

Scalar equation

$$\delta_j = y'_j \sum_{k=1}^{n} \delta_k w_{jk}$$

$$y'_j = (1 - y_j) y_j$$

Vector format equation

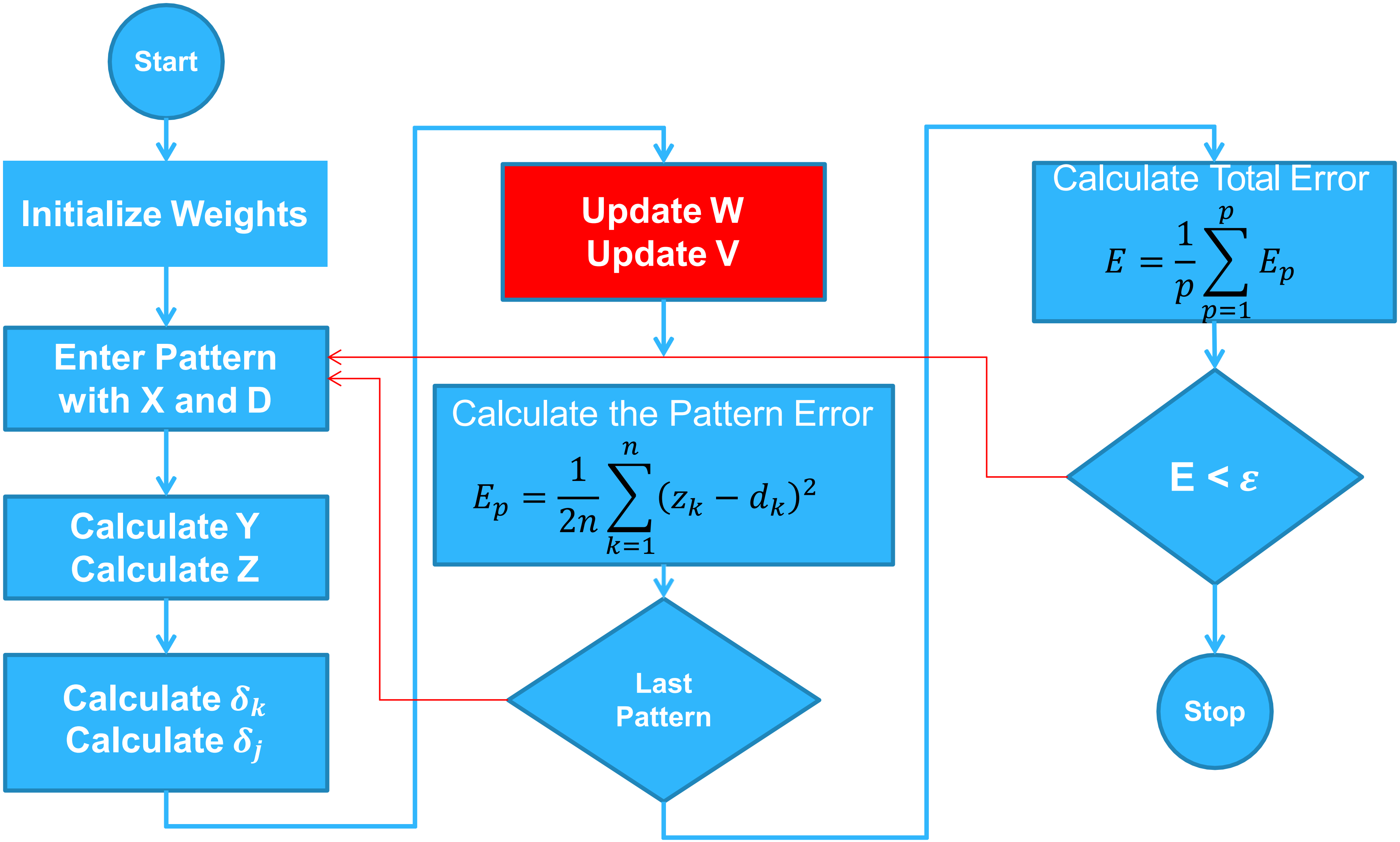
$$\delta Y = (1 - Y) . Y . W(\delta Z)$$

$$\delta Y = \left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \right) . \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} . \begin{bmatrix} -0.8223 & -0.2883 \\ -0.0942 & 0.3501 \\ 0.3362 & -1.8359 \\ -0.9047 & 1.0360 \end{bmatrix} \begin{bmatrix} -0.7425 \\ -1.3690 \end{bmatrix}$$

dY' =

0.2513 -0.1023 0.5659 -0.1866

Backpropagation Algorithm



Update Weights

Scalar equation

$$w_{jk} = w_{jk} - \alpha \delta_k y_j$$

Matrix equation

$$W = W - \alpha\, Y\, (\delta z)^T$$

$$W = \begin{bmatrix} -0.8223 & -0.2883 \\ -0.0942 & 0.3501 \\ 0.3362 & -1.8359 \\ -0.9047 & 1.0360 \end{bmatrix} - 0.1 \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \begin{bmatrix} -0.7425 & -1.3690 \end{bmatrix}$$

W =

-0.7852	-0.2198
-0.0571	0.4185
0.3733	-1.7674
-0.8675	1.1044

Update Weights

Scalar equation

$$v_{ij} = v_{ij} - \alpha \delta_j x_i$$

Matrix equation

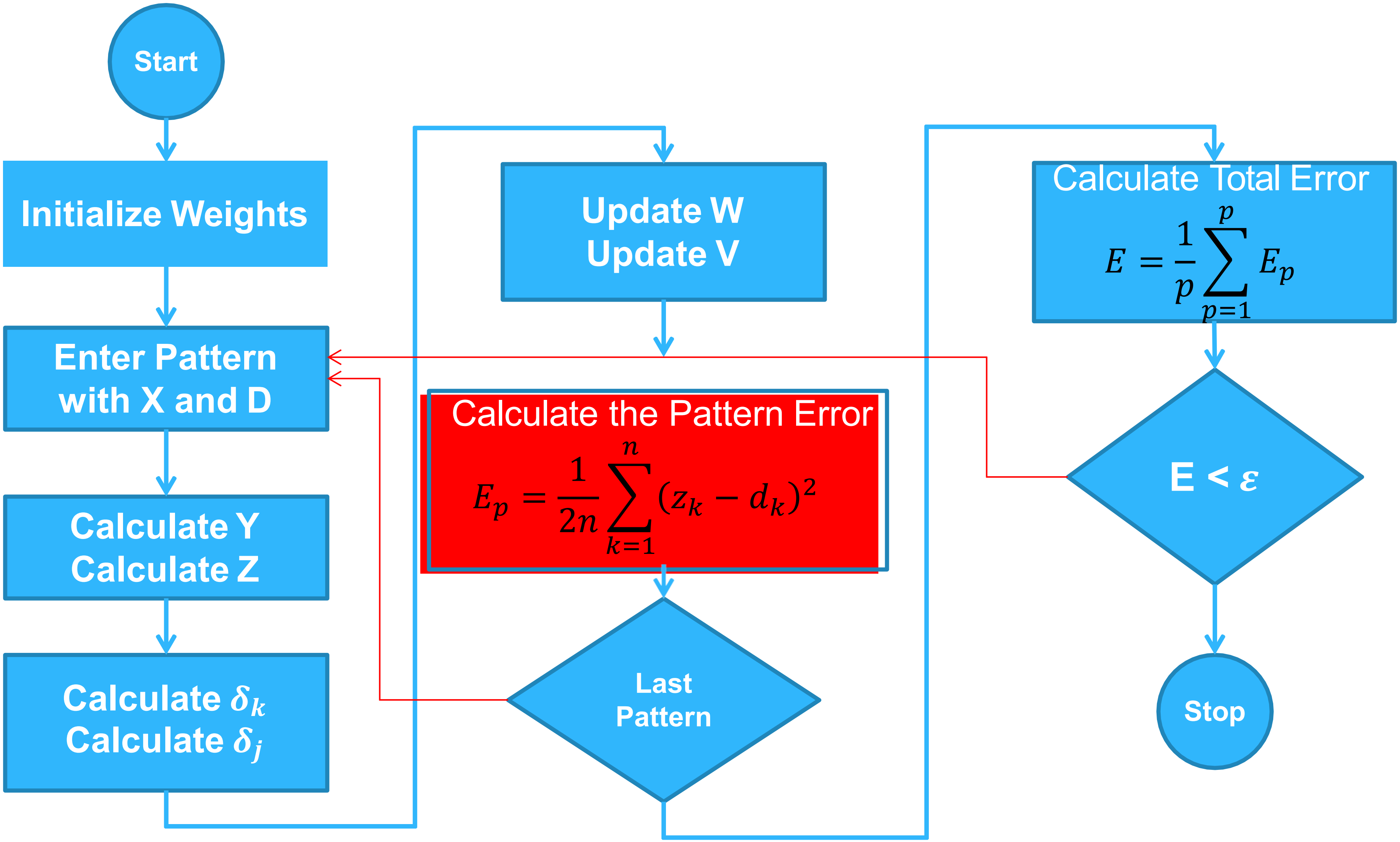
$$V = V - \alpha\, X\, (\delta Y)^T$$

$$V = \begin{bmatrix} -0.4677 & -0.8608 & -0.2339 & -0.0867 \\ -0.1249 & 0.7847 & -1.0570 & -1.4694 \\ 1.4790 & 0.3086 & -0.2841 & 0.1922 \end{bmatrix} - \mathbf{0.1} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0.2513 & -0.1023 & 0.5659 & -0.1866 \end{bmatrix}$$

V =

-0.4677	-0.8608	-0.2339	-0.0867
-0.1249	0.7847	-1.0570	-1.4694
1.4790	0.3086	-0.2841	0.1922

Backpropagation Algorithm



Calculate pattern error

$$\boldsymbol{E_1 = \frac{1}{2x2}\left((-0.7425 - 0)^2 + (-0.3690 - 1)^2\right) = 0.7787}$$

Pattern 2

Enter Pattern:

X =

0
0
1

D =

1
0

Calculate Output:

Y =

0.8144
0.5765
0.4294
0.5479

Z =

-0.9874
-0.0916

Pattern 2

Calculate Gradient Components

dZ' = -1.9874 -0.0916

dY' = 0.2389 0.0184 -0.1421 0.4020

Update Weights

W =

-0.6233	-0.2123
0.0575	0.4238
0.4587	-1.7635
-0.7586	1.1094

V =

-0.4677	-0.8608	-0.2339	-0.0867
-0.1249	0.7847	-1.0570	-1.4694
1.4551	0.3068	-0.2699	0.1520

Pattern error , E2 = 0.9853

Next patterns

- **Pattern 3**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 4**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 5**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 6**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 7**
  - Enter pattern; calculate output; calculate gradient; update weights

Pattern 8

Enter Pattern:

X =

1
1
1

D =

0
1

Calculate Output:

Y =

0.6829
0.5704
0.1858
0.1798

Z =

0.0393
-0.0055

Pattern 8

Calculate Gradient Components

dZ' = 0.0393 -1.0055

dY' = 0.0400 -0.1044 0.2699 -0.1702

Update Weights

W =

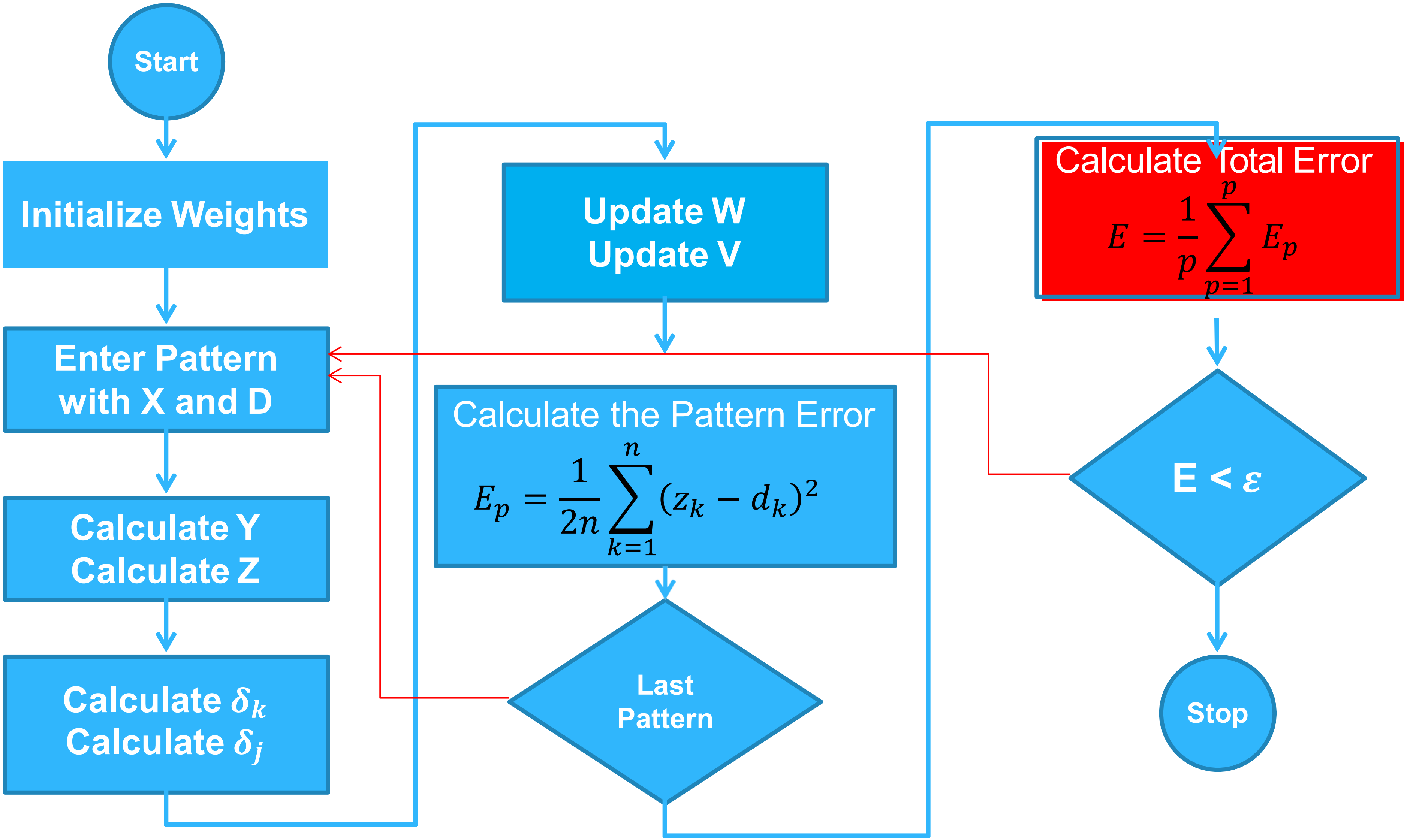
-0.2855	-0.1261
0.3743	0.4960
0.6432	-1.7301
-0.5677	1.1433

V =

-0.5034	-0.8247	-0.2321	-0.1103
-0.1686	0.8090	-1.0549	-1.4896
1.4271	0.3304	-0.2711	0.1333

Calculate pattern error E8 = 0.2531

Backpropagation Algorithm



Calculate Error for 8 patterns

$$E = (E_1 + E_2 + E_3 + E_4 + E_5 + E_6 + E_7 + E_8)/8$$

$$E = 0.095 \overset{?}{<} 0.001$$

No ➔ go to next iteration

Next Iteration

Iteration= 2

Pattern = 1

X =

0
0
0

Y =

0.5000
0.5000
0.5000
0.5000

D =

0
1

Z =

0.0821
-0.1085

Next Iteration

dZ' = 0.0821 -1.1085

dY' = 0.0291 -0.1298 0.4926 -0.3285

W =

-0.2896	-0.0707
0.3702	0.5514
0.6391	-1.6747
-0.5718	1.1988

V =

-0.5034	-0.8247	-0.2321	-0.1103
-0.1686	0.8090	-1.0549	-1.4896
1.4271	0.3304	-0.2711	0.1333

Next patterns

- **Pattern 2**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 3**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 4**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 5**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 6**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 7**
  - Enter pattern; calculate output; calculate gradient; update weights
- **Pattern 8**
  - Enter pattern; calculate output; calculate gradient; update weights

Calculate Error for 8 patterns

$$E = (E_1 + E_2 + E_3 + E_4 + E_5 + E_6 + E_7 + E_8)/8$$

?

E = < 0.001

No → go to next iteration ??

After 1500 iteration

Final Output Values

Z =

0.0038	0.9817	1.0010	1.0007	0.9820	1.0287	1.0005	-0.0014
1.0019	-0.0153	0.0012	0.0021	-0.0150	0.0263	0.0018	0.9937

Compare with Desired

D =

0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1

After 1500 iteration