Practice Problems

Book: PRINCIPLES OF SOFT COMPUTING, 2ND ED by S. N. Sivanandam, S. N. Deepa

Dr Dayal Kumar Behera

Example 1: Activation Function

3. Obtain the output of the neuron Y for the network shown in Figure 3 using activation functions as: (i) binary sigmoidal and (ii) bipolar sigmoidal.

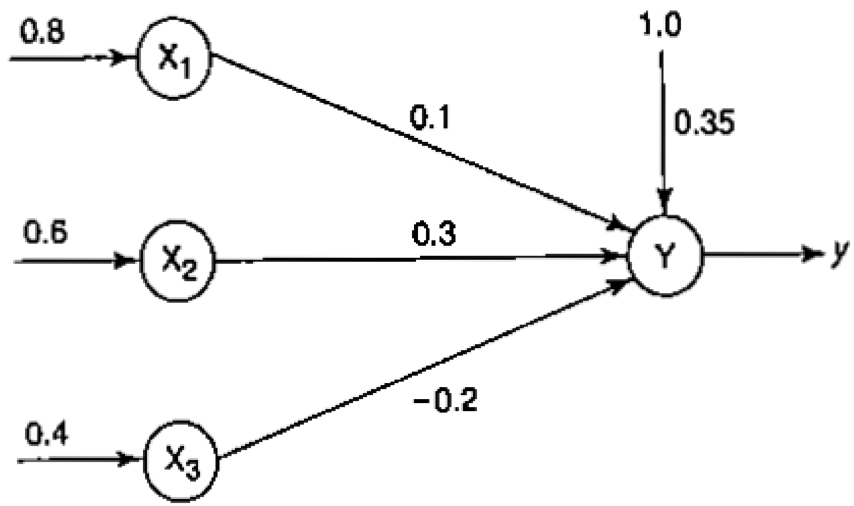


Figure 3 Neural net.

Example 1: Activation Function

Solution: The given network has three input neurons with bias and one output neuron. These form a single-layer network. The inputs are given as $[x_1, x_2, x_3] = [0.8, 0.6, 0.4]$ and the weights are $[w_1, w_2, w_3] = [0.1, 0.3, -0.2]$ with bias $b = 0.35$ (its input is always 1).

Example 1: Activation Function

The net input to the output neuron is

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

[$n = 3$, because only 3 input neurons are given]

$$= b + x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$= 0.35 + 0.8 \times 0.1 + 0.6 \times 0.3 + 0.4 \times (-0.2)$$

$$= 0.35 + 0.08 + 0.18 - 0.08 = 0.53$$

(i) For binary sigmoidal activation function,

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-0.53}} = 0.625$$

(ii) For bipolar sigmoidal activation function,

$$y = f(y_{in}) = \frac{2}{1 + e^{-y_{in}}} - 1 = \frac{2}{1 + e^{-0.53}} - 1$$

$$= 0.259$$

Example 2: Perceptron (AND Function)

- Find the new weights after epoch-1 to classify **AND** function with bipolar input and targets using **perceptron** learning algorithm/rule.
- Set initial weight w1=w2=b=0. learning rate = 1 and threshold = 0.

Architecture

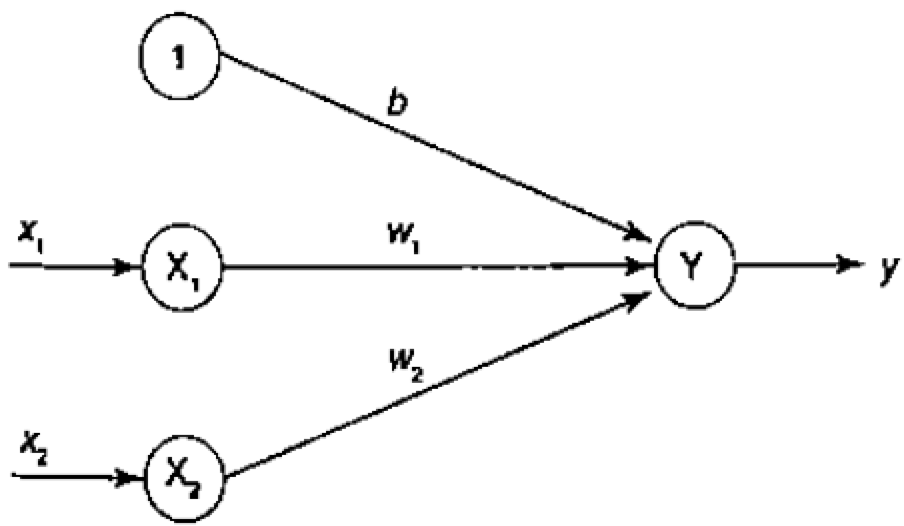


Figure 1 Perceptron network for AND function.

First training sample

For the first training sample, x1 = 1, *X2* = 1 and *t* = 1, with weights and bias, *w1* = 0, *W2* = 0 and *b=0. learning rate α = 1.*

- Calculate the net input

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 0 + 1 \times 0 + 1 \times 0 = 0$$

- The output y is computed by applying activations over the net input calculated:

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

Here we have taken $\theta = 0$. Hence, when, $y_{in} = 0$, $y = 0$.

- Check whether $t = y$. Here, $t = 1$ and $y = 0$, so $t \neq y$, hence weight updation takes place:

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times 1 = 1$$

Here, the change in weights are

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta b = \alpha t$$

Weight and Bias updates

Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	−1	1	−1	1	1	−1	1	−1	0	2	0
−1	1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2											
1	1	1	1	1	1	0	0	0	1	1	−1
1	−1	1	−1	−1	−1	0	0	0	1	1	−1
−1	1	1	−1	−1	−1	0	0	0	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1

Example 3: Perceptron (OR Function)

- Find the new weights after epoch-1 to classify **OR** function with bipolar input and targets using **perceptron** learning algorithm/rule.
- Set initial weight w1=w2=b=0. learning rate = 1 and threshold = 0.

Example 3: Perceptron (OR Function)

Example 4: Perceptron

- Find weights required to perform the following classification using perceptron network. Assume learning rate α as 1 and initial weights as 0 and threshold θ = 0.2.

Input					
x_1	x_2	x_3	x_4	b	Target (t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1

Activation

Since the threshold θ = 0.2, so the activation function is

$$y = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \\ -1 & \text{if } y_{in} < -0.2 \end{cases}$$

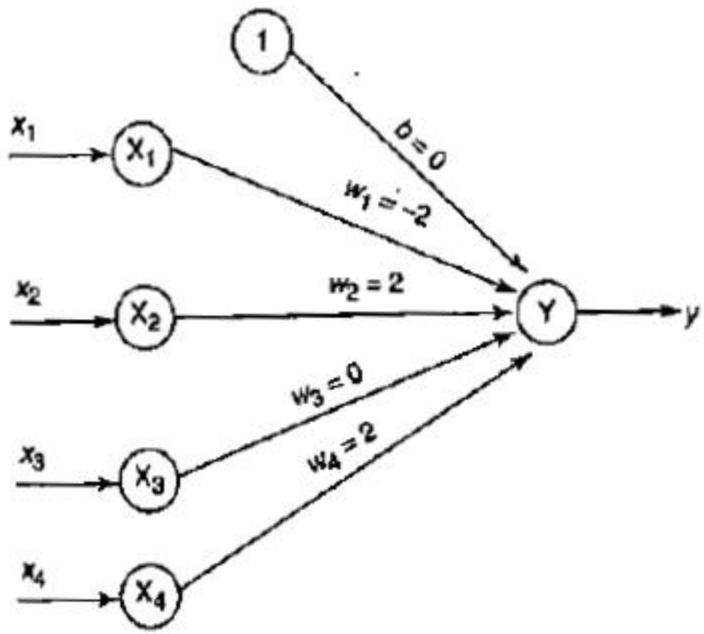
The net input is given by

$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

Weight updates during Training

Inputs					Target	Net input	Output	Weight changes					Weights				
													(w_1	w_2	w_3	w_4	b)
(x_1	x_2	x_3	x_4	b)	(t)	(y_{in})	(y)	(Δw_1	Δw_2	Δw_3	Δw_4	Δb)	(0	0	0	0	0)
EPOCH-1																	
(1	1	1	1	1)	1	0	0	1	1	1	1	1	1	1	1	1	1
(−1	1	−1	−1	1)	1	−1	−1	−1	1	−1	−1	1	0	2	0	0	2
(1	1	1	−1	1)	−1	4	1	−1	−1	−1	1	−1	−1	1	−1	1	1
(1	−1	−1	1	1)	−1	1	1	−1	1	1	−1	−1	−2	2	0	0	0
EPOCH-2																	
(1	1	1	1	1)	1	0	0	1	1	1	1	1	−1	3	1	1	1
(−1	1	−1	−1	1)	1	3	1	0	0	0	0	0	−1	3	1	1	1
(1	1	1	−1	1)	−1	4	1	−1	−1	−1	1	−1	−2	2	0	2	0
(1	−1	−1	1	1)	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
EPOCH-3																	
(1	1	1	1	1)	1	2	1	0	0	0	0	0	−2	2	0	2	0
(−1	1	−1	−1	1)	1	2	1	0	0	0	0	0	−2	2	0	2	0
(1	1	1	−1	1)	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
(1	−1	−1	1	1)	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0

Model for Testing

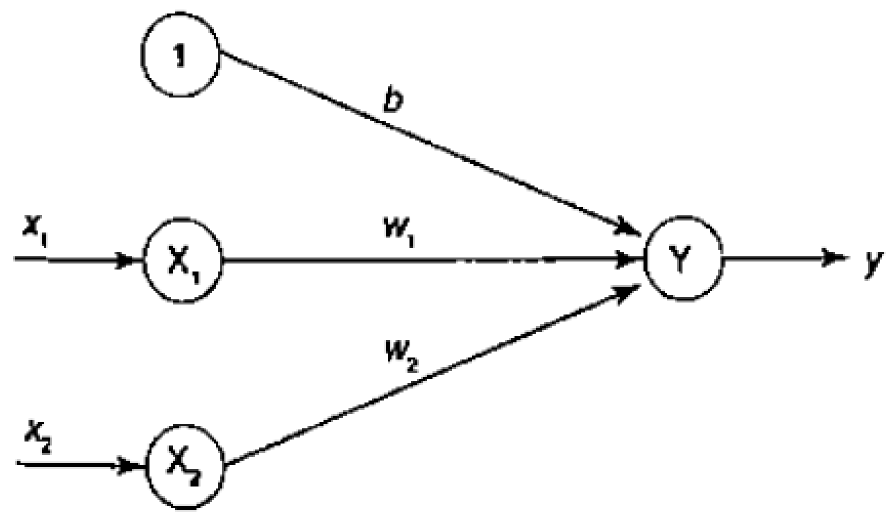


Example 5: ADALINE

- Find the new weights and total squared error after epoch-1 to classify **OR** function with bipolar input and targets using **ADALINE** network.
- Set initial weight w1=w2=b=0.1. learning rate = 0.1.

x_1	x_2	1	t
1	1	1	1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

Architecture



ADALINE: OR

For the first input sample, X1 = 1, X2 = 1, t = 1,we calculate the net input as

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i = b + \sum_{i=1}^{2} x_i w_i$$

$$= b + x_1 w_1 + x_2 w_2$$

$$= 0.1 + 1 \times 0.1 + 1 \times 0.1 = 0.3$$

Weight Updates

$$(t - y_{in}) = (1 - 0.3) = 0.7$$

$$\Delta w_1 = \alpha(t - y_{in})x_1$$
$$\Delta w_2 = \alpha(t - y_{in})x_2$$
$$\Delta b = \alpha(t - y_{in})$$

$$w_i(\text{new}) = w_i(\text{old}) + \alpha(t - y_{in})x_i$$

$$\begin{aligned} w_1(\text{new}) &= w_1(\text{old}) + \Delta w_1 = 0.1 + 0.1 \times 0.7 \times 1 \\ &= 0.1 + 0.07 = 0.17 \\ w_2(\text{new}) &= w_2(\text{old}) + \Delta w_2 = 0.1 \\ &\quad + 0.1 \times 0.7 \times 1 = 0.17 \\ b(\text{new}) &= b(\text{old}) + \Delta b = 0.1 + 0.1 \times 0.7 = 0.17 \end{aligned}$$

Training

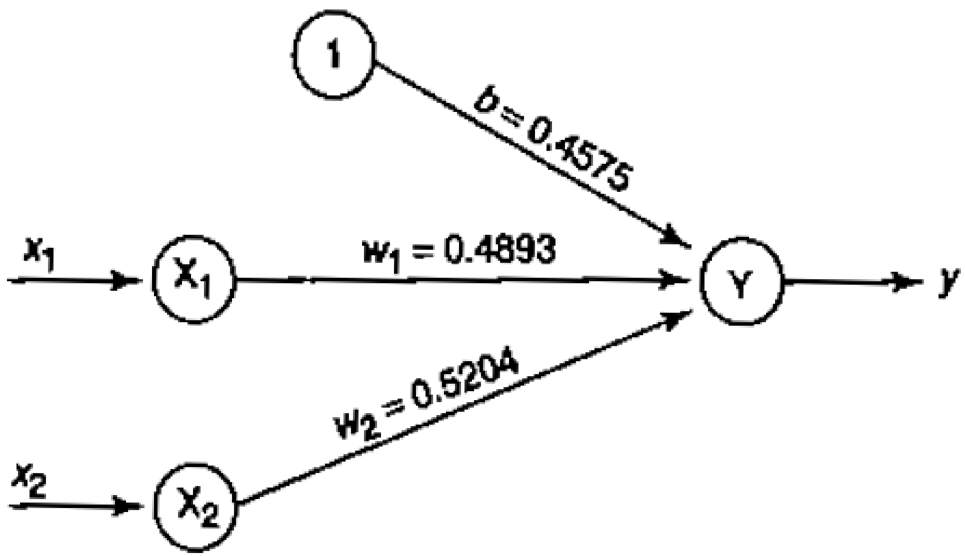
Inputs x_1	x_2	1	Target t	Net input y_{in}	$(t - y_{in})$	Weight changes Δw_1	Δw_2	Δb	Weights w_1 (0.1	w_2 0.1	b 0.1)	Error $(t - y_{in})^2$
EPOCH-1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	−1	1	1	0.17	0.83	0.083	−0.083	0.083	0.253	0.087	0.253	0.69
−1	1	1	1	0.087	0.913	−0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
−1	−1	1	−1	0.0043	−1.0043	0.1004	0.1004	−0.1004	0.2621	0.2787	0.2439	1.01
EPOCH-2												
1	1	1	1	0.7847	0.2153	0.0215	0.0215	0.0215	0.2837	0.3003	0.2654	0.046
1	−1	1	1	0.2488	0.7512	0.7512	−0.0751	0.0751	0.3588	0.2251	0.3405	0.564
−1	1	1	1	0.2069	0.7931	−0.7931	0.0793	0.0793	0.2795	0.3044	0.4198	0.629
−1	−1	1	−1	−0.1641	−0.8359	0.0836	0.0836	−0.0836	0.3631	0.388	0.336	0.699
EPOCH-3												
1	1	1	1	1.0873	−0.0873	−0.087	−0.087	−0.087	0.3543	0.3793	0.3275	0.0076
1	−1	1	1	0.3025	+0.6975	0.0697	−0.0697	0.0697	0.4241	0.3096	0.3973	0.487
−1	1	1	1	0.2827	0.7173	−0.0717	0.0717	0.0717	0.3523	0.3813	0.469	0.515
−1	−1	1	−1	−0.2647	−0.7353	0.0735	0.0735	−0.0735	0.4259	0.4548	0.3954	0.541
EPOCH-4												
1	1	1	1	1.2761	−0.2761	−0.0276	−0.0276	−0.0276	0.3983	0.4272	0.3678	0.076
1	−1	1	1	0.3389	0.6611	0.0661	−0.0661	0.0661	0.4644	0.3611	0.4339	0.437
−1	1	1	1	0.3307	0.6693	−0.0669	0.0669	0.0699	0.3974	0.428	0.5009	0.448
−1	−1	1	−1	−0.3246	−0.6754	0.0675	0.0675	−0.0675	0.465	0.4956	0.4333	0.456
EPOCH-5												
1	1	1	1	1.3939	−0.3939	−0.0394	−0.0394	−0.0394	0.4256	0.4562	0.393	0.155
1	−1	1	1	0.3634	0.6366	0.0637	−0.0637	0.0637	0.4893	0.3925	0.457	0.405
−1	1	1	1	0.3609	0.6391	−0.0639	0.0639	0.0639	0.4253	0.4654	0.5215	0.408
−1	−1	1	−1	−0.3603	−0.6397	0.064	0.064	−0.064	0.4893	0.5204	0.4575	0.409

Total Squared Error

Epoch 1	3.02
Epoch 2	1.938
Epoch 3	1.5506
Epoch 4	1.417
Epoch 5	1.377

It can be noticed that as training goes on, the error value gets minimized. Hence, training may be continued for further minimization of error.

Model for Testing



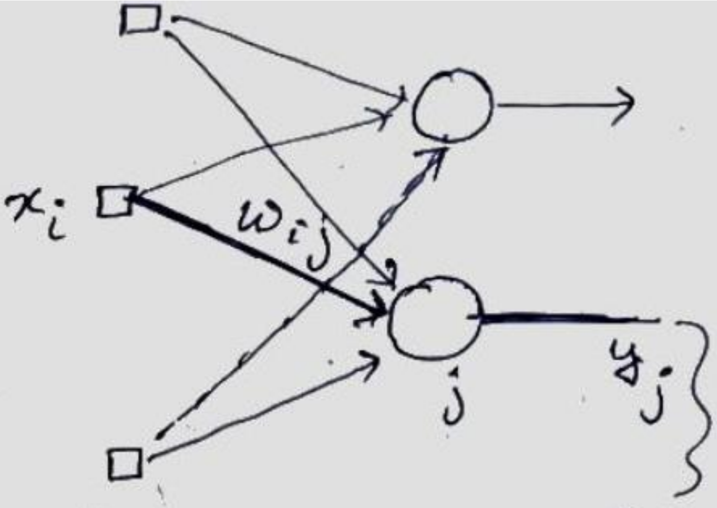
MADALINE

Delta Rule

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$

$$\Delta w_{ij} = \eta \, e_j \, x_i$$

$$e_j = t - y_{in_j}$$

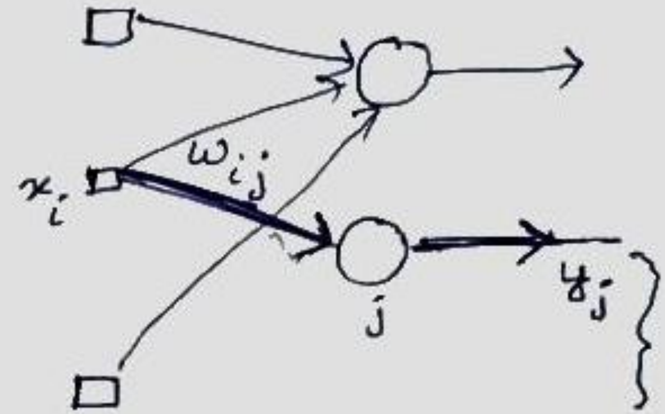


$$e_j = t_j - y_j$$

$y_j = y_{in_j}$ (Linear Activation function)

Generalized Delta Rule

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$$



$$e_j = t_j - y_j$$

$$\Delta w_{ij} = \eta\, \delta_j\, x_i$$

$$\delta_j = e_j\, \phi'(y_{in_j})$$

$$\phi'(y_{in_j}) = \frac{d}{dx}(\text{Activation function})$$

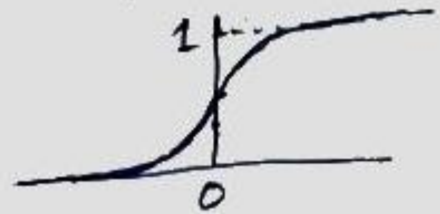
Generalized Delta Rule

In case of sigmoid or logistic activation (Binary)

$$\phi(x) = \frac{1}{1+e^{-\lambda x}}$$ where λ = steepness parameter

let $\lambda = 1$,

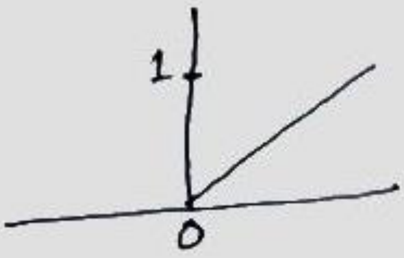
$$\phi(x) = \frac{1}{1+e^{-x}}$$



$$\phi'(x) = \phi(x)\cdot(1-\phi(x))$$

Generalized Delta Rule

In case of linear activation



$$\phi(x) = x$$

$$\phi'(x) = \frac{d}{dx}(x) = 1$$

$$\delta_j = e_j \, \phi'(x) = e_j \cdot 1 = e_j \qquad \boxed{\delta_j = e_j}$$

$$\boxed{w_{ij} = w_{ij} + \eta \, e_j \, x_i}$$

Note# When we use linear activation function, the generalized delta rule becomes simple delta rule.

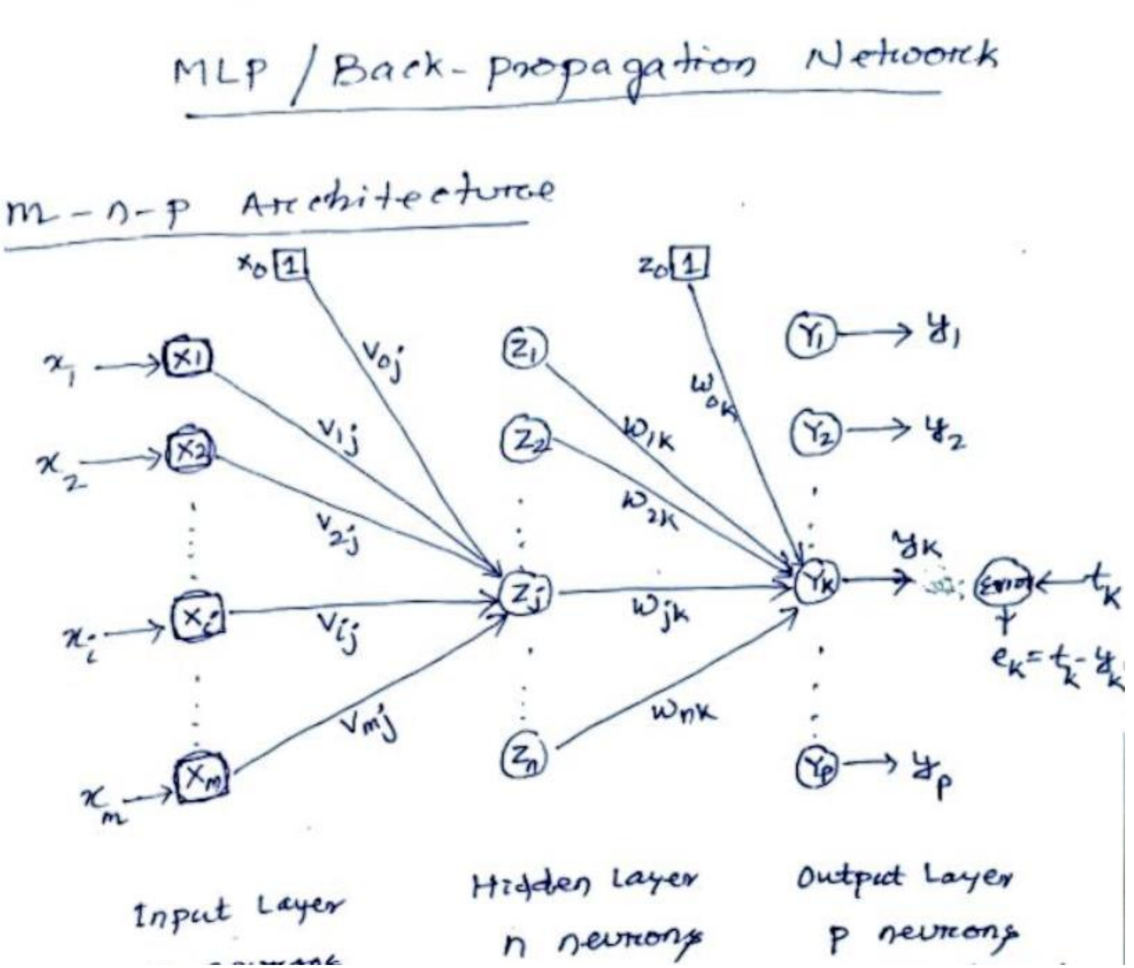
Delta Rule vs. Generalized Delta Rule

Note#

Delta Rule – uses linear activation function

Generalized Delta Rule – uses sigmoid activation

Back Propagation NN



Forward Pass

m − n − p

Forward pass

Input i, Hidden j, output k

Input to Hidden Unit Z_j $(j = 1 \text{ to } n)$

$$Z_{inj} = v_{oj} + \sum_{i=1}^{m} x_i \cdot v_{ij}$$

$$= \sum_{i=0}^{m} x_i \cdot v_{ij} = V^T X = V \cdot X$$

dot product

Output of the Hidden unit Z_j

Apply activation function over Z_{inj}

$$Z_j = f(Z_{inj})$$

$f()$: Activation function.

Forward Pass

Input to output unit Y_k ($k = 1$ to p)

$$y_{ink} = w_{0k} + \sum_{j=1}^{n} z_j \cdot w_{jk}$$

$$= \sum_{j=0}^{n} z_j \, w_{jk} = W^T Z = W \cdot Z$$

dot product

Output of output unit/Layer

Apply activation function.

$$y_k = f(y_{ink})$$

Error

Error associated with output unit Y_k

$$e_k = t_k - y_k$$

t_k: target

y_k: computed output

Squared Error

$$e_k^2 = (t_k - y_k)^2$$

- squared error is minimized by the use of steepest descent / Gradient descent method.

Error

- For easy mathematical derivation, error of k^{th} output neuron can be written as

$$E_k = \frac{1}{2}(t_k - y_k)^2$$

- For one training sample, error associated with output layer

$$E = \sum_{k=1}^{P} E_k = \sum_{k=1}^{P} \frac{1}{2}(t_k - y_k)^2$$

- For "T" training samples, total error in the prediction

$$E_{tot} = \sum_{t=1}^{T} \sum_{k=1}^{P} E_k = \sum_{l=1}^{L} \sum_{k=1}^{P} \frac{1}{2}(t_k - y_k)^2$$

Backward Pass (Back Propagation of Error)

Local Gradient / Error correction term / Back-propagated error from the output unit Y_k ($k = 1$ to p)

output layer

$$\delta_k^{(o)} = e_k\, f'(y_{in\,k})$$

$$= (t_k - y_k)\, f'(y_{in\,k})$$

derivative of activation function

change in weights and bias as per the Generalized Delta Rule (steepest descent method)

$$w_{jk}^{new} = w_{jk}^{old} + \Delta w_{jk}$$

where $w_{jk} = \eta\, \delta_k z_j$

This δ_k ~~is pr~~ of output layer is propagated to each hidden unit. (backwards)

Backward Pass

Weighted sum of δ at each hidden unit z_j ($j=1$ to n) from the output units Y_k ($k=1$ to p)

$$\delta_{inj} = \sum_{k=1}^{p} \delta_k w_{jk}$$

Local Gradient / propagated Error from hidden unit z_j

$$\overbrace{\delta_j^{(h)}}^{\text{hidden layer}} = \delta_{inj} \cdot f'(z_{inj})$$

Weight updates.

$$v_{ij}^{new} = v_{ij}^{old} + \Delta v_{ij}$$

$$\Delta v_{ij} = \eta \, \delta_j \, x_i$$

Backward Pass

Note # Modified Generalized Delta Rule with momentum constant

$$w_{jk}^{(t+1)} = w_{jk}^{(t)} + \alpha \, \Delta w_{jk}^{(t-1)} + \Delta w_{jk}^{(t)}$$

$$\Delta w_{jk} = \eta \, \delta_k \, z_j$$

α: momentum term
η: learning rate
t: time stamp.

$\eta \uparrow$, higher rate of learning, But unstable. (oscillatory)

$\eta \downarrow$, slower rate of learning, But stable.

$\eta \uparrow$ with α, higher learning + stable. (Quick convergence) + (less oscillation)

Backward Pass

Hidden Layer weight updates

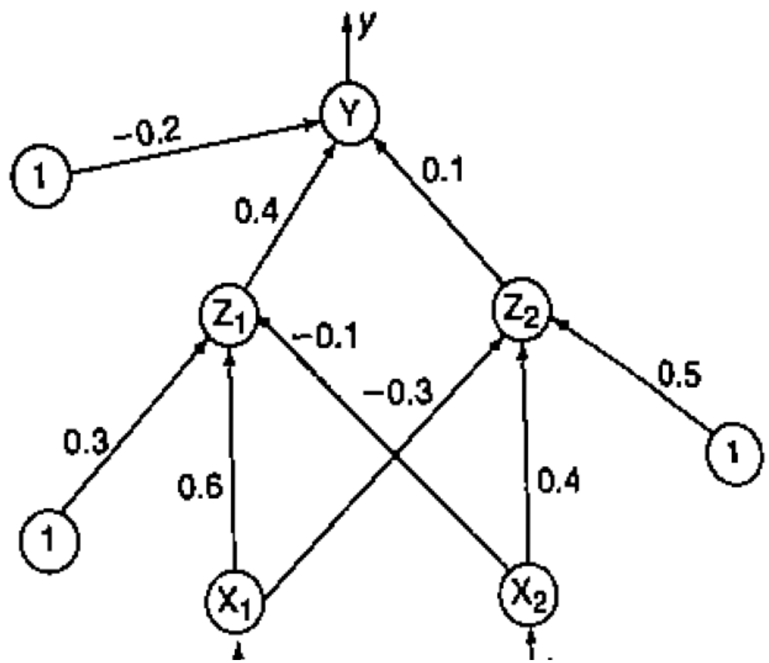
$$v_{ij}^{(t+1)} = v_{ij}^{(t)} + \alpha \Delta v_{ij}^{(t-1)} + \Delta v_{ij}^{(t)}$$

$$\Delta v_{ij}^{(t)} = \eta \, \delta_j^{(h)} x_i$$

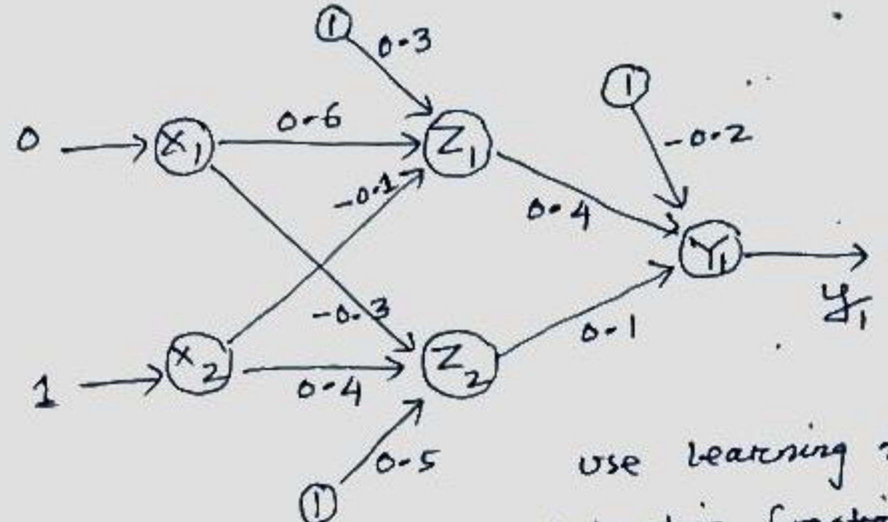
hidden Layer
local gradient

Practice Problem

Using back-propagation network, find the new weights for the given NN. It is presented with the input pattern [0, 1] and the target output is 1. Use a learning rate = 0.25 and binary sigmoidal activation function.



Practice Problem



$t = 1$

use learning rate $\eta = 0.25$

Activation function $f(x)$

$= $ Binary Sigmoid

$= \dfrac{1}{1+e^{-x}}$

Slope parameter $\lambda = 1$.

Momentum $\alpha = 0$.

Given the training sample

$[x_1\ x_2] = [0, 1]$

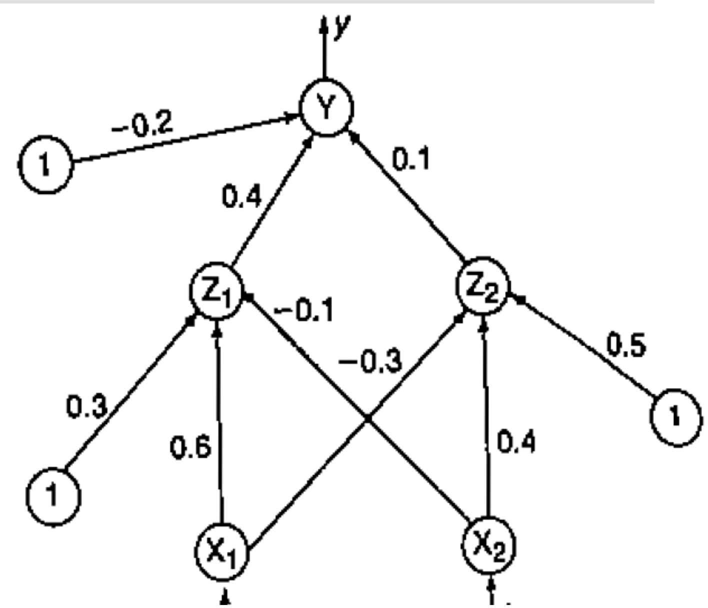
target $= t = 1$

Forward Pass (Hidden Layer Calculation)

Hidden Layer input

$$z_{in1} = 1\cdot v_{01} + x_1\cdot v_{11} + x_2\cdot v_{21}$$
$$= 1\cdot 0.3 + 0\cdot 0.6 + 1\times -0.1 = 0.2$$

$$z_{in2} = 1\cdot v_{02} + x_1\cdot v_{12} + x_2\cdot v_{22}$$
$$= 1\cdot 0.5 + 0\times -0.3 + 1\times 0.4 = 0.9$$



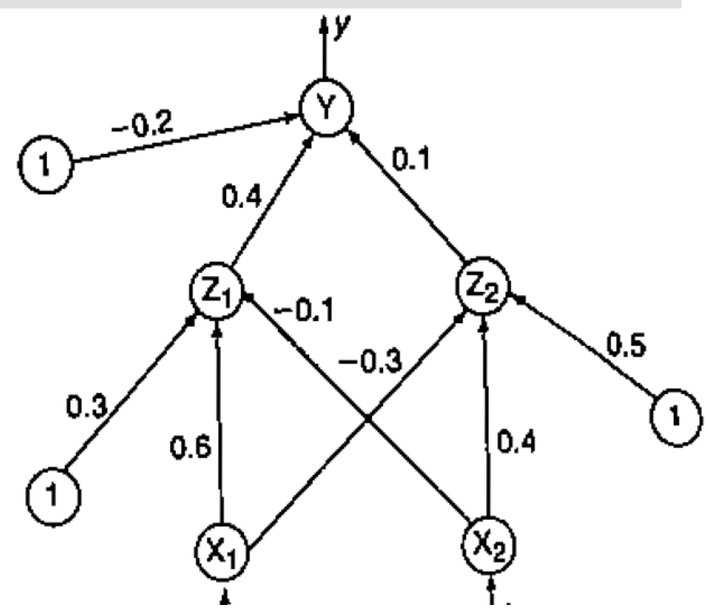
Forward Pass (Hidden Layer Calculation)

Hidden Layer output

Applying activation function to calculate the output of hidden layer.

$$Z_1 = f(Z_{in1}) = \frac{1}{1+e^{-Z_{in1}}} = \frac{1}{1+e^{-0.2}} = 0.5498$$

$$Z_2 = f(Z_{in2}) = \frac{1}{1+e^{-Z_{in2}}} = \frac{1}{1+e^{-0.9}} = 0.7109$$



Forward Pass (Output Layer Calculation)

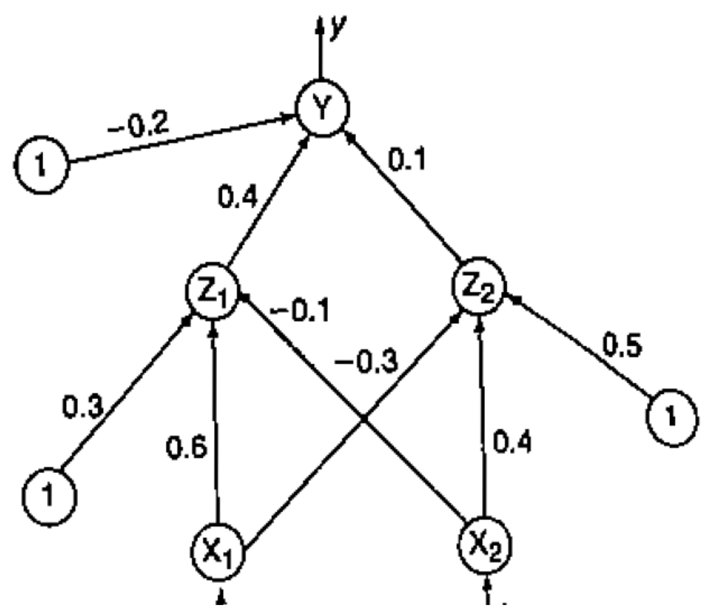
output layer input

$$y_{in1} = 1 \cdot w_{01} + z_1 \cdot w_{11} + z_2 w_{21}$$
$$= 1 \cdot -0.2 + 0.5498 \cdot 0.4 + 0.7109 * 0.1 = 0.091$$

output layer output

$$y_1 = f(y_{in1}) = \frac{1}{1+e^{-y_{in1}}} = \frac{1}{1+e^{-0.091}} = 0.5227$$

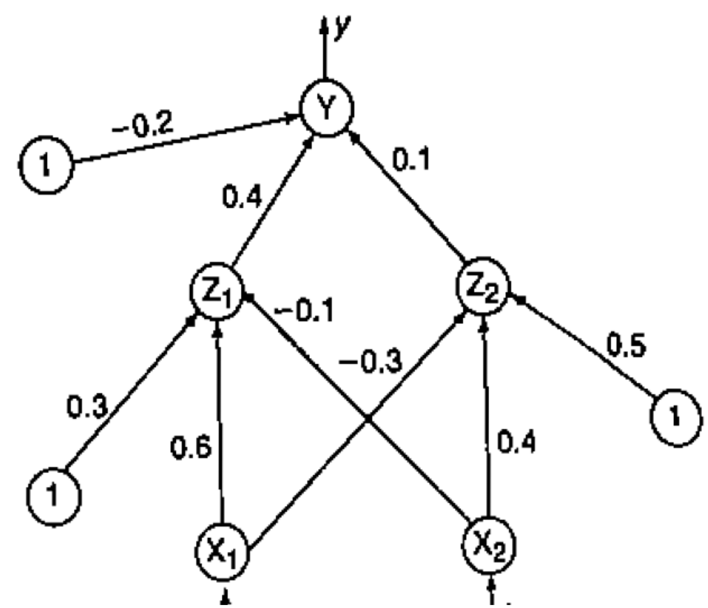
Squared Error : $(1 - 0.5227)^2$



Backward Pass (Output Layer Calculation)

Local Gradient / Error correction from output layer

$$\delta_1 = (t_1 - y_1)\, f'(y_{in})$$
$$= (1 - 0.5227)\; y_1 (1 - y_1)$$
$$= (1 - 0.5227)\; 0.5227\, (1 - 0.5227)$$
$$= 0.1191$$



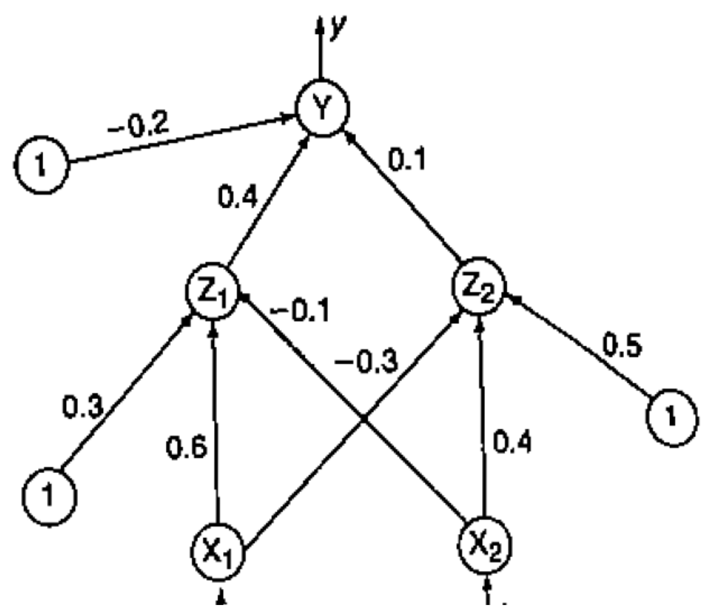
Backward Pass (Output Layer Calculation)

weight correction between hidden and output layer

$$\Delta w_{01} = \eta \cdot \delta_1 \cdot 1 = 0.25 * 0.1191 * 1 = 0.02978$$

$$\Delta w_{11} = \eta \cdot \delta_1 z_1 = 0.25 * 0.1191 * 0.5498 = 0.0164$$

$$\Delta w_{21} = \eta \cdot \delta_1 \cdot z_2 = 0.25 * 0.1191 * 0.7109 = 0.02117$$



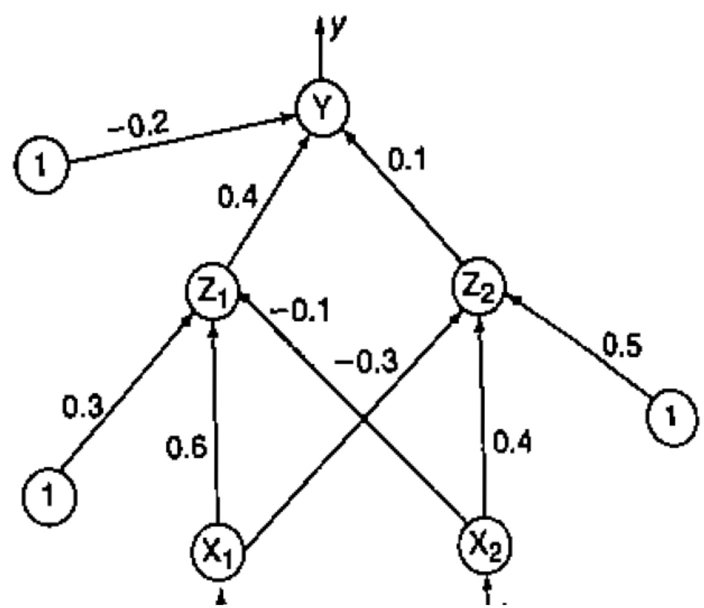
Output Layer Updated Weight

updated weight

$$w_{01}(\text{new}) = W_{01}(\text{old}) + \Delta w_{01} = -0.2 + 0.0298\cancel{7}8$$
$$= -0.17022$$

$$w_{11}(\text{new}) = w_{11}(\text{old}) + \Delta w_{11} = 0.4 + 0.0164 = 0.4164$$

$$w_{21}(\text{new}) = W_{21}(\text{old}) + \Delta w_{21} = 0.1 + 0.02117 = 0.12117$$

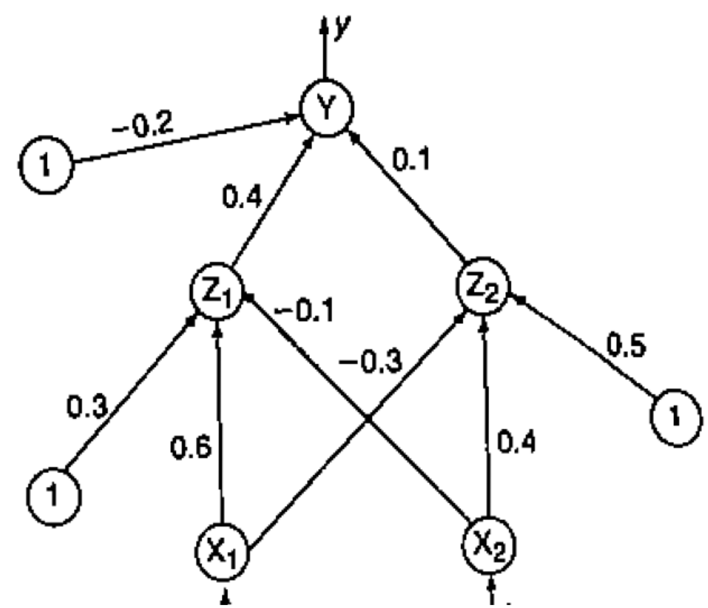


Backward Pass (Hidden Layer Calculation)

Local Gradient / Error propagation from hidden layer (j = 1 to 2)

$$\delta_j = \delta_{inj} f'(z_{inj})$$

$$\delta_{inj} = \sum_{k=1}^{p} \delta_k \cdot w_{jk} = \delta_1 \cdot w_{j1}$$



Backward Pass (Hidden Layer Calculation)

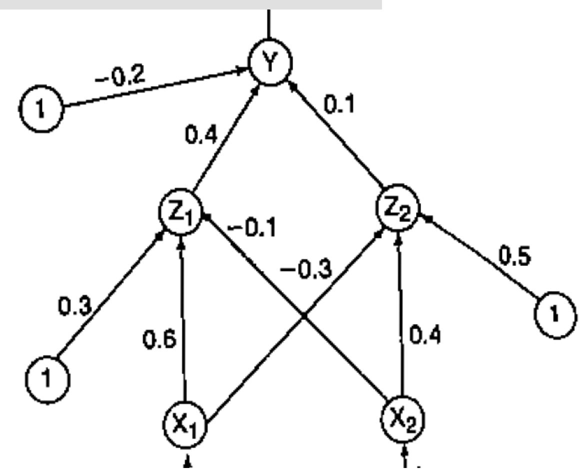
Local Gradient/ Error propagation from hidden layer

$$\delta_{in1} = \delta_1 \cdot w_{11} = 0.1191 \times 0.4 = 0.04764$$

$$\delta_{in2} = \delta_1 \cdot w_{21} = 0.1191 \times 0.1 = 0.01191$$

$$\delta_1 = \delta_{in1} \cdot f'(z_{in1}) = \delta_{in1} \cdot z_1 (1 - z_1)$$
$$= 0.04764 * 0.5496 (1 - 0.5498)$$
$$= \cancel{0.2475}\ 0.0118$$

$$\delta_2 = \delta_{in2} \cdot f'(z_{in2}) = \delta_{in2} \cdot z_2 (1 - z_2)$$
$$= 0.1191 * 0.7109 (1 - 0.7109)$$
$$= 0.00245$$



Backward Pass (Output Layer Calculation)

weight correction between input and hidden layer

$$\Delta v_{01} = \eta\, \delta_1 \cdot 1 = 0.25 * 0.0118 = 0.00295$$

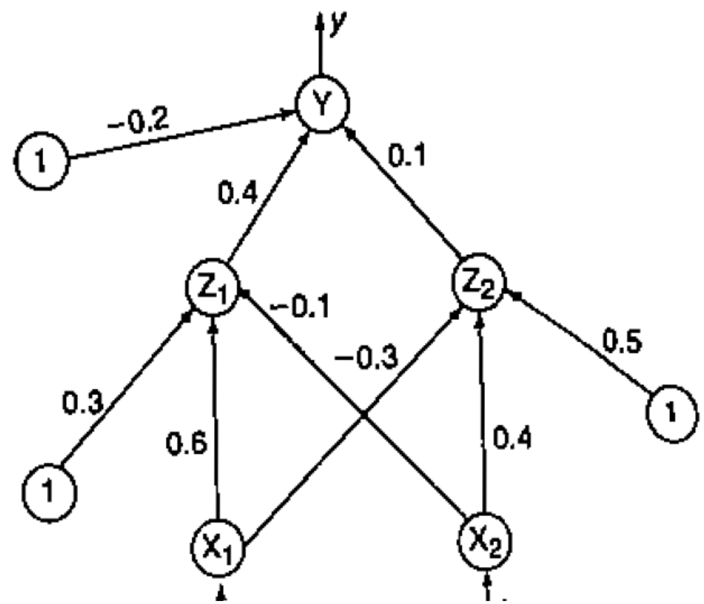
$$\Delta v_{11} = \eta\, \delta_1 \cdot x_1 = 0.25 * 0.0118 * 0 = 0$$

$$\Delta v_{21} = \eta\, \delta_1 \cdot x_2 = 0.25 * 0.0118 * 1 = 0.00295$$

$$\Delta v_{02} = \eta\, \delta_2 \cdot 1 = 0.25 * 0.00245 * 1 = 0.0006125$$

$$\Delta v_{12} = \eta\, \delta_2 \cdot x_1 = 0.25 * 0.00245 * 0 = 0$$

$$\Delta v_{22} = \eta\, \delta_2 \cdot x_2 = 0.25 * 0.00245 * 1 = 0.0006125$$



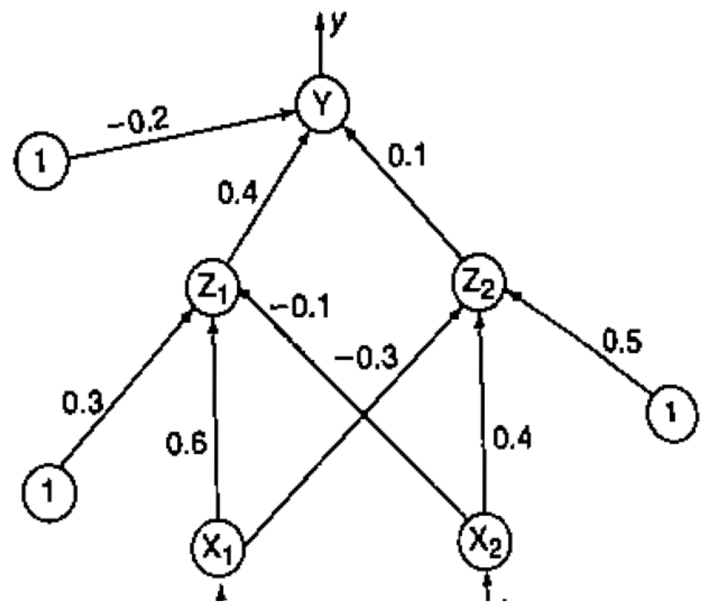
Hidden Layer Updated Weight

Updated weight

$$v_{01}(new) = v_{01}(old) + \Delta v_{01} = 0.3 + 0.00295 = 0.30295$$

$$v_{11}(new) = v_{11}(old) + \Delta v_{11} = 0.6 + 0 = 0.6$$

$$\vdots$$



Derivative of activation function f()

Binary sigmoid function / Logistic function

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

where λ = steepness parameter

Derivative of this function

$$f'(x) = \lambda f(x) [1 - f(x)]$$

Bipolar sigmoid

$$f(x) = \frac{2}{1+e^{-\lambda x}} - 1 = \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}$$

$$f'(x) = \frac{\lambda}{2} (1 + f(x)) (1 - f(x))$$

Derivative of activation function f()

Hyperbolic tangent function

$$h(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{1-e^{-2x}}{1+e^{-2x}}$$

$$h'(x) = (1+h(x))(1-h(x))$$

(Hyperbolic tangent function is closely related to bipolar ~~to~~ sigmoid)