Introduction to Neural Networks

Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Neuron in ANNs tend to have fewer connections than biological neurons.
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.

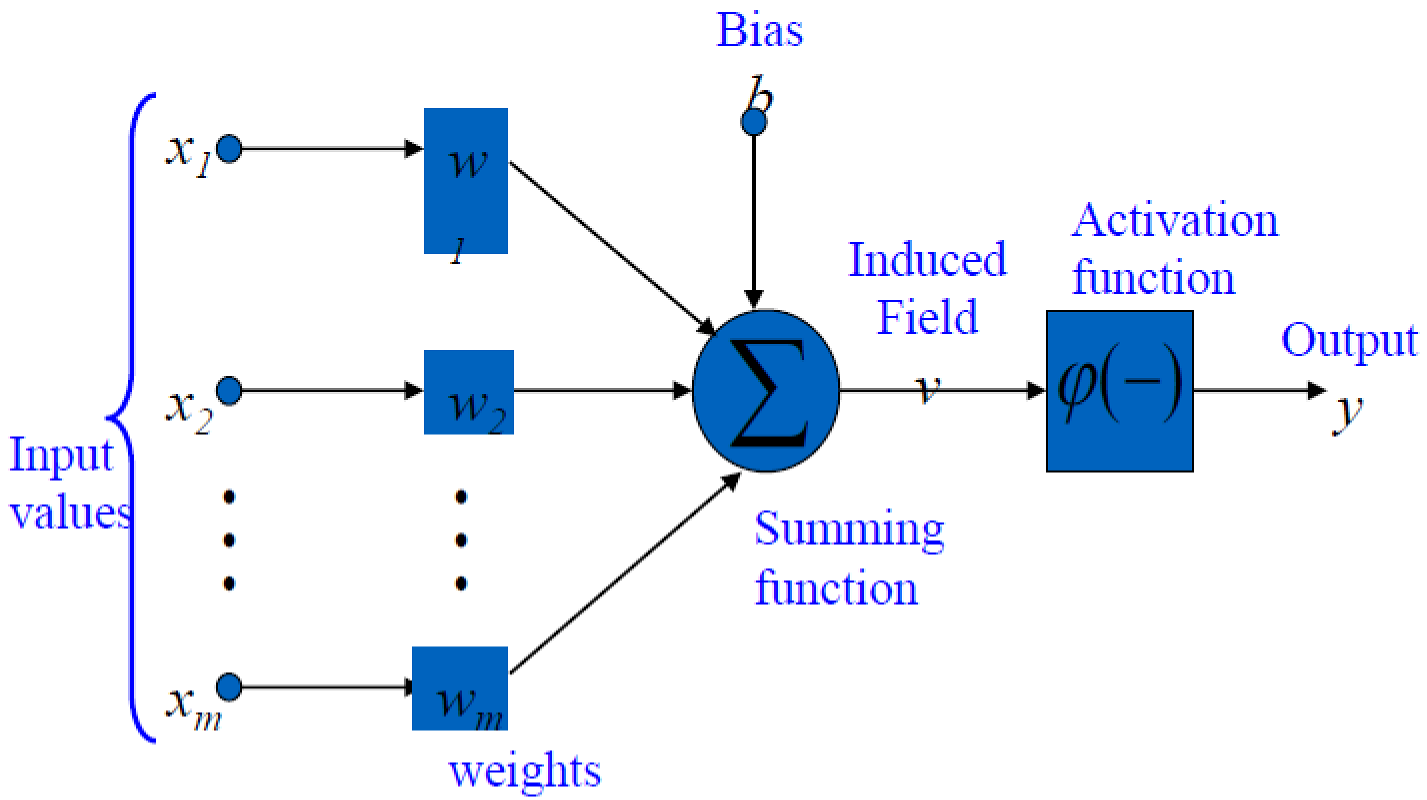
Contd..

- An Artificial Neural Network is specified by:
  - **neuron model**: the information processing unit of the NN,
  - **an architecture**: a set of neurons and links connecting neurons. Each link has a weight,
  - **a learning algorithm**: used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.
- The aim is to obtain a NN that is trained and generalizes well.
- It should behaves correctly on new instances of the learning task.

Neuron

- The neuron is the basic information processing unit of a NN. It consists of:
  1. A set of links, describing the neuron inputs, with weights $W_1, W_2, \dots, W_m$
  2. An adder function (linear combiner) for computing the weighted sum of the inputs: (real numbers) $$u = \sum_{j=1}^{m} w_j x_j$$
  3. Activation function φ for limiting the amplitude of the neuron output. Here 'b' denotes bias. $$y = \varphi(u + b)$$

The Neuron Diagram



Bias of a Neuron

- The bias ***b*** has the effect of applying a transformation to the weighted sum ***u***

$$v = u + b$$

- The bias is an external parameter of the neuron. It can be modeled by adding an extra input.
- ***v*** is called **induced field** of the neuron

$$v = \sum_{j=0}^{m} w_j x_j$$

$$w_0 = b$$

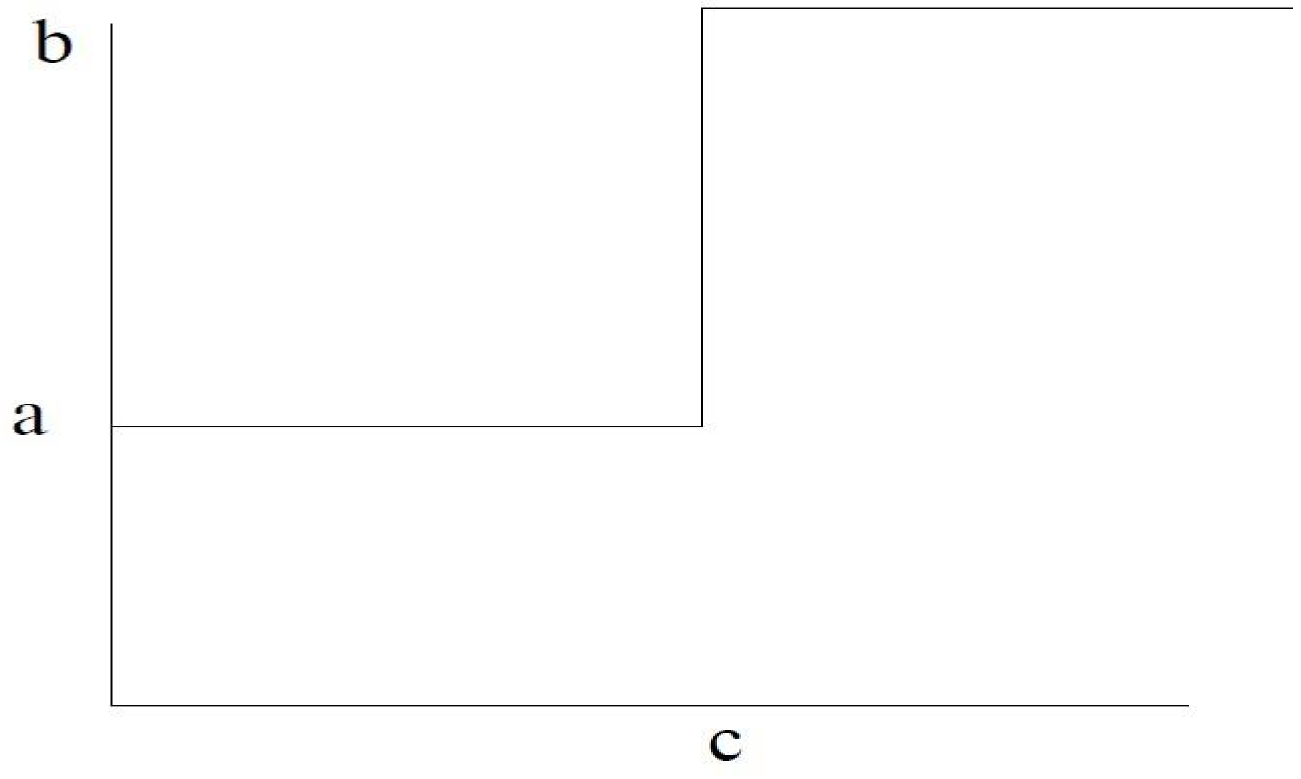
Neuron Models

- The choice of activation function φ determines the neuron model.

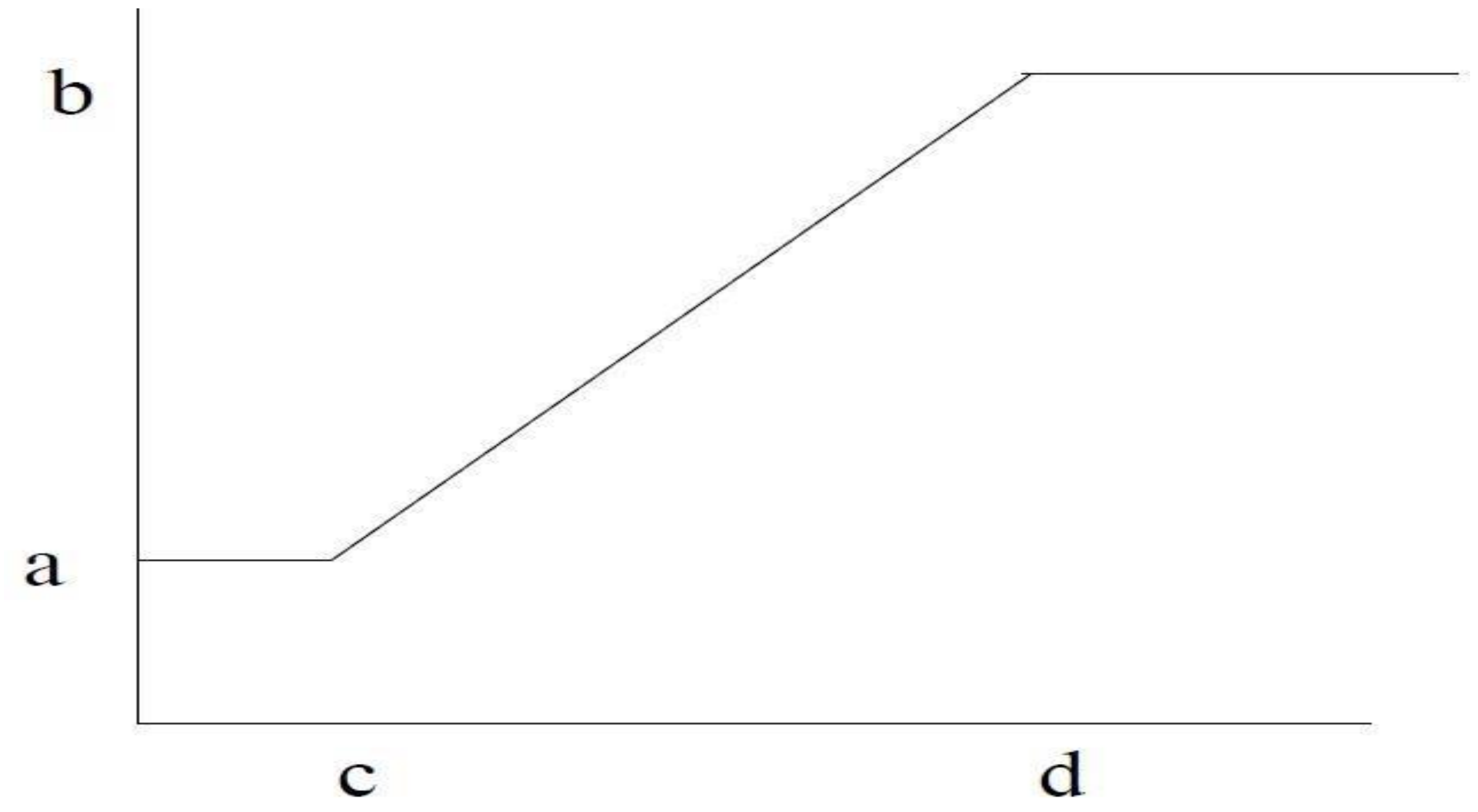
Examples:

- step function: $$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > c \end{cases}$$
- ramp function: $$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > d \\ a + ((v-c)(b-a)/(d-c)) & \text{otherwise} \end{cases}$$
- sigmoid function with z,x,y parameters $$\varphi(v) = z + \frac{1}{1 + \exp(-xv + y)}$$
- Gaussian function: $$\varphi(v) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{v-\mu}{\sigma}\right)^2\right)$$

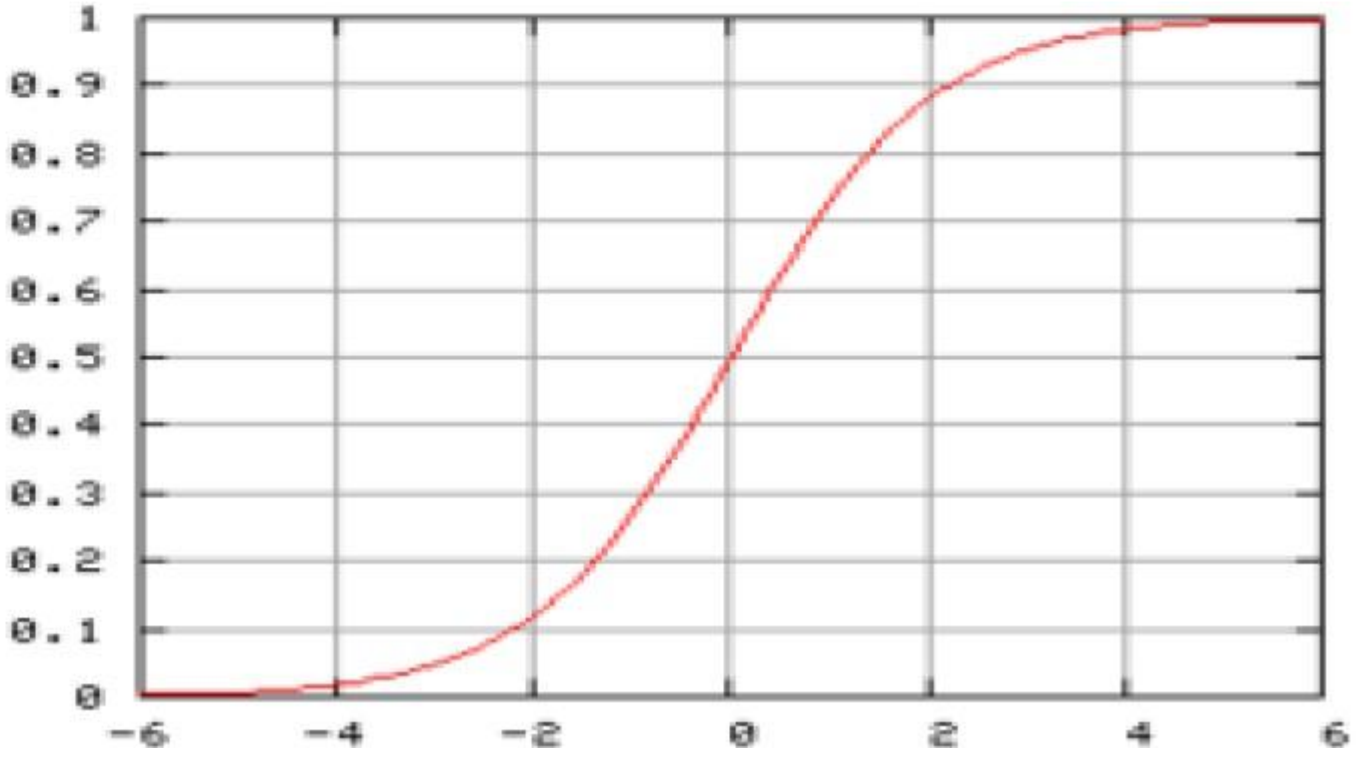
Step Function



Ramp Function



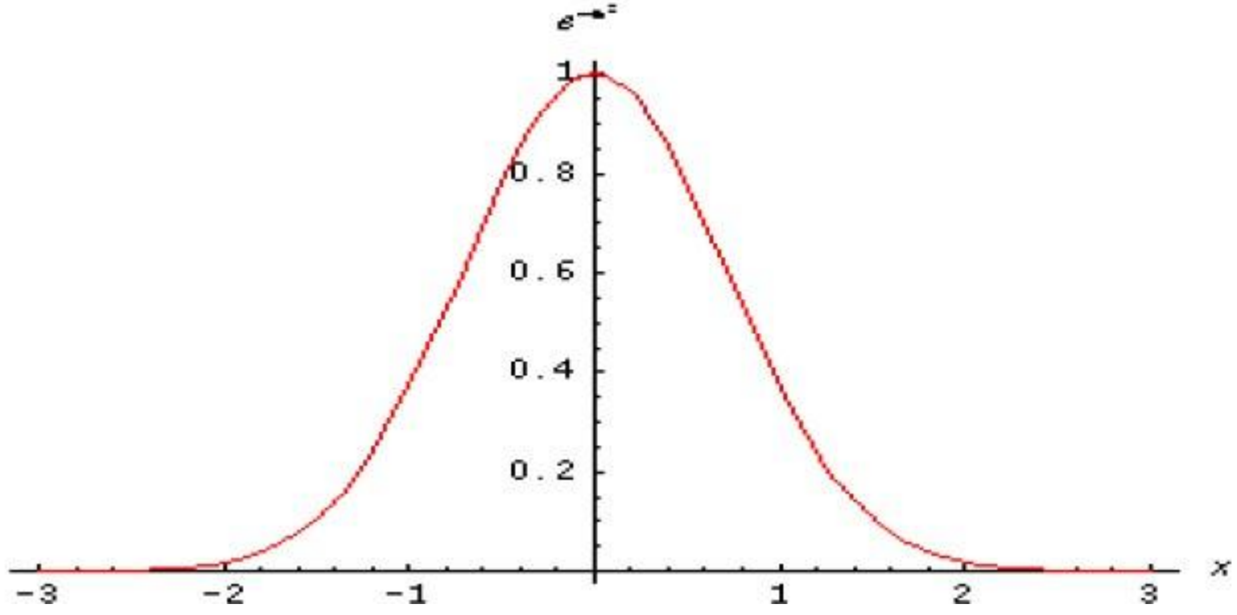
Sigmoid function



The Neuron Diagram

• The **Gaussian function** is the probability function of the normal distribution. Sometimes also called the frequency curve.

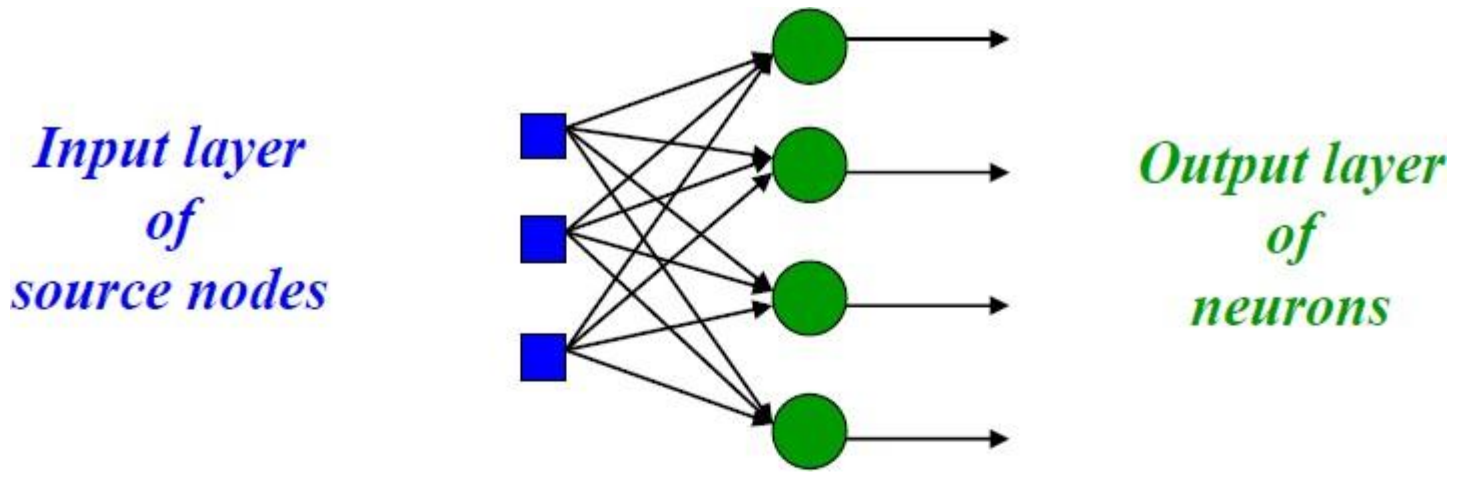
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2},$$



Network Architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train

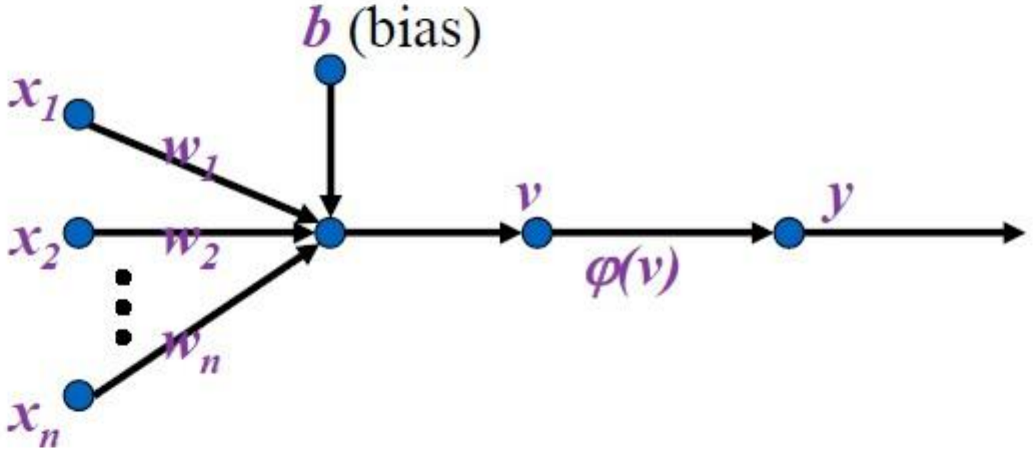
Single Layer Feed-forward



Perceptron: Neuron Model (Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

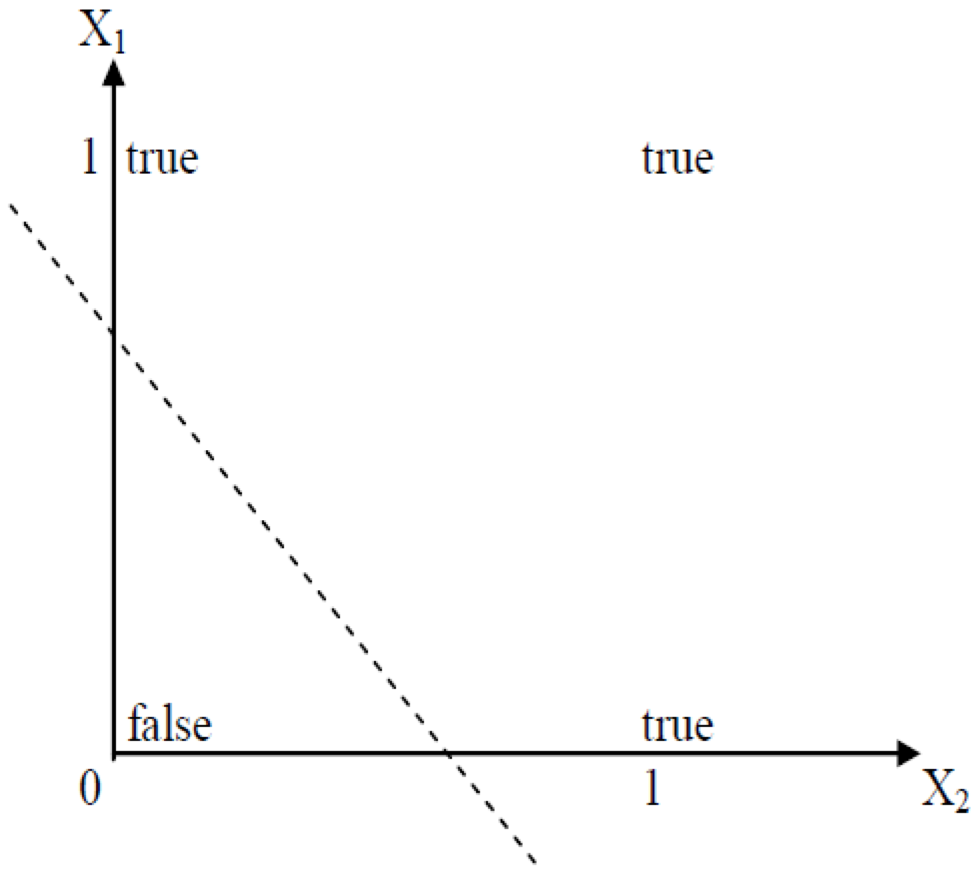
$$\varphi(v) = \begin{cases} +1 \text{ if } & v \geq 0 \\ -1 \text{ if } & v < 0 \end{cases}$$



Perceptron for Classification

- The perceptron is used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- The perceptron can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.
- Given training examples of classes C_1, C_2 train the perceptron in such a way that :
  - *If the output of the perceptron is +1 then the input is assigned to class C_1*
  - *If the output is -1 then the input is assigned to C_2*

Boolean function OR – Linearly separable



Learning Process for Perceptron

- Initially assign random weights to inputs between -0.5 and +0.5
- Training data is presented to perceptron and its output is observed.
- If output is incorrect, the weights are adjusted accordingly using following formula.

  $wi \leftarrow wi + (a * xi * e)$, where ‘e’ is error produced and ‘a’ $(-1 < a < 1)$ is learning rate

  - ‘a’ is defined as 0 if output is correct, it is +ve, if output is too low and –ve, if output is too high.
  - Once the modification to weights has taken place, the next piece of training data is used in the same way.
  - Once all the training data have been applied, the process starts again until all the weights are correct and all errors are zero.
  - Each iteration of this process is known as an epoch.

Example: Perceptron to learn OR function

- Initially consider w1 = -0.2 and w2 = 0.4
- Training data say, x1 = 0 and x2 = 0, output is 0.

  Compute y = Step(w1*x1 + w2*x2) = 0.

  Output is correct so weights are not changed.
- For training data x1=0 and x2 = 1, output is 1

  Compute y = Step(w1*x1 + w2*x2) = 0.4 = 1.

  Output is correct so weights are not changed.

Next training data x1=1 and x2 = 0 and output is 1

Compute y = Step(w1*x1 + w2*x2) = - 0.2 = 0. Output is incorrect, hence weights are to be changed.

Assume a = 0.2 and error e=1

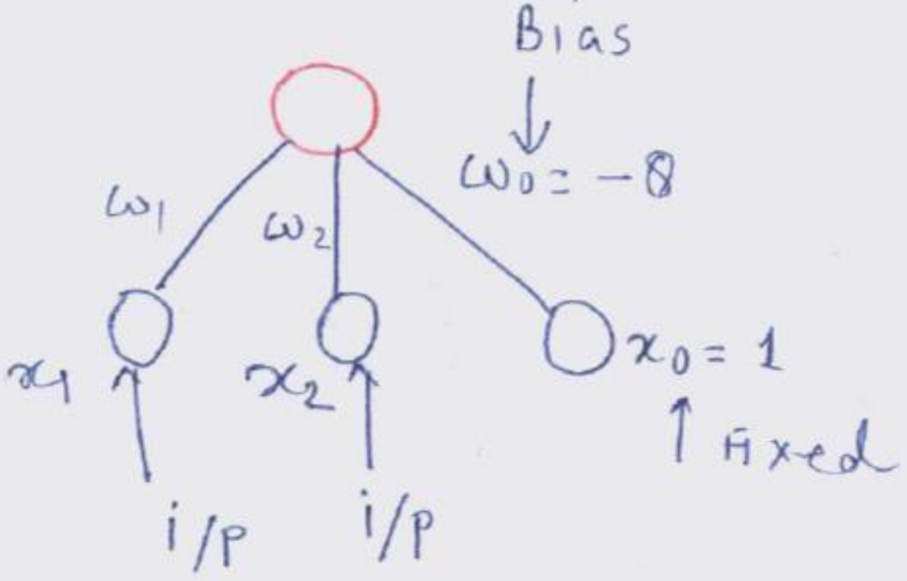
wi = wi + (a * xi * e) gives w1 = 0 and w2 =0.4

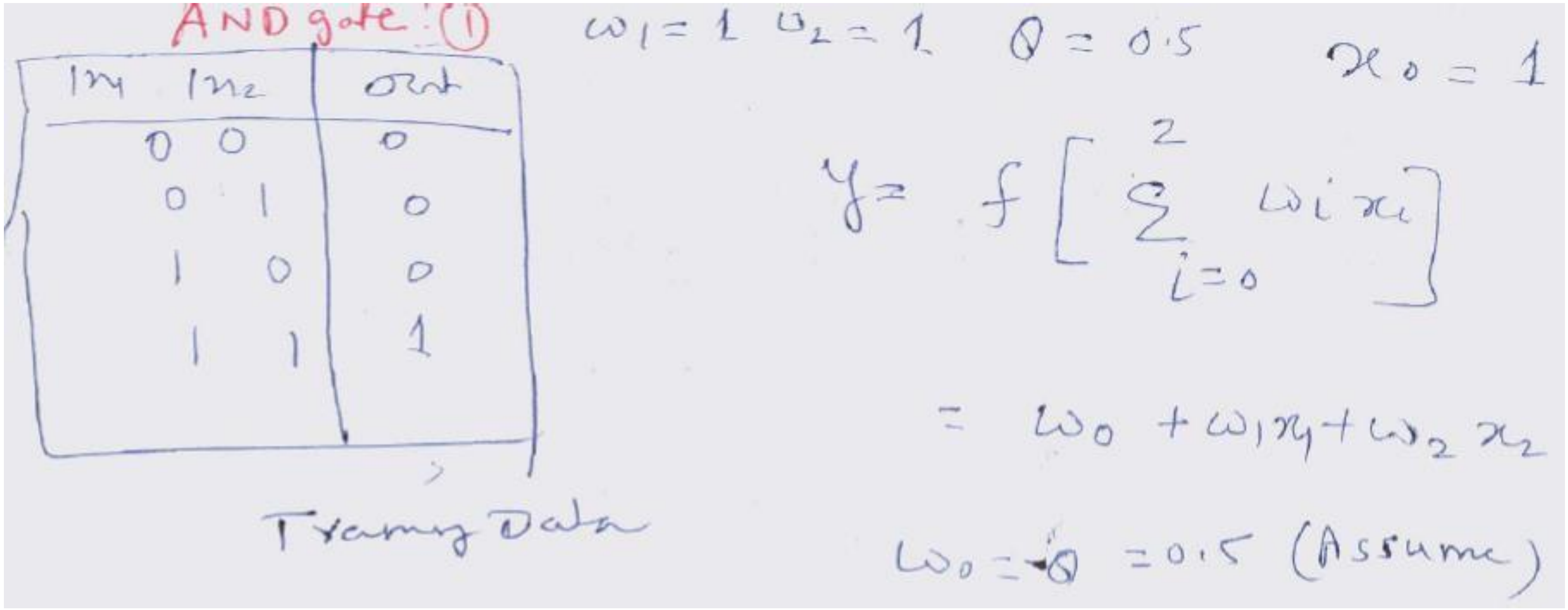
With these weights, test the remaining test data.

- Repeat the process till we get stable result.

Tutorial (AND Gate)

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1





AND gate : (1)

$w_1 = 1$ $w_2 = 1$ $\theta = 0.5$ $x_0 = 1$

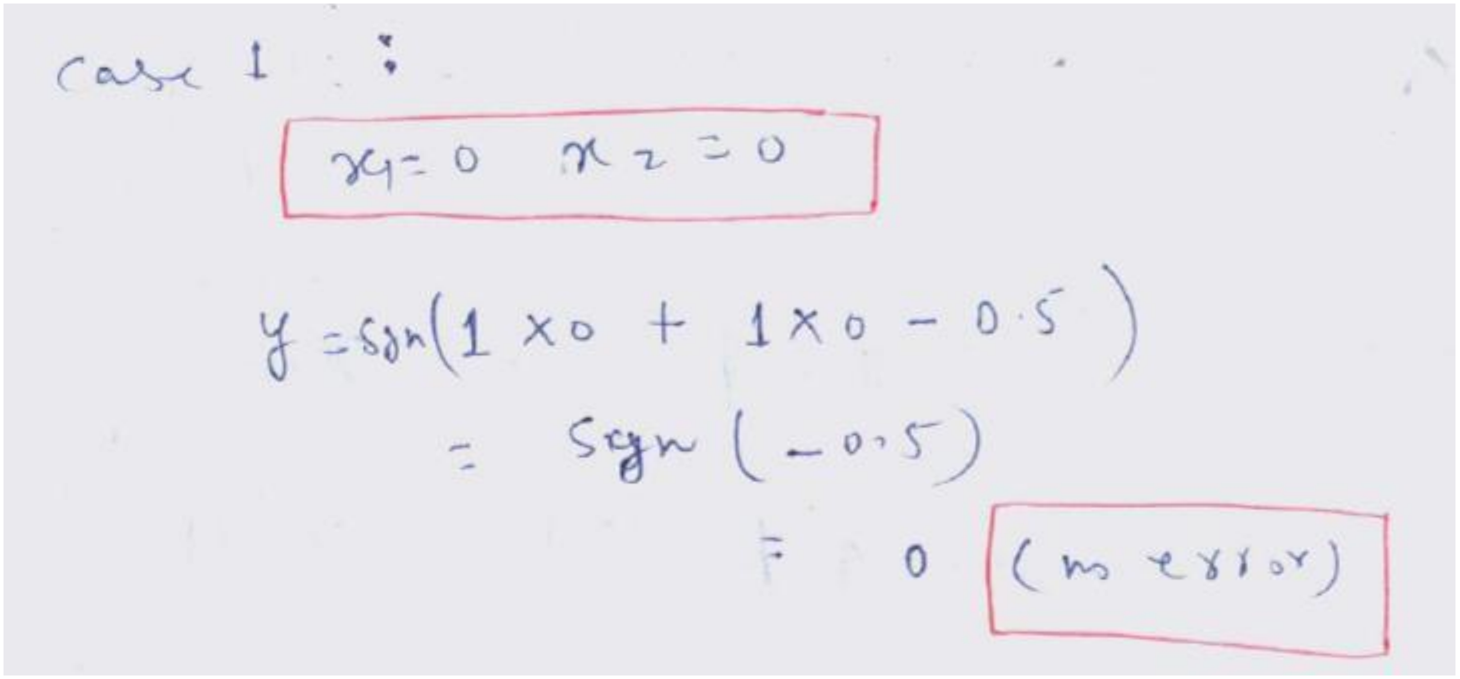
In1	In2	Out
0	0	0
0	1	0
1	0	0
1	1	1

Training Data

$$y = f\left[\sum_{i=0}^{2} w_i x_i\right]$$

$$= w_0 + w_1 x_1 + w_2 x_2$$

$w_0 = -\theta = 0.5$ (Assume)



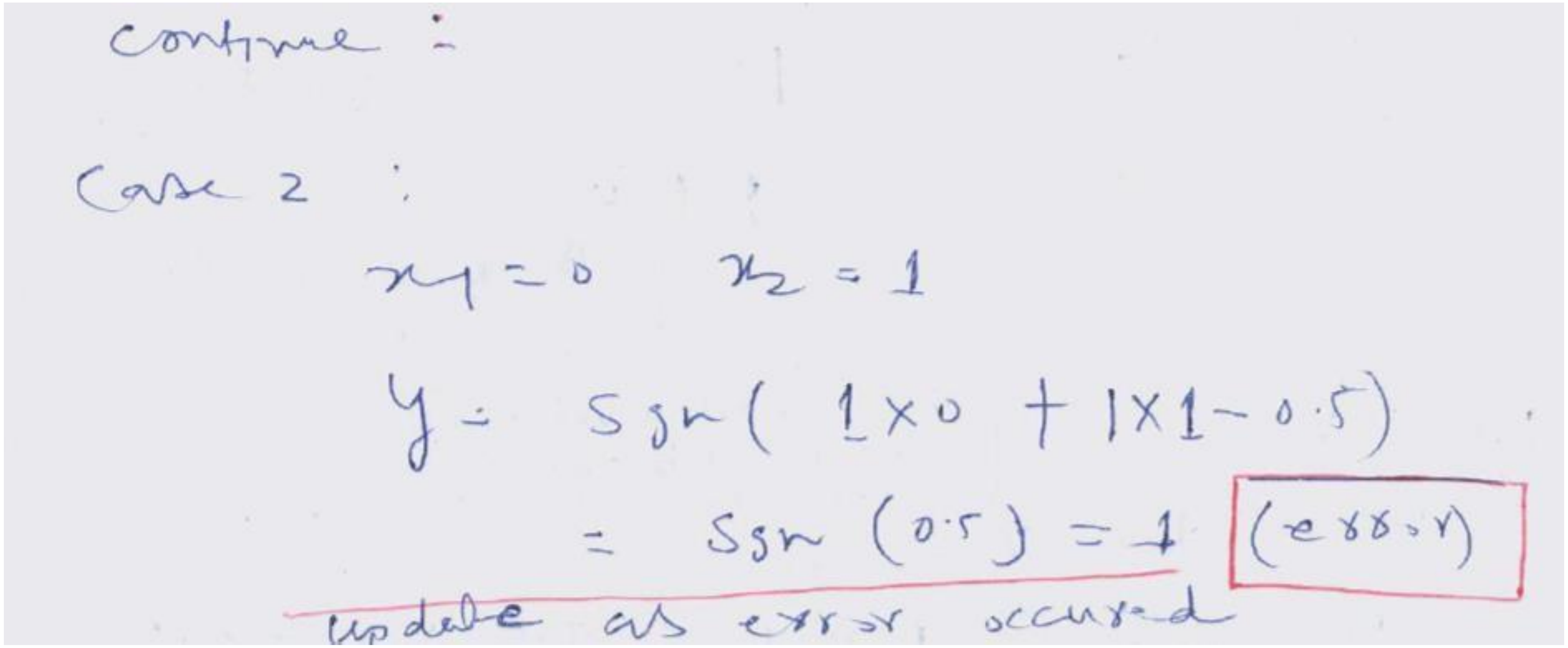
case 1 :

$$\boxed{x_1 = 0 \quad x_2 = 0}$$

$$y = \operatorname{sgn}(1 \times 0 + 1 \times 0 - 0.5)$$

$$= \operatorname{sgn}(-0.5)$$

$$= 0 \quad \boxed{(\text{no error})}$$



continue :-

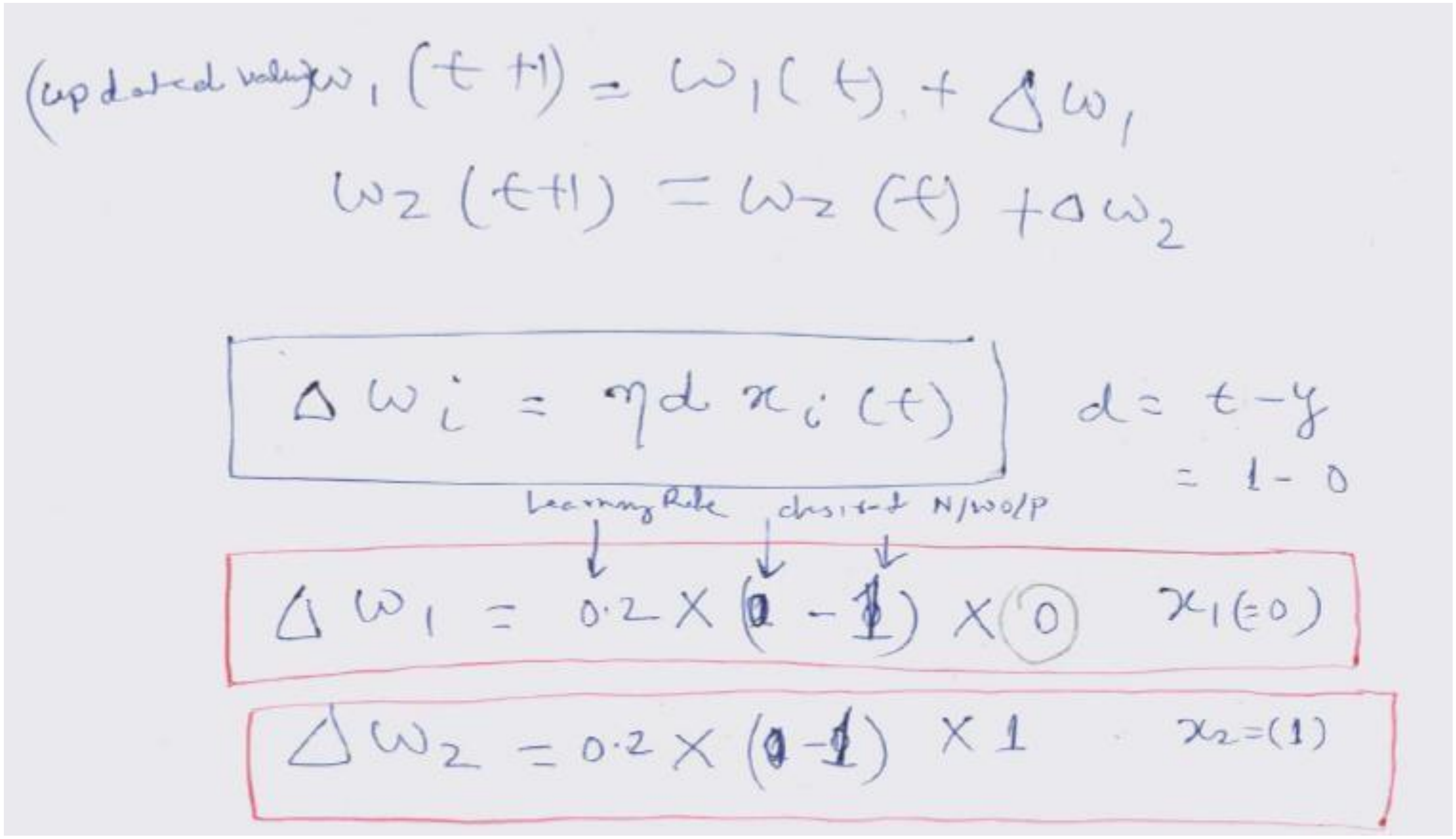
Case 2 :

$$x_1 = 0 \quad x_2 = 1$$

$$y = \text{sgn}(1 \times 0 + 1 \times 1 - 0.5)$$

$$= \underline{\text{sgn}(0.5) = 1} \quad \boxed{(\text{error})}$$

update as error occured



(updated value) $w_1(t+1) = w_1(t) + \Delta w_1$

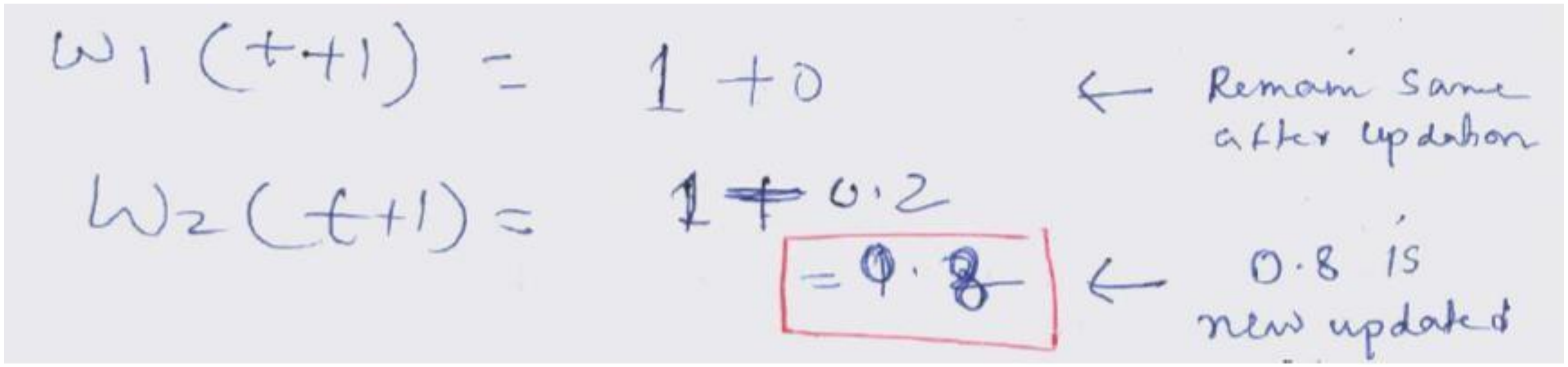
$$w_2(t+1) = w_2(t) + \Delta w_2$$

$$\boxed{\Delta w_i = \eta d\, x_i(t)} \qquad d = t - y$$
$$= 1 - 0$$

Learning Rate desired N/W O/P

$$\boxed{\Delta w_1 = 0.2 \times (0 - 1) \times 0 \qquad x_1 (= 0)}$$

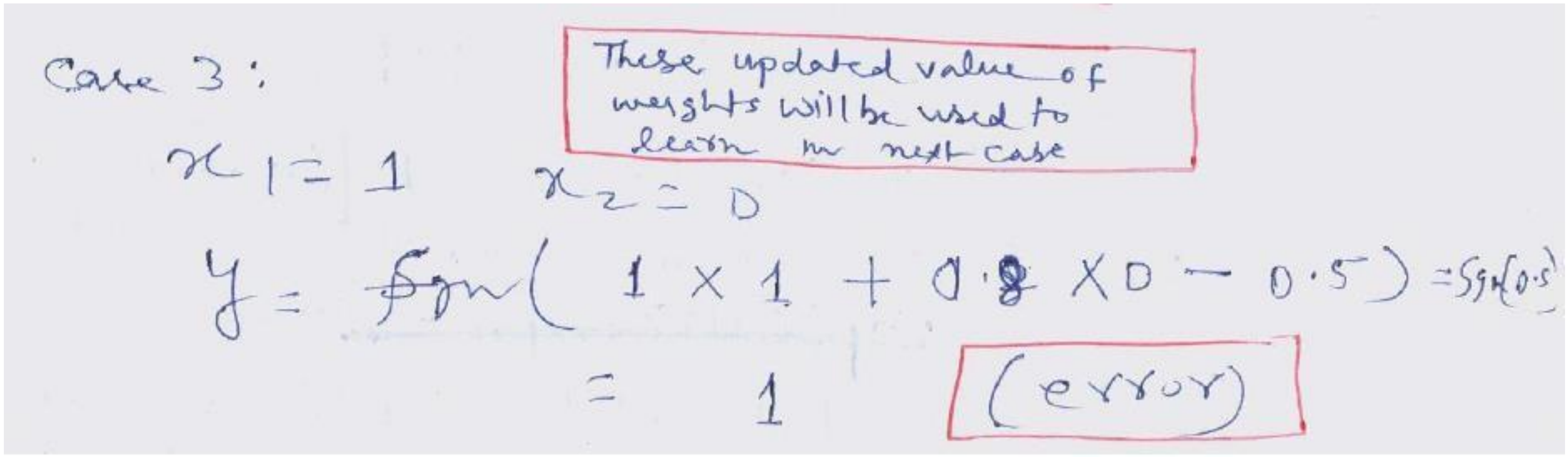
$$\boxed{\Delta w_2 = 0.2 \times (0 - 1) \times 1 \qquad x_2 = (1)}$$



$w_1(t+1) = 1 + 0$ ← Remain same after updation

$W_2(t+1) = 1 - 0.2$

$\boxed{= 0.8}$ ← 0.8 is new updated



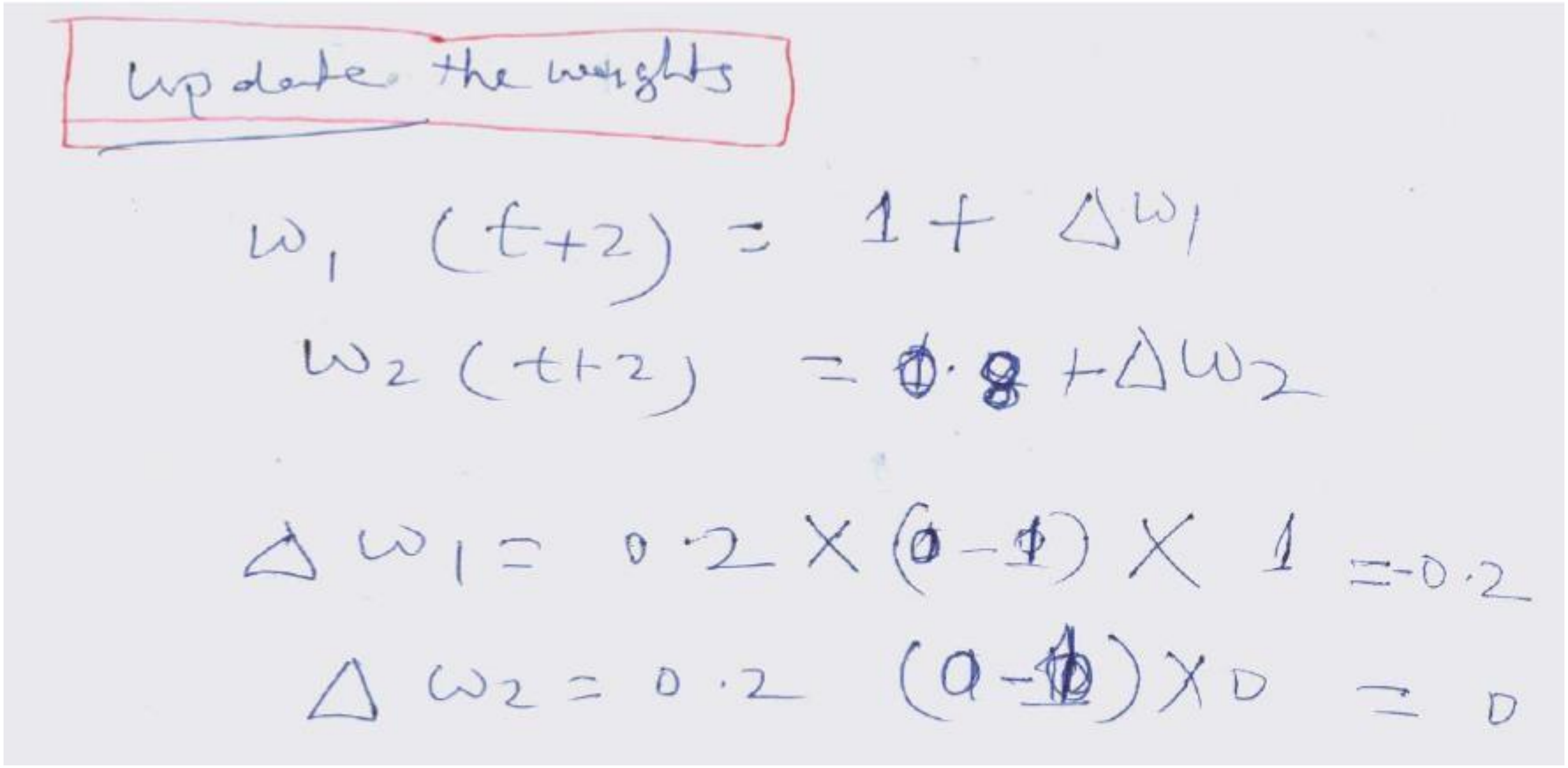
Case 3:

$\boxed{\text{These updated value of weights will be used to learn in next case}}$

$x_1 = 1 \quad x_2 = 0$

$y = \text{sgn}(1 \times 1 + 0.8 \times 0 - 0.5) = \text{sgn}(0.5)$

$= 1 \quad \boxed{(\text{error})}$



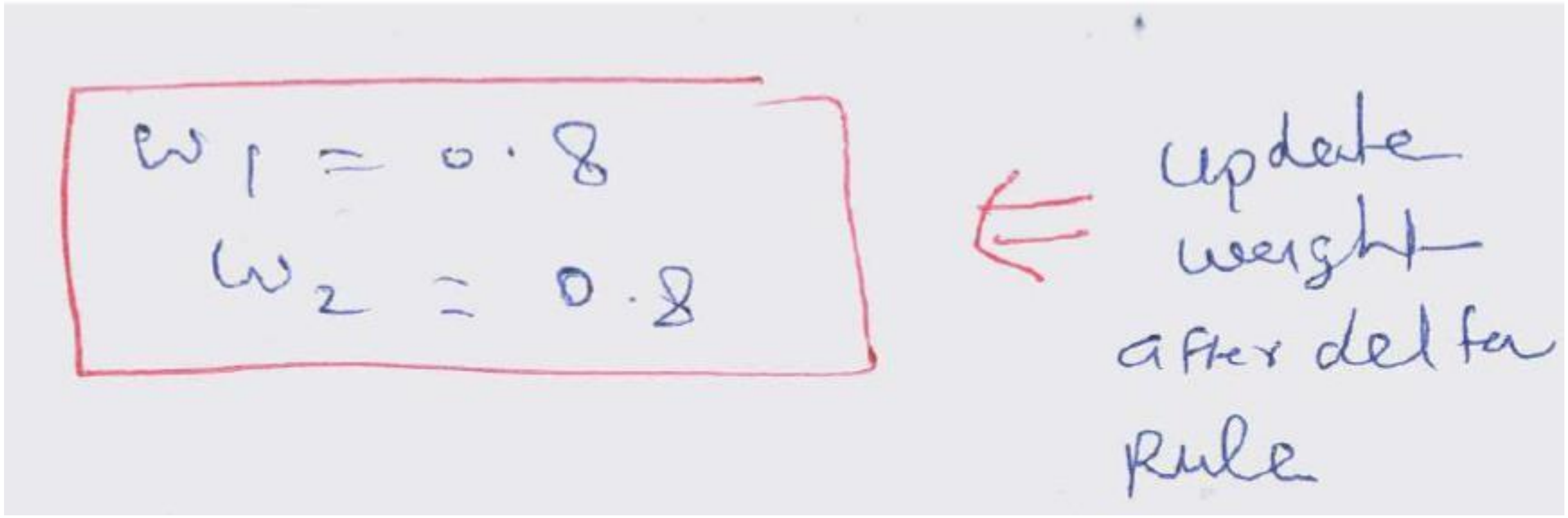
$\boxed{\text{Update the weights}}$

$$w_1(t+2) = 1 + \Delta w_1$$

$$w_2(t+2) = 0.8 + \Delta w_2$$

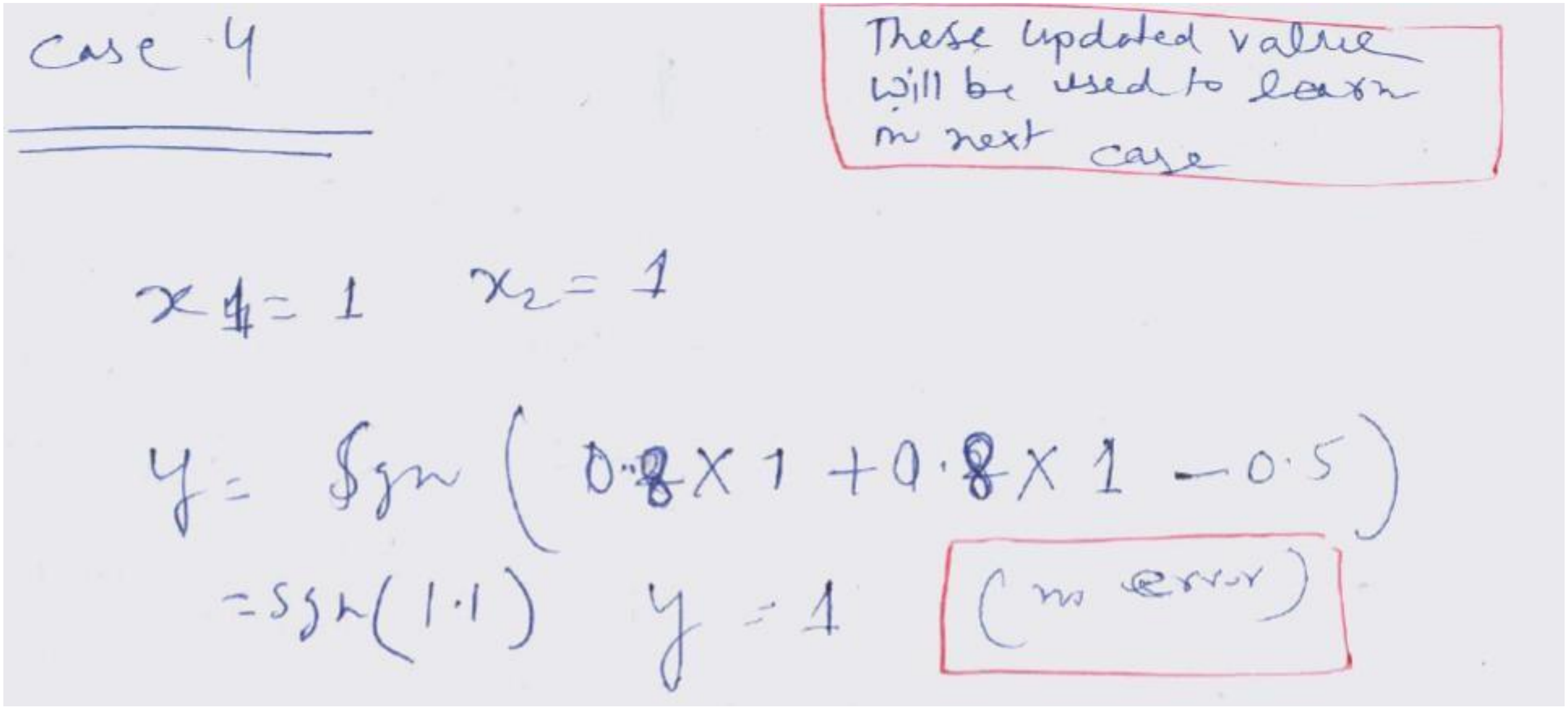
$$\Delta w_1 = 0.2 \times (0 - 1) \times 1 = -0.2$$

$$\Delta w_2 = 0.2 \; (0 - 1) \times 0 = 0$$



$$\boxed{\begin{aligned} w_1 &= 0.8 \\ w_2 &= 0.8 \end{aligned}} \Leftarrow$$

update weight after delta Rule



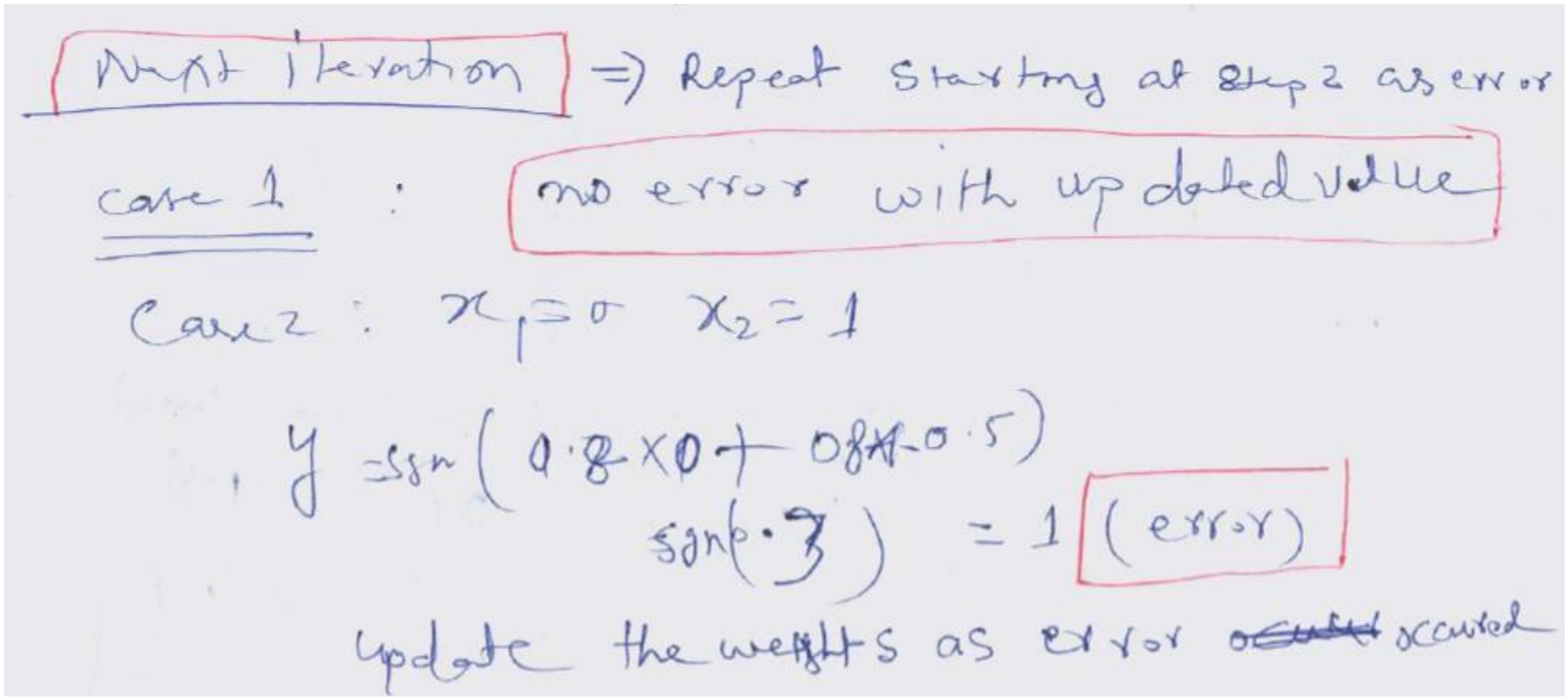
Case 4

These updated value will be used to learn in next case

$x_1 = 1 \quad x_2 = 1$

$y = \text{sgn}(0.8 \times 1 + 0.8 \times 1 - 0.5)$

$= \text{sgn}(1.1) \quad y = 1$ $\boxed{(\text{no error})}$



$\boxed{\text{Next iteration}}$ ⇒ Repeat starting at step 2 as error

Case 1 : $\boxed{\text{no error with updated value}}$

Case 2 : $x_1 = 0 \quad x_2 = 1$

$y = \mathrm{sgn}(0.8 \times 0 + 0.8 \times 1 - 0.5)$

$\mathrm{sgn}(0.3) = 1 \ \boxed{(\text{error})}$

Update the weights as error ~~[illegible]~~ occured

$\boxed{\Delta w_1 = 0.2 \times (0 - 1) \times 0 \quad x_1 = 0} = ~~0.2~~ \; 0$

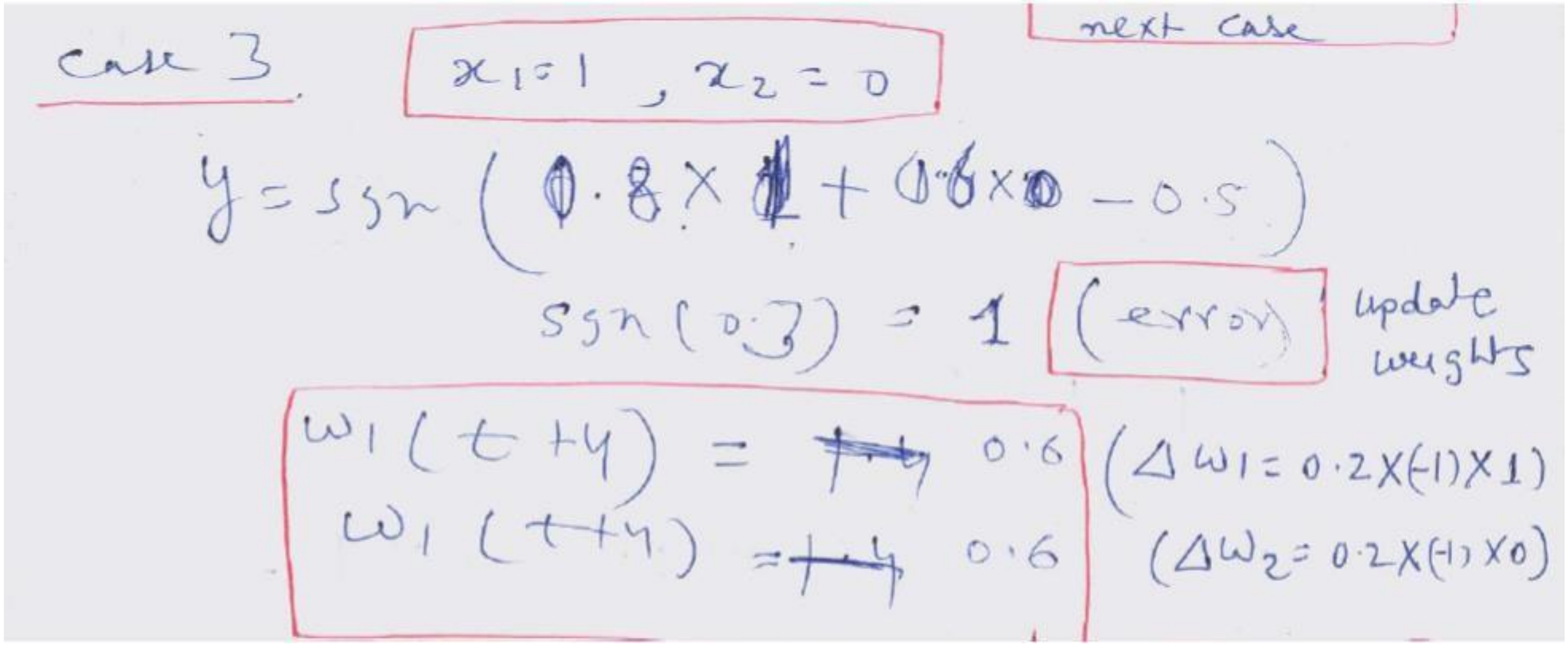
$\boxed{\Delta w_2 = 0.2 \times (0 - 1) \times 1 \quad x_2 = 1} = -0.2$

$w_1(t+3) = w_1(t+2) + \Delta w_1$

$w_2(t+3) = w_2(t+2) + \Delta w_2$

$w_1(t+3) =$ ~~1.2~~ 0.8

$w_2(t+3) =$ ~~1.4~~ 0.6



next case

Case 3

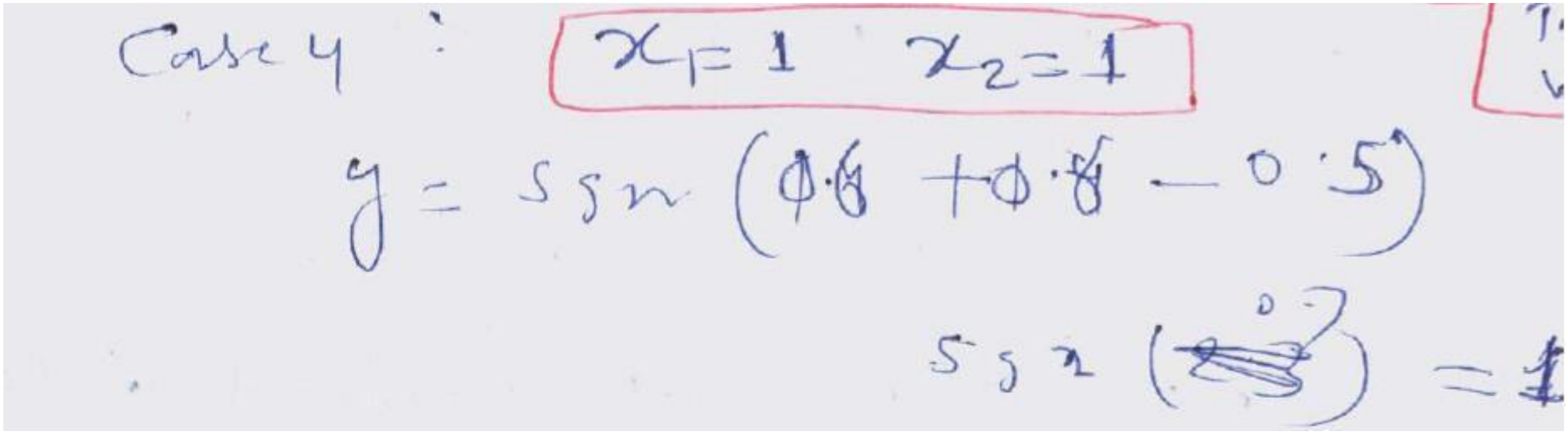
$x_1 = 1$, $x_2 = 0$

$$y = \mathrm{sgn}(0.8 \times 1 + 0.6 \times 0 - 0.5)$$

$\mathrm{sgn}(0.3) = 1$ (error) update weights

$w_1(t+1) = \cancel{1.4}\ 0.6$ $(\Delta w_1 = 0.2 \times (-1) \times 1)$

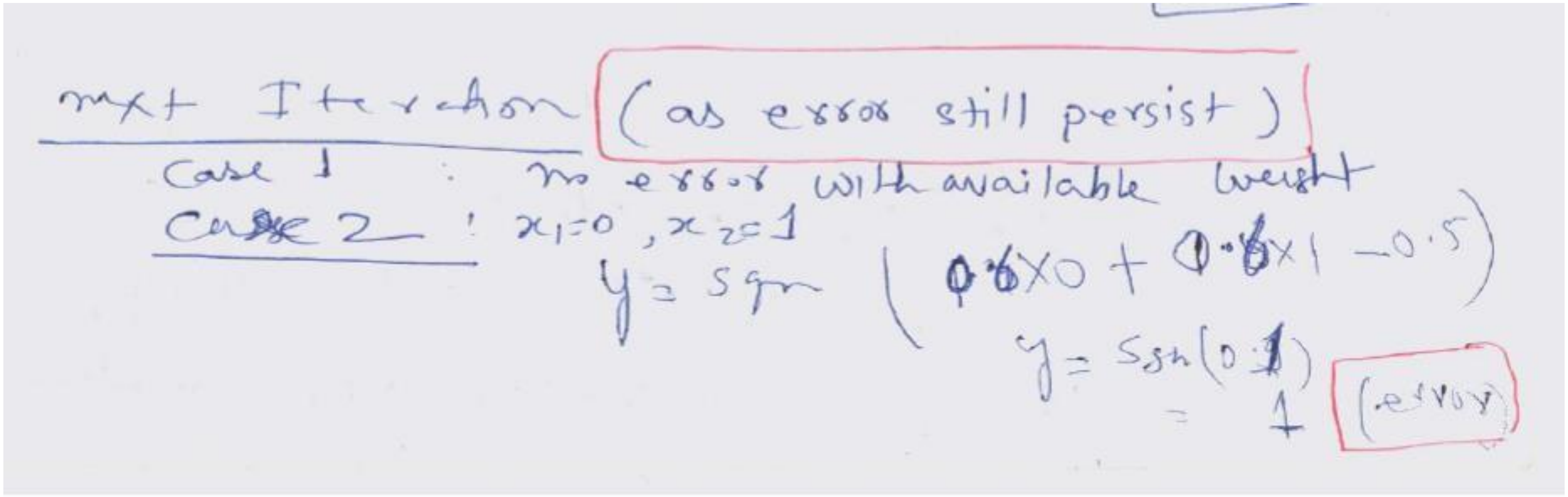
$w_1(t+1) = \cancel{1.4}\ 0.6$ $(\Delta w_2 = 0.2 \times (-1) \times 0)$



Case 4 : $\boxed{x_1 = 1 \quad x_2 = 1}$

$$y = sgn(0.6 + 0.6 - 0.5)$$

$$sgn(\overset{0.7}{\cancel{2.3}}) = 1$$



next Iteration (as error still persist)

Case 1 : no error with available weight

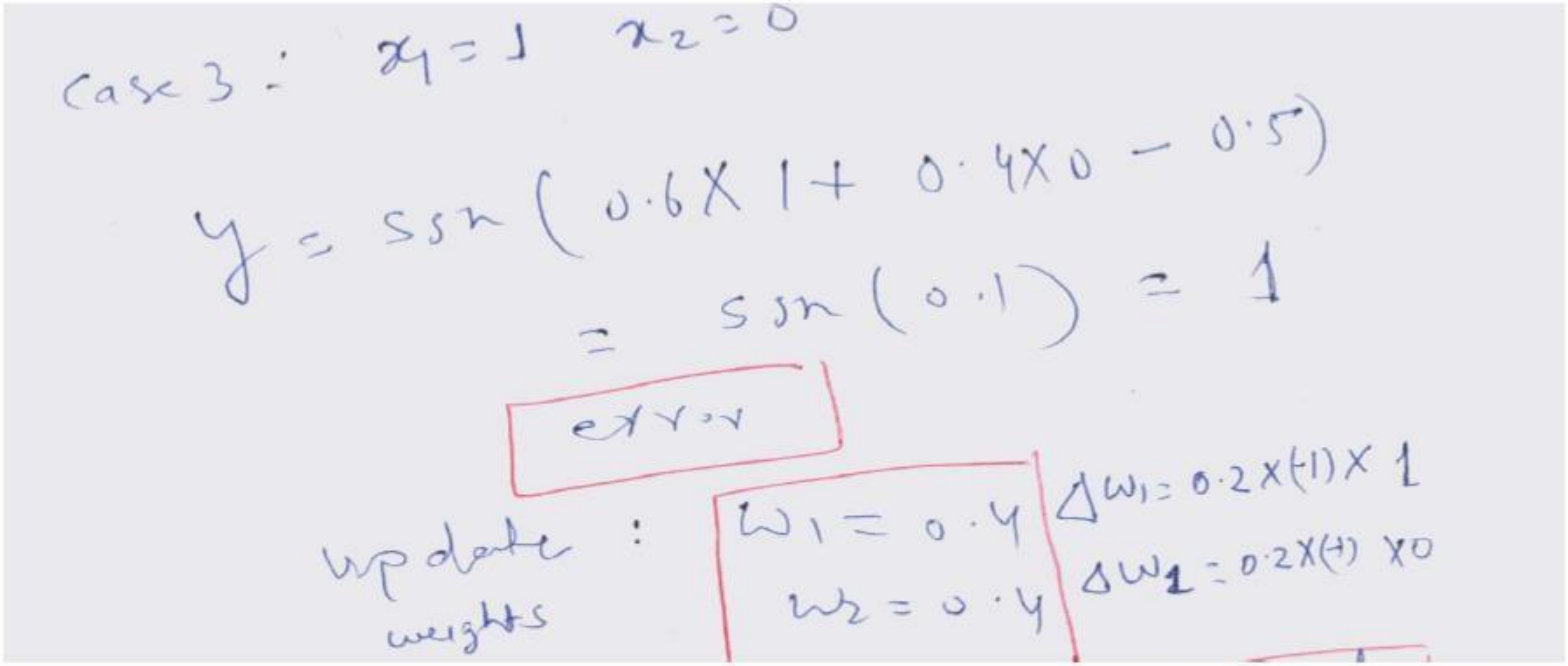
Case 2 : $x_1=0$, $x_2=1$

$y = \text{sgn}\left(0.6\times 0 + 0.6\times 1 - 0.5\right)$

$y = \text{sgn}(0.1)$

$= 1$ (error)

Updated weight : W1 = 0.6 , W2= 0.4



Case 3 : $x_1 = 1 \quad x_2 = 0$

$$y = sgn(0.6 \times 1 + 0.4 \times 0 - 0.5)$$
$$= sgn(0.1) = 1$$

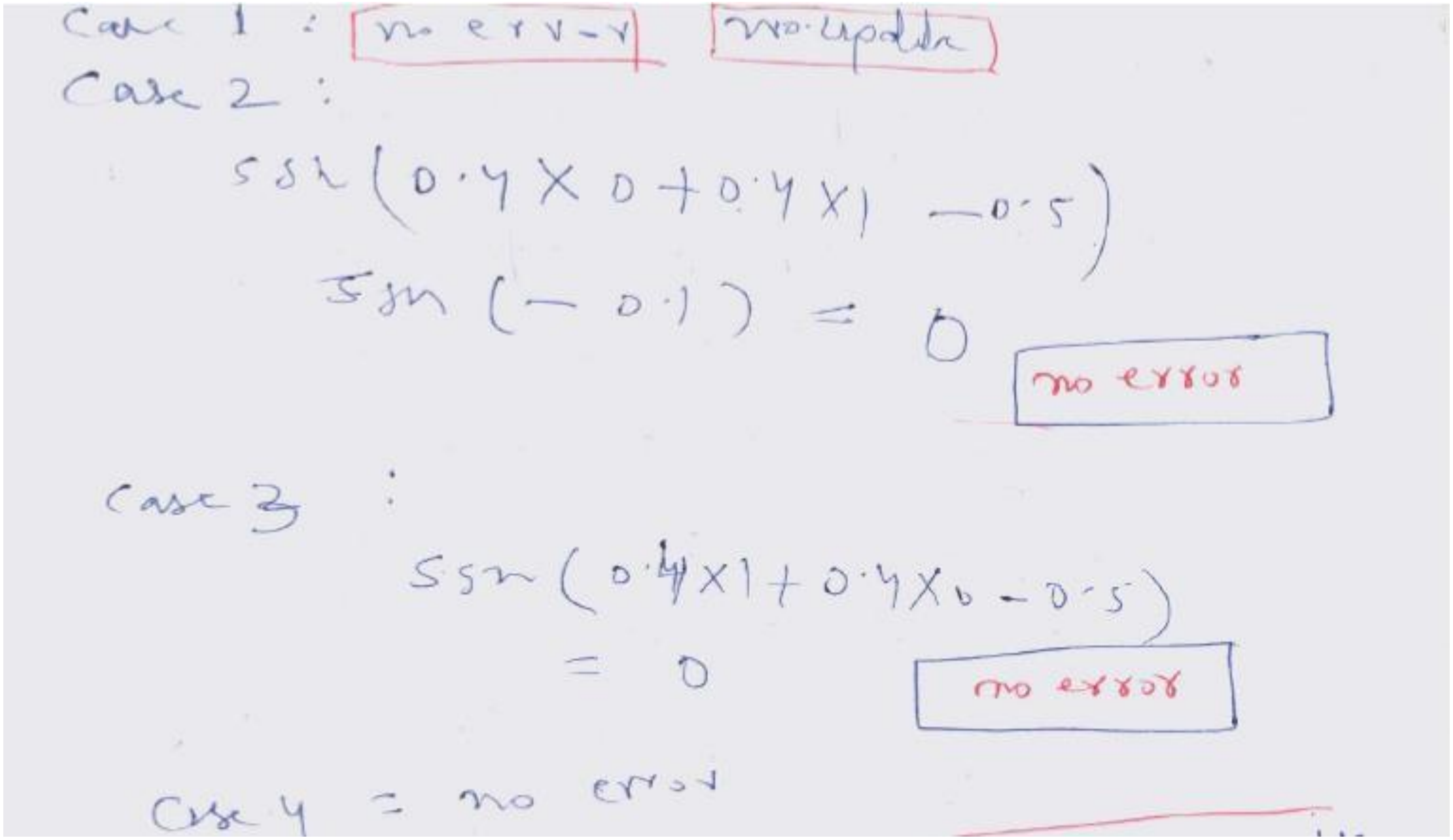
$= \boxed{\text{error}}$

update weights : $\boxed{\begin{matrix} w_1 = 0.4 \\ w_2 = 0.4 \end{matrix}}$

$\Delta w_1 = 0.2 \times (-1) \times 1$

$\Delta w_2 = 0.2 \times (-1) \times 0$

Case 4, x1=1, x2=1 no error observed.



Case 1 : $\boxed{\text{no error}}$ $\boxed{\text{no update}}$

Case 2 :

$$\text{sgn}(0.4 \times 0 + 0.4 \times 1 - 0.5)$$

$$\text{sgn}(-0.1) = 0$$

$\boxed{\text{no error}}$

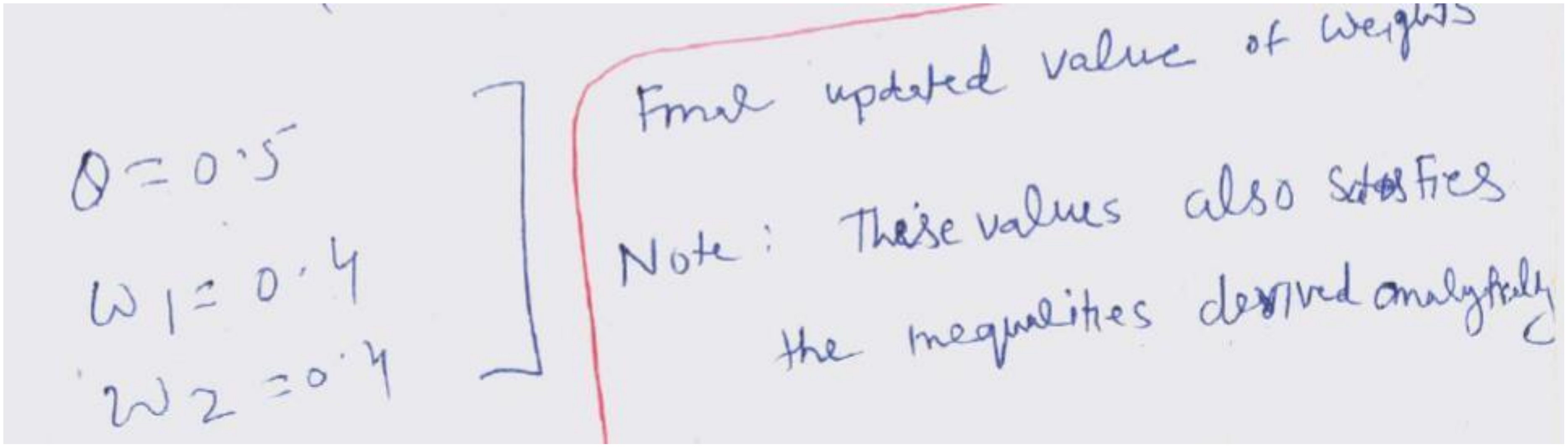
Case 3 :

$$\text{sgn}(0.4 \times 1 + 0.4 \times 0 - 0.5)$$

$$= 0$$

$\boxed{\text{no error}}$

Case 4 = no error



$\theta = 0.5$

$w_1 = 0.4$

$w_2 = 0.4$

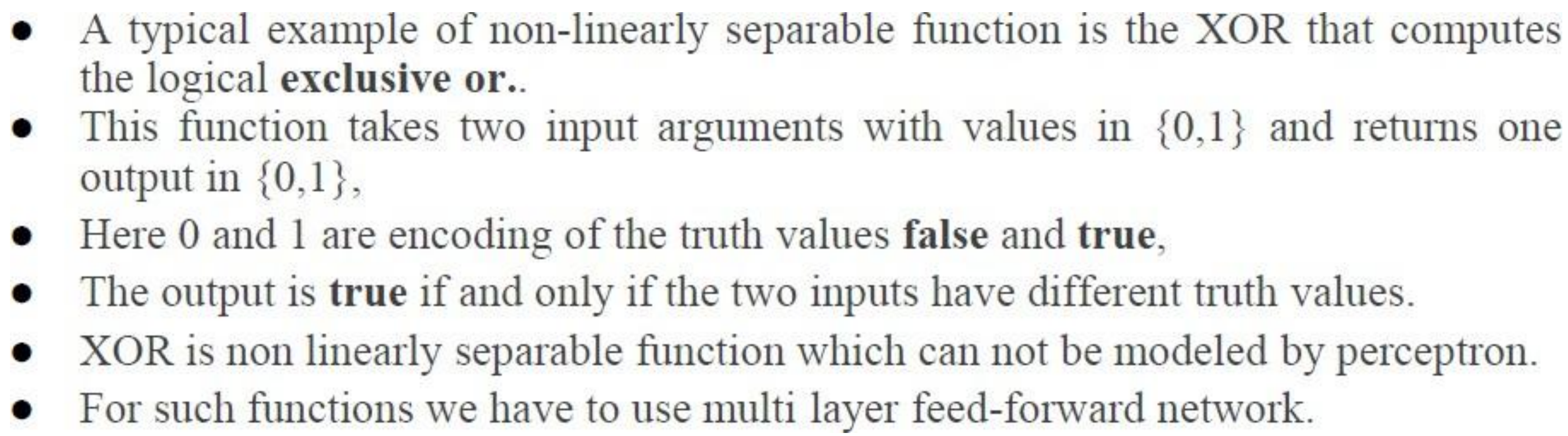
Final updated value of weights

Note: These values also satisfies the inequalities derived analytically

Perceptron: Limitations

- The perceptron can only model linearly separable functions,
  - those functions which can be drawn in 2-dim graph and single straight line separates values in two part.
- Boolean functions given below are linearly separable:
  - AND
  - OR
  - COMPLEMENT
- It cannot model XOR function as it is non linearly separable.
  - When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

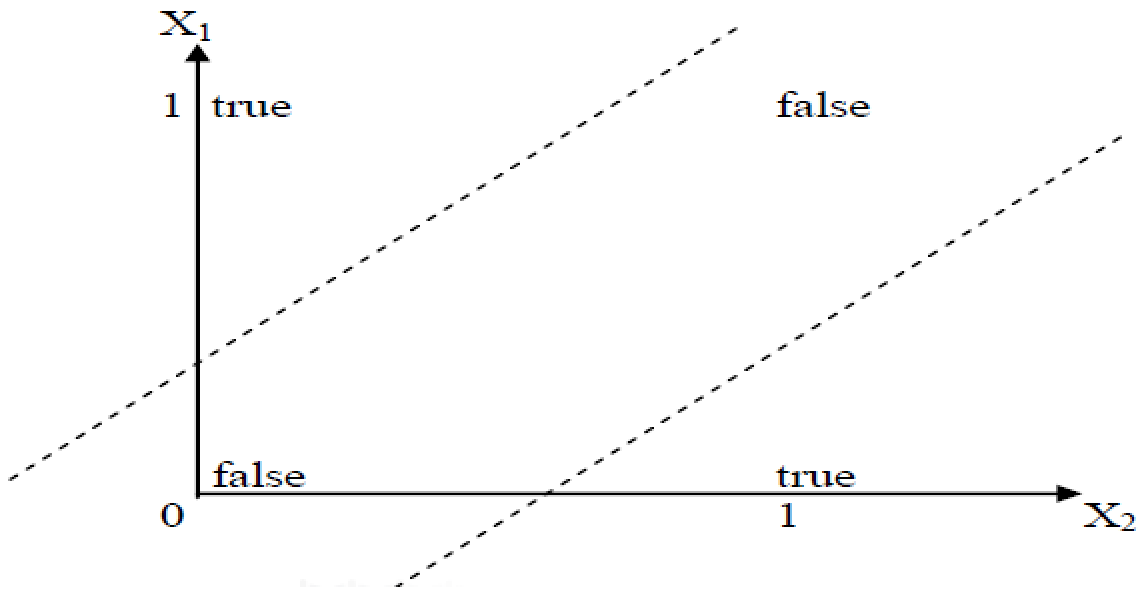
XOR –Non linearly separable function



- A typical example of non-linearly separable function is the XOR that computes the logical **exclusive or.**.
- This function takes two input arguments with values in {0,1} and returns one output in {0,1},
- Here 0 and 1 are encoding of the truth values **false** and **true**,
- The output is **true** if and only if the two inputs have different truth values.
- XOR is non linearly separable function which can not be modeled by perceptron.
- For such functions we have to use multi layer feed-forward network.

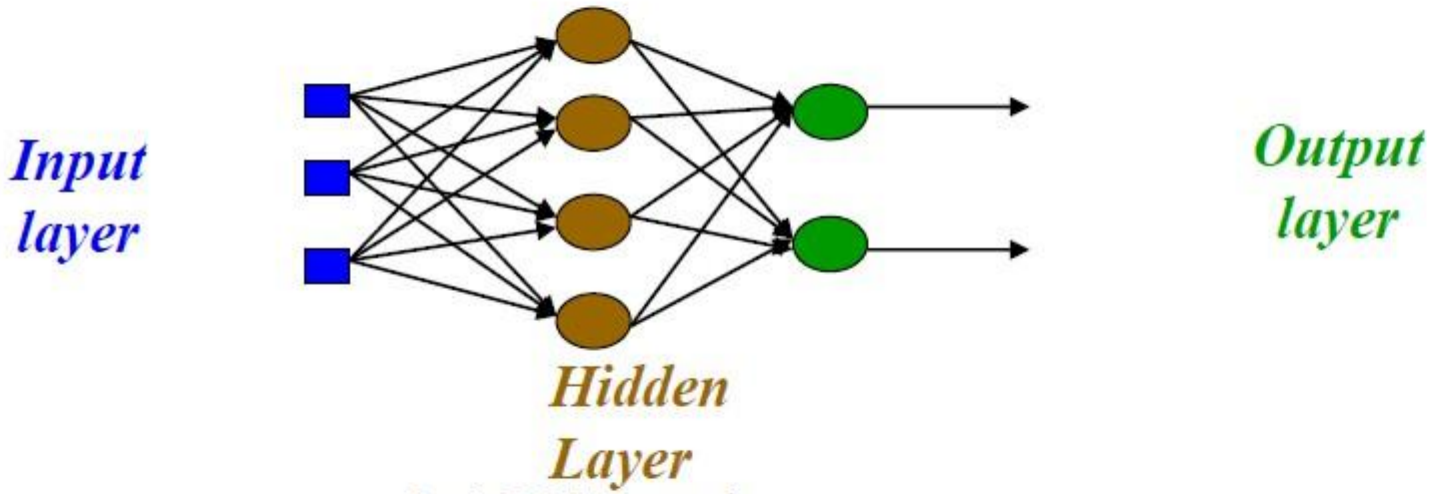
Input		Output
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

These two classes (true and false) cannot be separated using a line. Hence XOR is non linearly separable.



Multi layer feed-forward NN (FFNN)

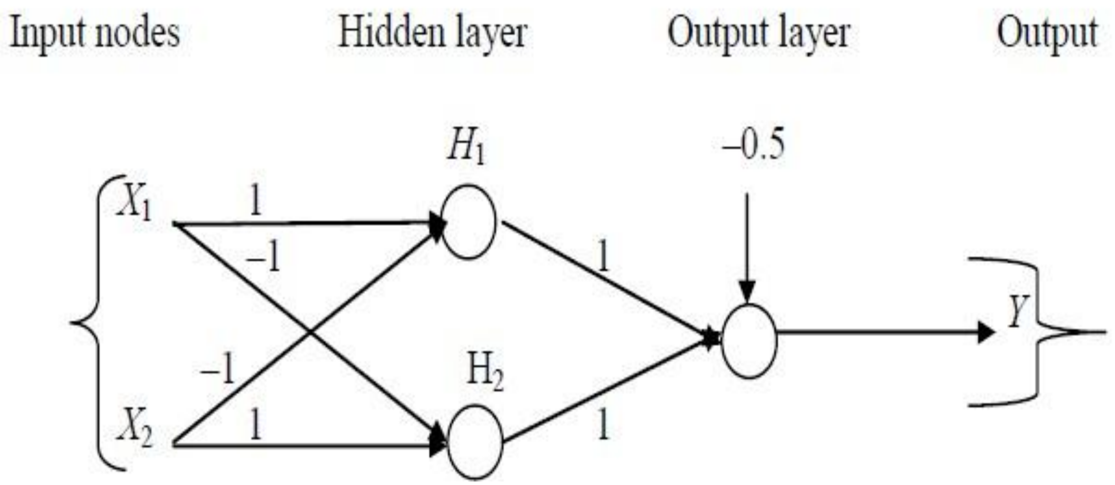
- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning tasks.



3-4-2 Network

FFNN for XOR

- The ANN for XOR has two hidden nodes that realizes this non-linear separation and uses the sign (step) activation function.
- Arrows from input nodes to two hidden nodes indicate the directions of the weight vectors (1,-1) and (-1,1).
- The output node is used to combine the outputs of the two hidden nodes.



Inputs		Output of Hidden Nodes		Output Node	X_1 XOR X_2
X_1	X_2	H_1	H_2		
0	0	0	0	–0.5 → 0	0
0	1	-1 → 0	1	0.5 → 1	1
1	0	1	-1 → 0	0.5 → 1	1
1	1	0	0	–0.5 → 0	0

Since we are representing two states by 0 (false) and 1 (true), we will map negative outputs (–1, –0.5) of hidden and output layers to 0 and positive output (0.5) to 1.

FFNN NEURON MODEL

- The classical learning algorithm of FFNN is based on the gradient descent method.
- For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- The activation function for node i may be defined as a simple form of the **sigmoid function** in the following manner:

$$\varphi(Vi) = \frac{1}{1+e^{(-A^*Vi)}}$$

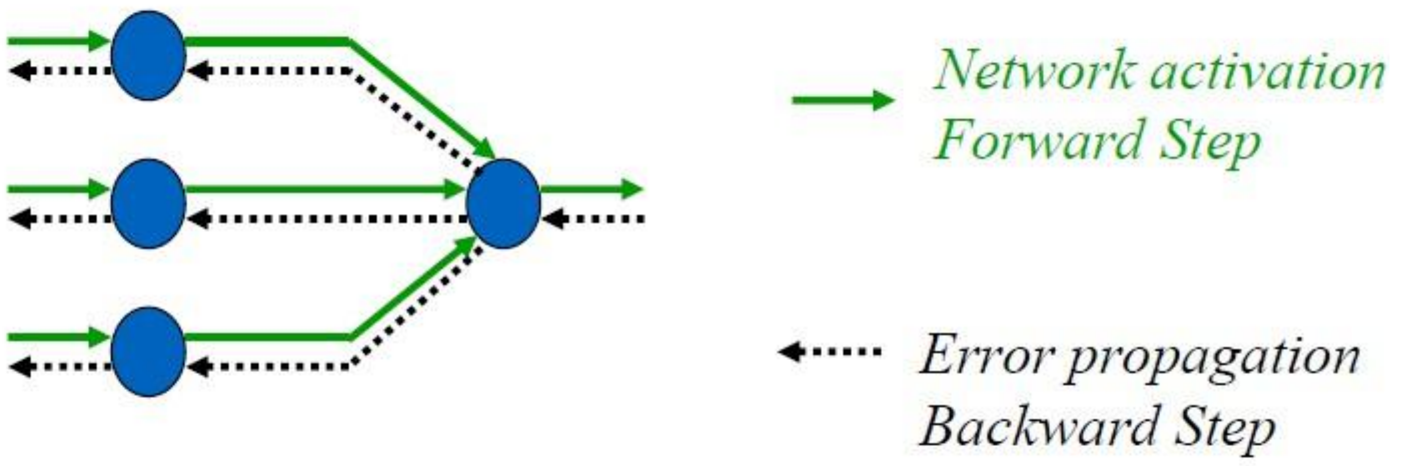
where $A > 0$, $V_i = \sum W_{ij} * Y_j$, such that W_{ij} is a weight of the link from node i to node j and Y_j is the output of node j.

Training Algorithm: Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

Backpropagation

- Back-propagation training algorithm



- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.

Contd..

- Consider a network of three layers.
- Let us use i to represent nodes in input layer, j to represent nodes in hidden layer and k represent nodes in output layer.
- w_{ij} refers to weight of connection between a node in input layer and node in hidden layer.
- The following equation is used to derive the output value Yj of node j

$$Yj = \frac{1}{1+e^{-X_j}}$$

where, $X_j = \sum x_i \,.\, w_{ij} - \theta_j$, $1 \leq i \leq n$; n is the number of inputs to node j, and θ_j is threshold for node j

Total Mean Squared Error

- The error of output neuron *k* after the activation of the network on the *n-th* training example (x(n), d(n)) is:

  $e_k(n) = d_k(n) - y_k(n)$

- The network error is the sum of the squared errors of the output neurons:

$$\mathbf{E(n)} = \sum \mathbf{e}_k^2\mathbf{(n)}$$

- The total mean squared error is the average of the network errors of the training examples.

$$E_{AV} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

Weight Update Rule

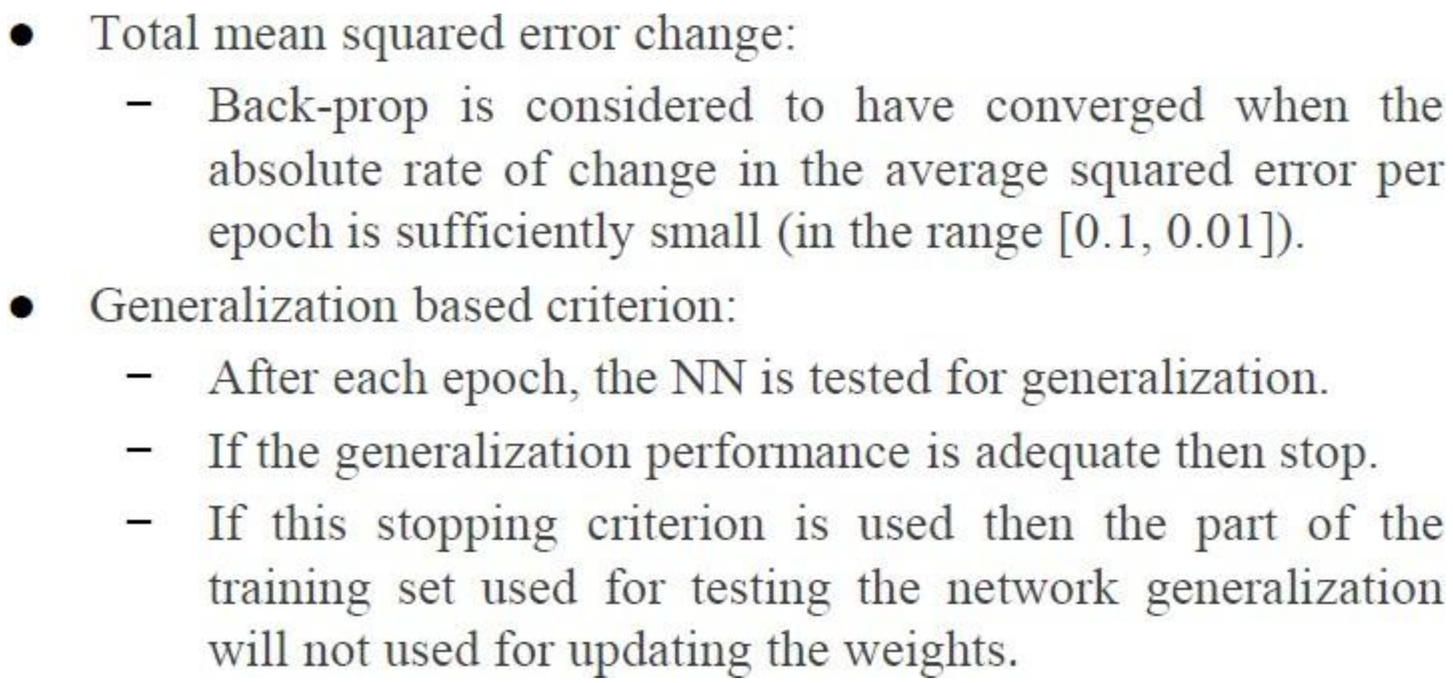
- The Backprop weight update rule is based on the gradient descent method:
  - It takes a step in the direction yielding the maximum decrease of the network error E.
  - This direction is the opposite of the gradient of E.
- Iteration of the Backpropagation algorithm is usually terminated when the sum of squares of errors of the output values for all training data in an epoch is less than some threshold such as 0.01

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \Delta \mathrm{w}_{\mathrm{ij}} = -\eta \frac{\partial E}{\partial \mathrm{w}_{\mathrm{ij}}}$$

Backpropogation learning algorithm (incremental-mode)

```
n=1;
initialize **weights** randomly;
while (stopping criterion not satisfied or n <max_iterations)
    for each example (x,d)
    - run the network with input x and compute the output y
    - update the weights in backward order starting from those
        of the output layer:
                $w_{ji} = w_{ji} + \Delta w_{ji}$
    with $\Delta w_{ji}$ computed using the (generalized) Delta rule
    end-for
    n = n+1;
end-while;
```

Stopping criterions



- Total mean squared error change:
  - Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used then the part of the training set used for testing the network generalization will not used for updating the weights.

NN DESIGN ISSUES

- Data representation
- Network Topology
- Network Parameters
- Training
- Validation

Example (Multilayer)

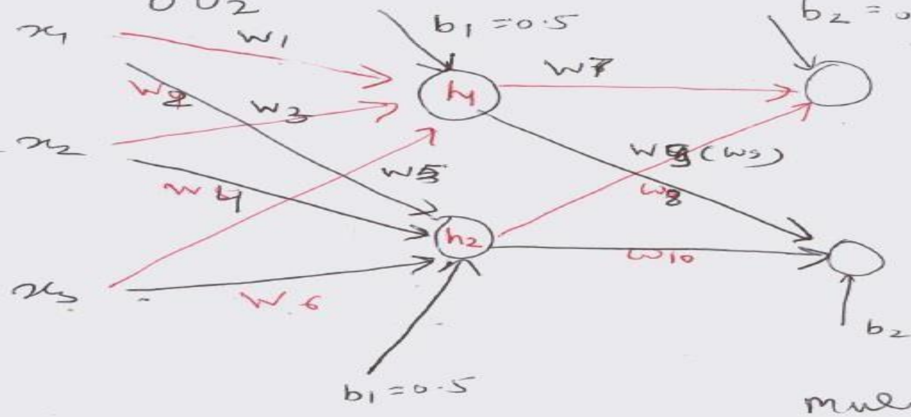
Sum of Square Error : (variance of measured data from the true mean of data)

$$E = \frac{1}{2}\left[(O_1 - t_1)^2 + (O_2 - t_2)^2\right]$$

[Error fw

$$\frac{\partial}{\partial O_1}(E) = (O_1 - t_1)$$

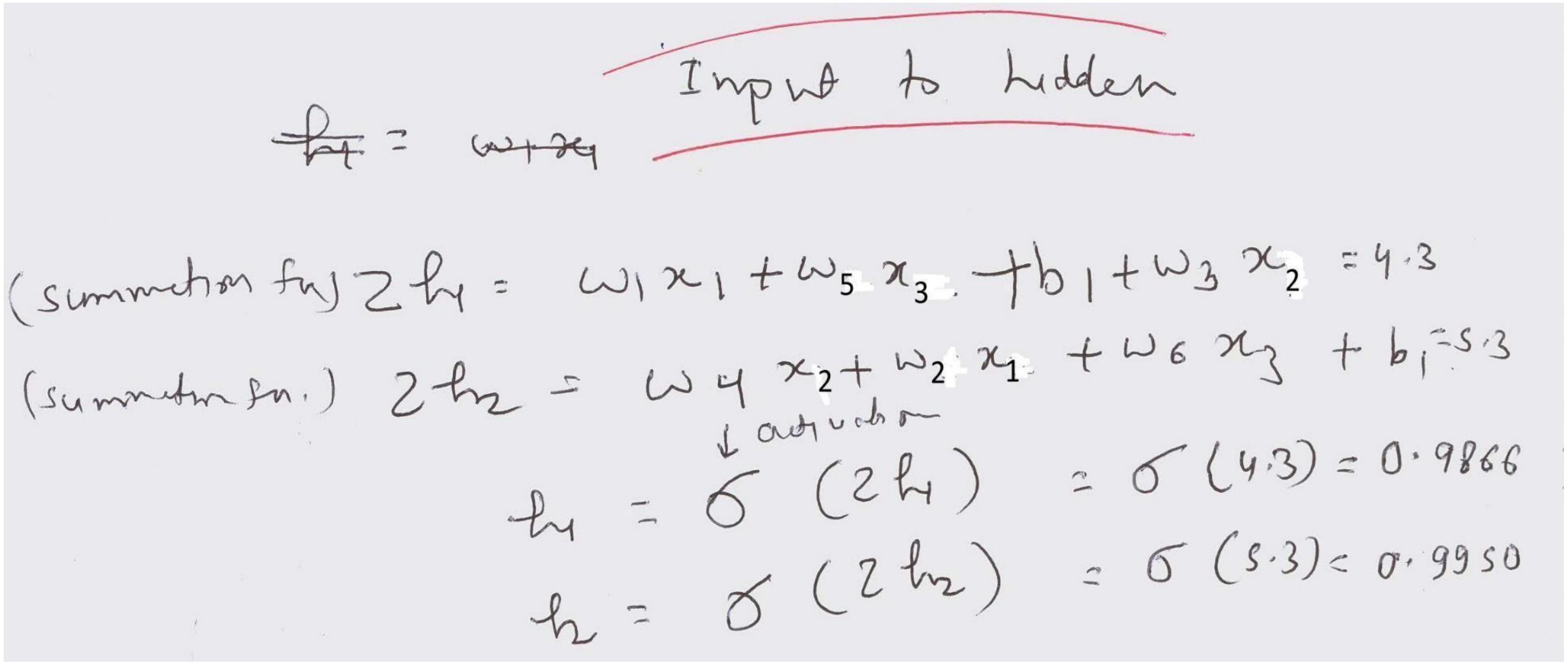
$$\frac{\partial}{\partial O_2}(E) = (O_2 - t_2)$$



h_1 : w_1, w_3, w_5
h_2 = w_2 w_4 w_6

$w_1 = 0.1$	$w_2 = 0.2$
$w_3 = 0.3$	$w_4 = 0.4$
$w_5 = 0.5$	$w_6 = 0.6$
$w_7 = 0.7$	$w_8 = 0.8$
$w_9 = 0.9$	$w_{10} = 0.1$

Multilayer Neural N/W



Input to hidden

~~h_1~~ = ~~$w_1 x_1$~~

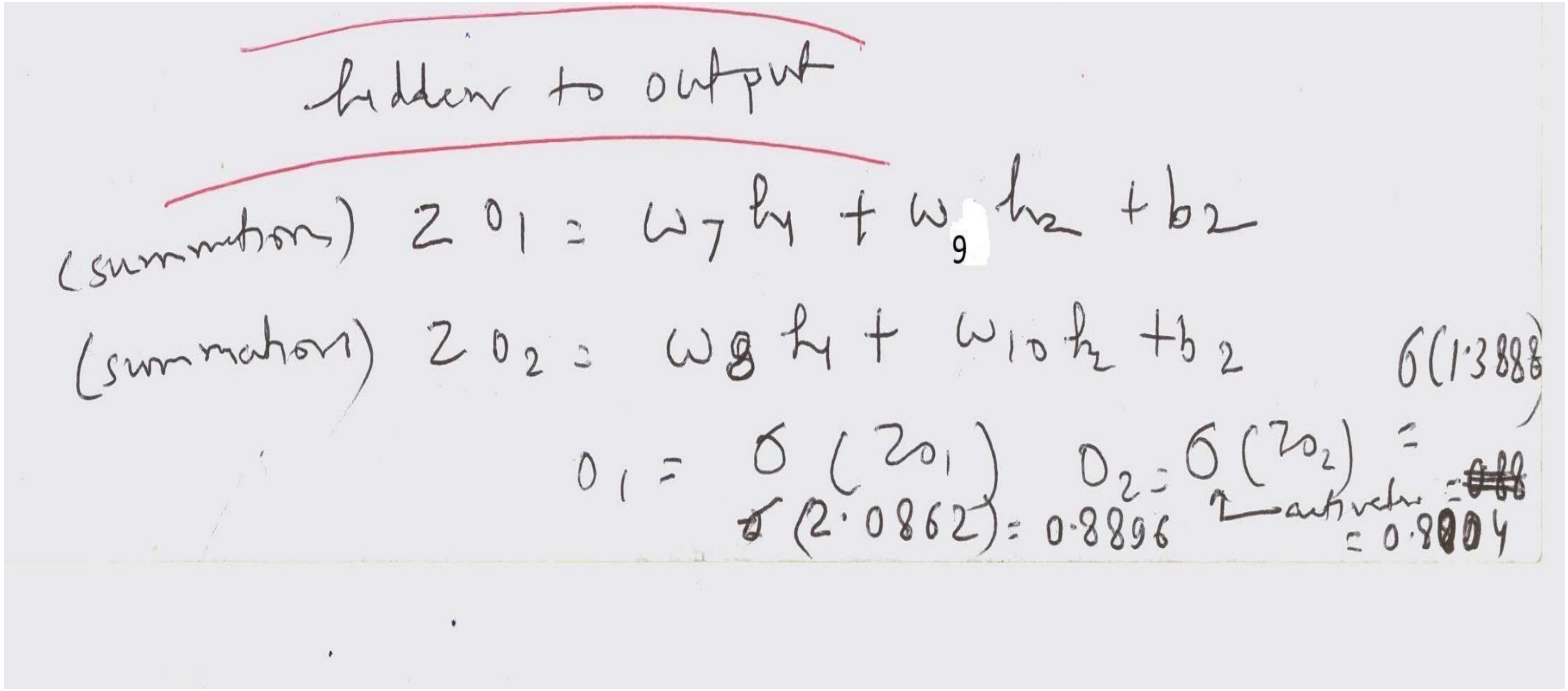
(summation fn) $z_{h_1} = w_1 x_1 + w_5 x_3 + b_1 + w_3 x_2 = 4.3$

(summation fn.) $z_{h_2} = w_4 x_2 + w_2 x_1 + w_6 x_3 + b_1 = 5.3$

$\downarrow$ activation

$h_1 = \sigma(z_{h_1}) = \sigma(4.3) = 0.9866$

$h_2 = \sigma(z_{h_2}) = \sigma(5.3) = 0.9950$



hidden to output

(summation) $z_{o_1} = w_7 h_1 + w_9 h_2 + b_2$

(summation) $z_{o_2} = w_8 h_1 + w_{10} h_2 + b_2$

$o_1 = \sigma(z_{o_1})$ $\quad O_2 = \sigma(z_{o_2}) = \sigma(1.3888) = ~~0.88~~ = 0.8004$

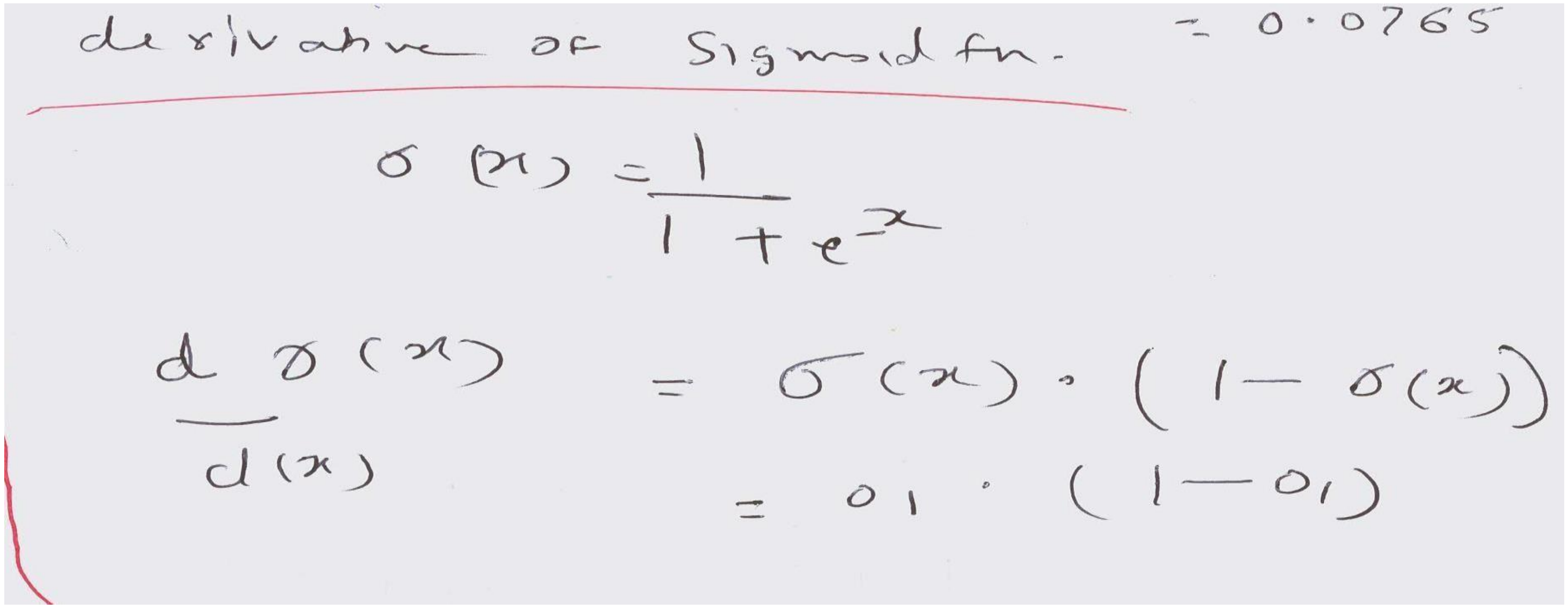
$\sigma(2.0862) = 0.8896$ ← activation

now we need to identify error occured due weights at hidden layer (w_7, w_9, w_8, w_{10})

$$\frac{dE}{dw_7} = \overset{\text{derivation} \downarrow \text{o/p}}{\frac{dE}{do_1}} \times \overset{\text{~~activation~~ summation} \downarrow}{\frac{do_1}{dz_{o1}}} \times \overset{\text{i/p} \downarrow}{\frac{dz_{o1}}{dw_7}}$$ (Chain Rule)

w_7, w_9 : o/p 1
$w_8\ w_{10}$: o/p 2

$$= (o_1 - t_1) \times o_1 \times (1 - o_1) \times h_1$$

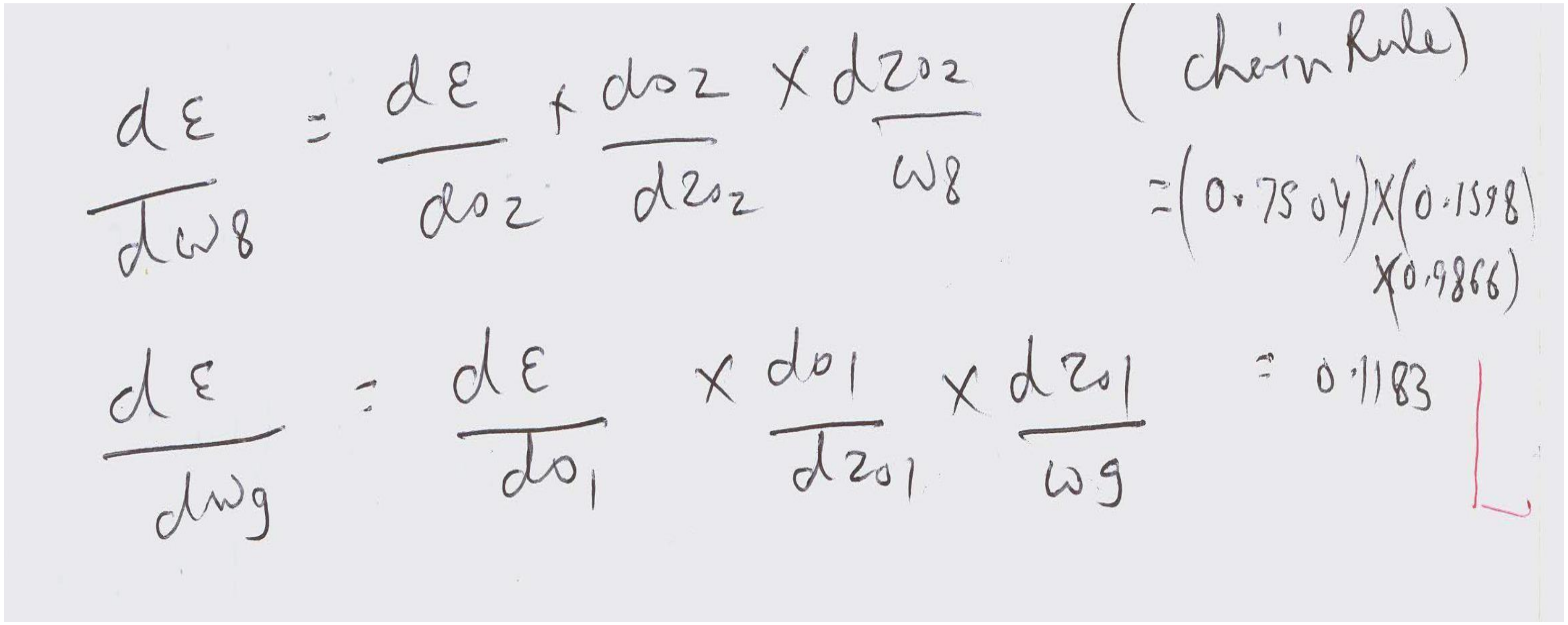


derivative of Sigmoid fn. $= 0.0765$

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

$$\frac{d\,\sigma(x)}{d(x)} = \sigma(x)\cdot(1-\sigma(x))$$

$$= O_1 \cdot (1-O_1)$$

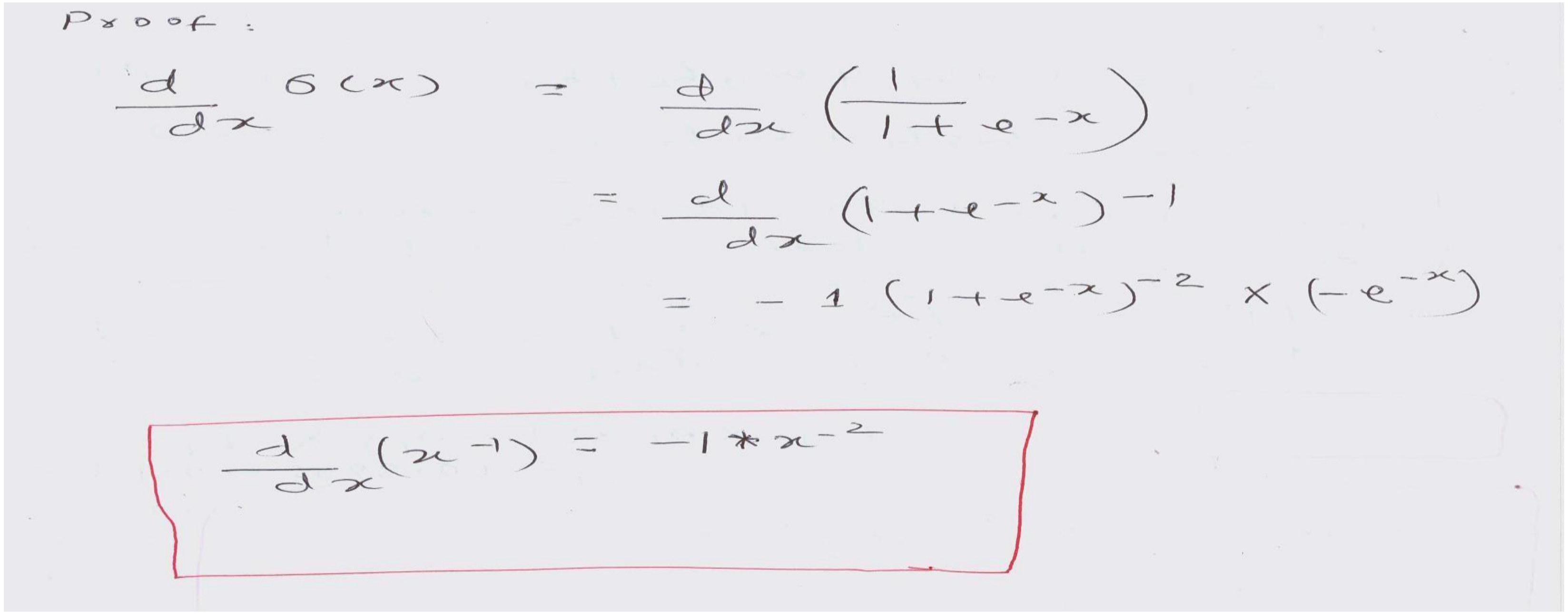


$$\frac{d\varepsilon}{dw8} = \frac{d\varepsilon}{do2} \times \frac{do2}{dzo2} \times \frac{dzo2}{w8}$$

(chain Rule)

$$= (0.7504) \times (0.1598) \times (0.9866)$$

$$\frac{d\varepsilon}{dw9} = \frac{d\varepsilon}{do_1} \times \frac{do1}{dzo1} \times \frac{dzo1}{w9} = 0.1183$$



Proof :

$$\frac{d}{dx}\sigma(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

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

$$= -1\,(1+e^{-x})^{-2} \times (-e^{-x})$$

$$\boxed{\frac{d}{dx}(x^{-1}) = -1 * x^{-2}}$$

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

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

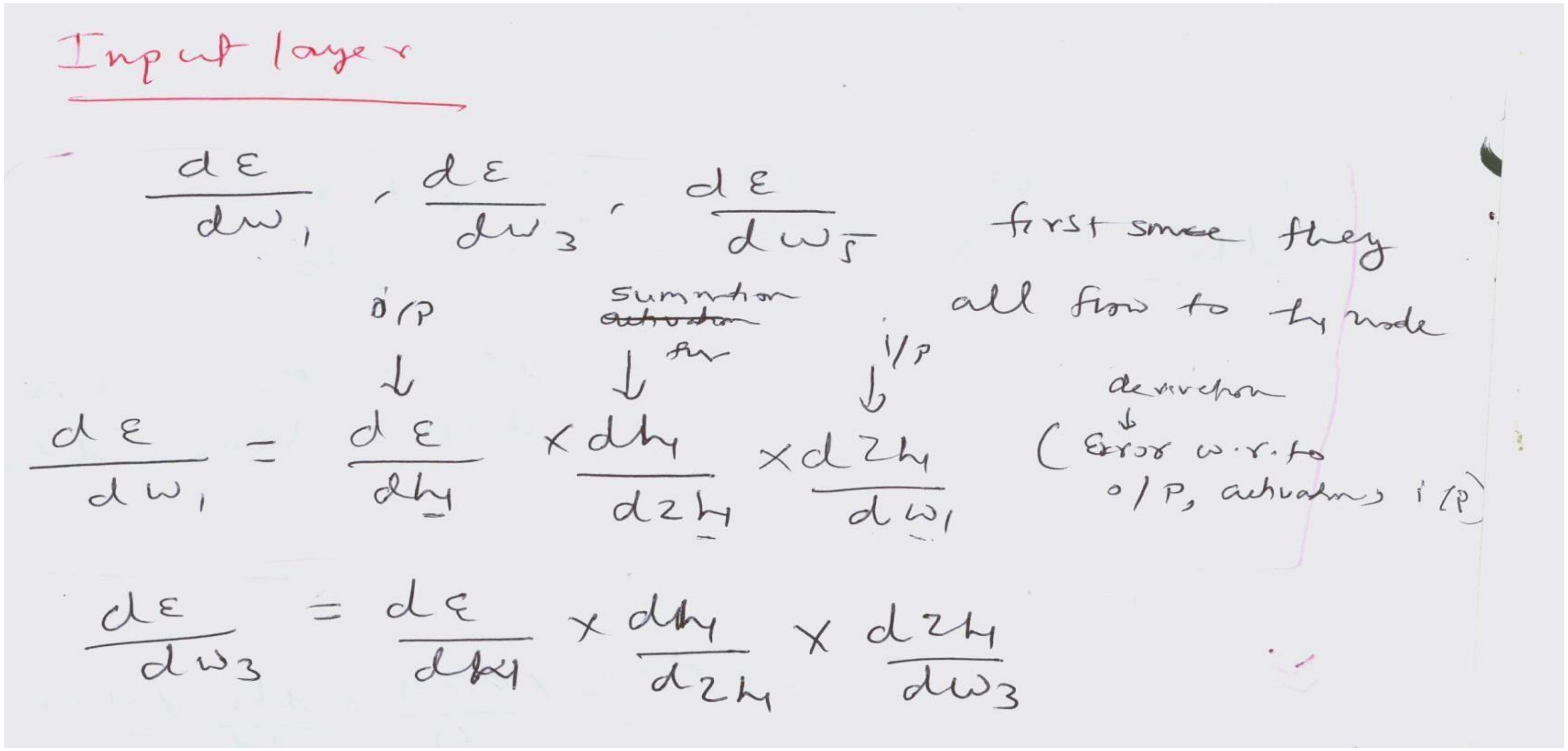
$$= \frac{1}{(1+e^{-x})} \times \frac{e^{-x}+1-1}{1+e^{-x}}$$

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

$$\frac{dE}{dw_{10}} = \frac{dE}{do_2} \times \frac{do_2}{dz_{o_2}} \times \frac{dz_{o_2}}{w_{10}}$$

(optional)

$$\frac{dE}{db_2} = \frac{dE}{do_1} \times \frac{do_1}{dz_{o_1}} \times \frac{dz_{o_1}}{db_2} + \frac{dE}{do_2} \times \frac{do_2}{dz_{o_2}} \times \frac{dz_{o_2}}{db_2}$$



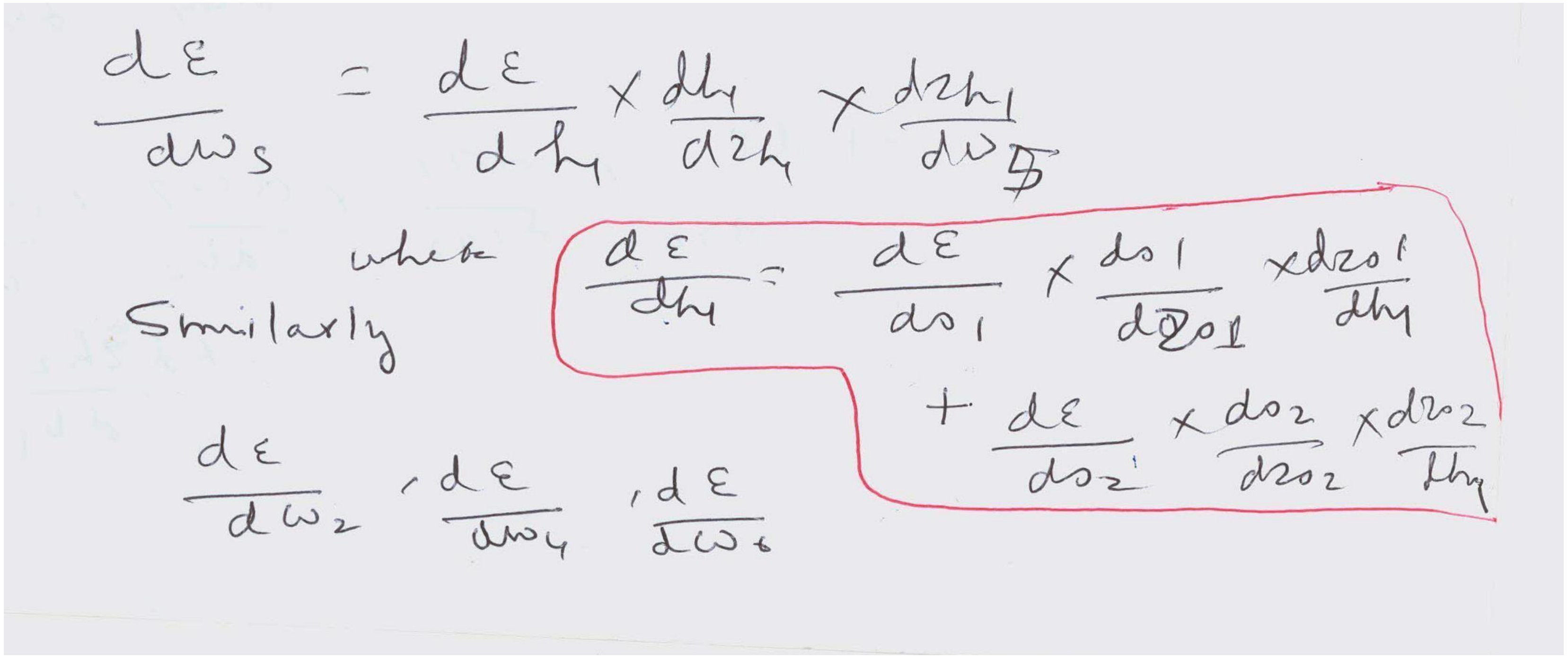
Input layer

$\frac{dE}{dw_1}, \frac{dE}{dw_3}, \frac{dE}{dw_5}$ first since they all flow to h_1 node

$$\frac{dE}{dw_1} = \overset{\overset{\text{o/p}}{\downarrow}}{\frac{dE}{dh_1}} \times \overset{\overset{\text{summation } \cancel{\text{activation}} \text{ fn}}{\downarrow}}{\frac{dh_1}{dz_{h_1}}} \times \overset{\overset{\text{i/p}}{\downarrow}}{\frac{dz_{h_1}}{dw_1}}$$

($\overset{\overset{\text{derivation}}{\downarrow}}{\text{Error}}$ w.r.to o/p, activation, i/p)

$$\frac{dE}{dw_3} = \frac{dE}{dh_1} \times \frac{dh_1}{dz_{h_1}} \times \frac{dz_{h_1}}{dw_3}$$



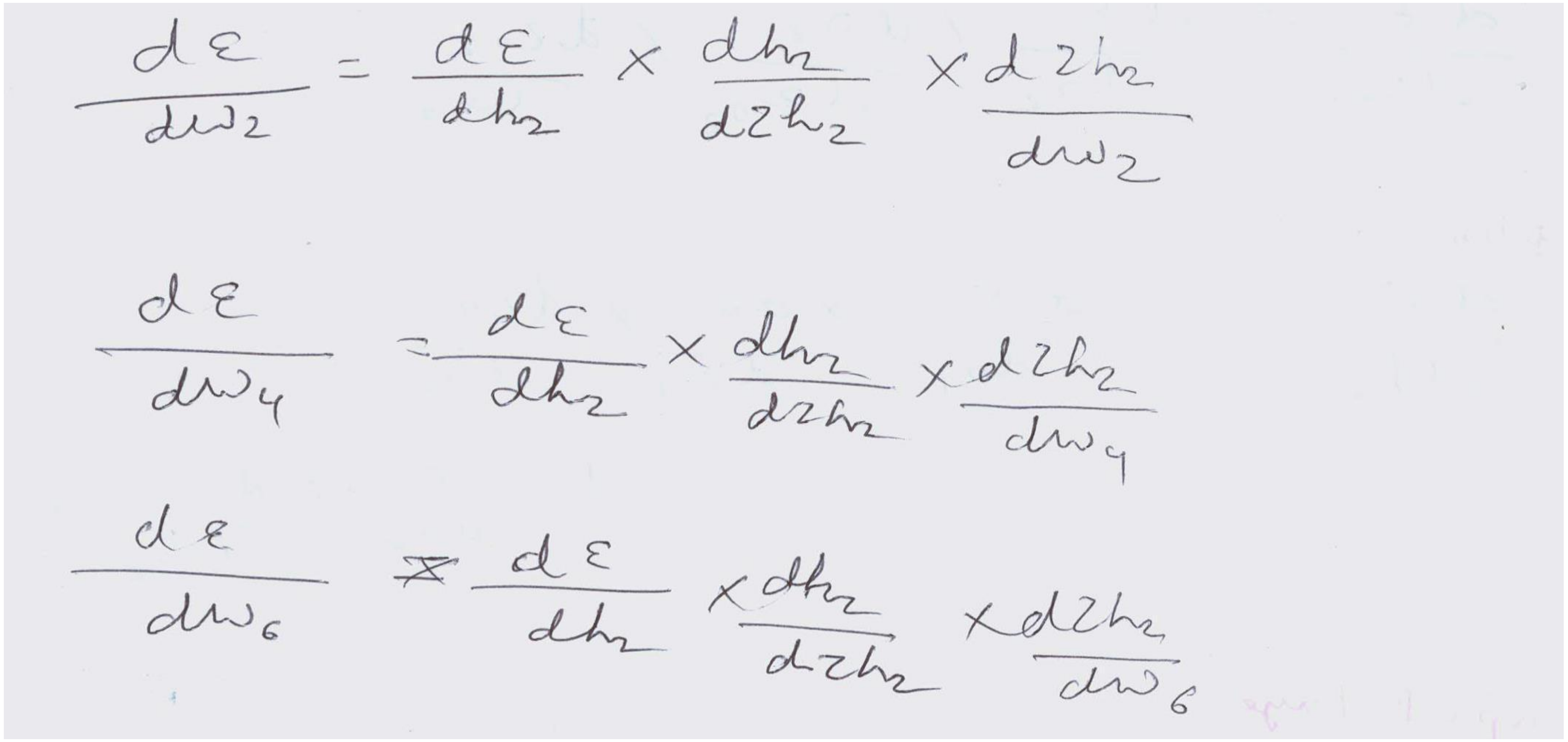
$$\frac{dE}{dw_5} = \frac{dE}{dh_1} \times \frac{dh_1}{dzh_1} \times \frac{dzh_1}{dw_5}$$

where

$$\boxed{\frac{dE}{dh_1} = \frac{dE}{do_1} \times \frac{do_1}{dzo_1} \times \frac{dzo_1}{dh_1} + \frac{dE}{do_2} \times \frac{do_2}{dzo_2} \times \frac{dzo_2}{dh_1}}$$

Similarly

$$\frac{dE}{dw_2}, \frac{dE}{dw_4}, \frac{dE}{dw_6}$$



$$\frac{dE}{dw_2} = \frac{dE}{dh_2} \times \frac{dh_2}{dZh_2} \times \frac{dZh_2}{dw_2}$$

$$\frac{dE}{dw_4} = \frac{dE}{dh_2} \times \frac{dh_2}{dZh_2} \times \frac{dZh_2}{dw_4}$$

$$\frac{dE}{dw_6} = \frac{dE}{dh_2} \times \frac{dh_2}{dZh_2} \times \frac{dZh_2}{dw_6}$$

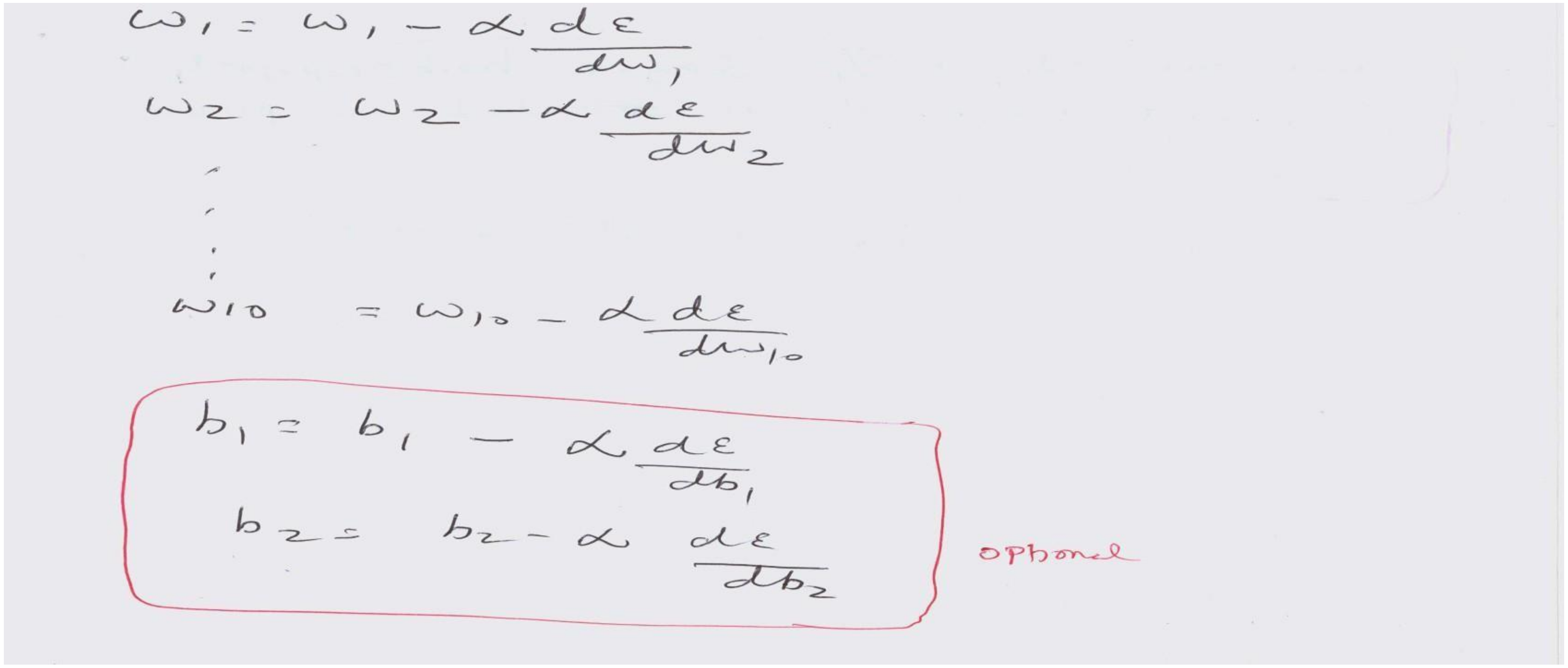
$$\frac{dE}{dh_2} = \cancel{\frac{dE}{dh_2}} \quad \frac{dE}{do_1} \times \frac{do_1}{dzo_1} \times \frac{dzo_1}{dh_2}$$

$$+ \frac{dE}{do_2} \times \frac{do_2}{dzo_2} \times \frac{dzo_2}{dh_2}$$

(optmal)

$$\frac{dE}{db_1} = \frac{dE}{do_1} \times \frac{do_1}{dzo_1} \times \frac{dzo_1}{dh_1} \times \frac{dh_1}{dzh_1} \times \frac{dzh_1}{db_1}$$

$$+ \frac{dE}{do_2} \times \frac{do_2}{dzo_2} \times \frac{dzo_2}{dh_2} \times \frac{dh_2}{dzh_2} \times \frac{dzh_2}{db_1}$$



$$w_1 = w_1 - \alpha \frac{dE}{dw_1}$$

$$w_2 = w_2 - \alpha \frac{dE}{dw_2}$$

$$\vdots$$

$$w_{10} = w_{10} - \alpha \frac{dE}{dw_{10}}$$

$$\boxed{\begin{aligned} b_1 &= b_1 - \alpha \frac{dE}{db_1} \\ b_2 &= b_2 - \alpha \frac{dE}{db_2} \end{aligned}}$$

optional

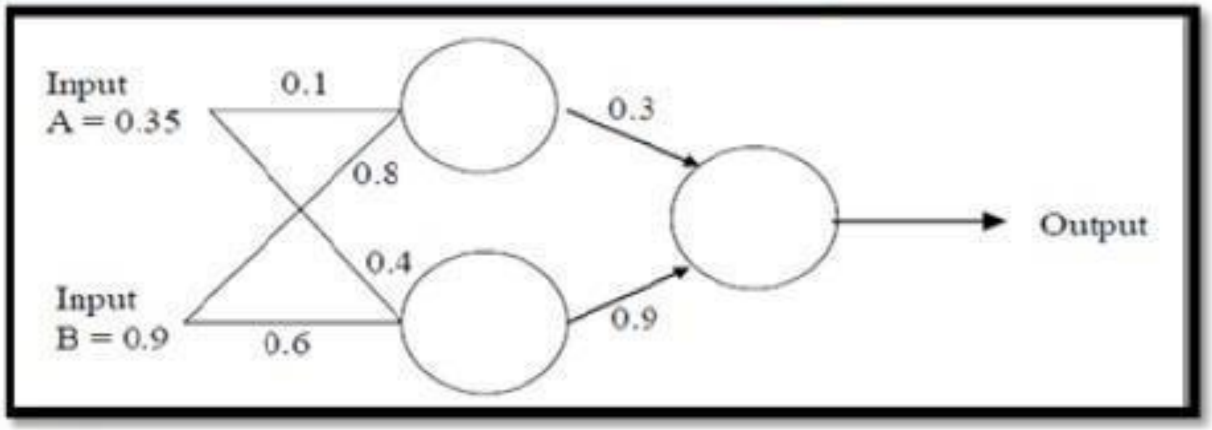
Tutorial

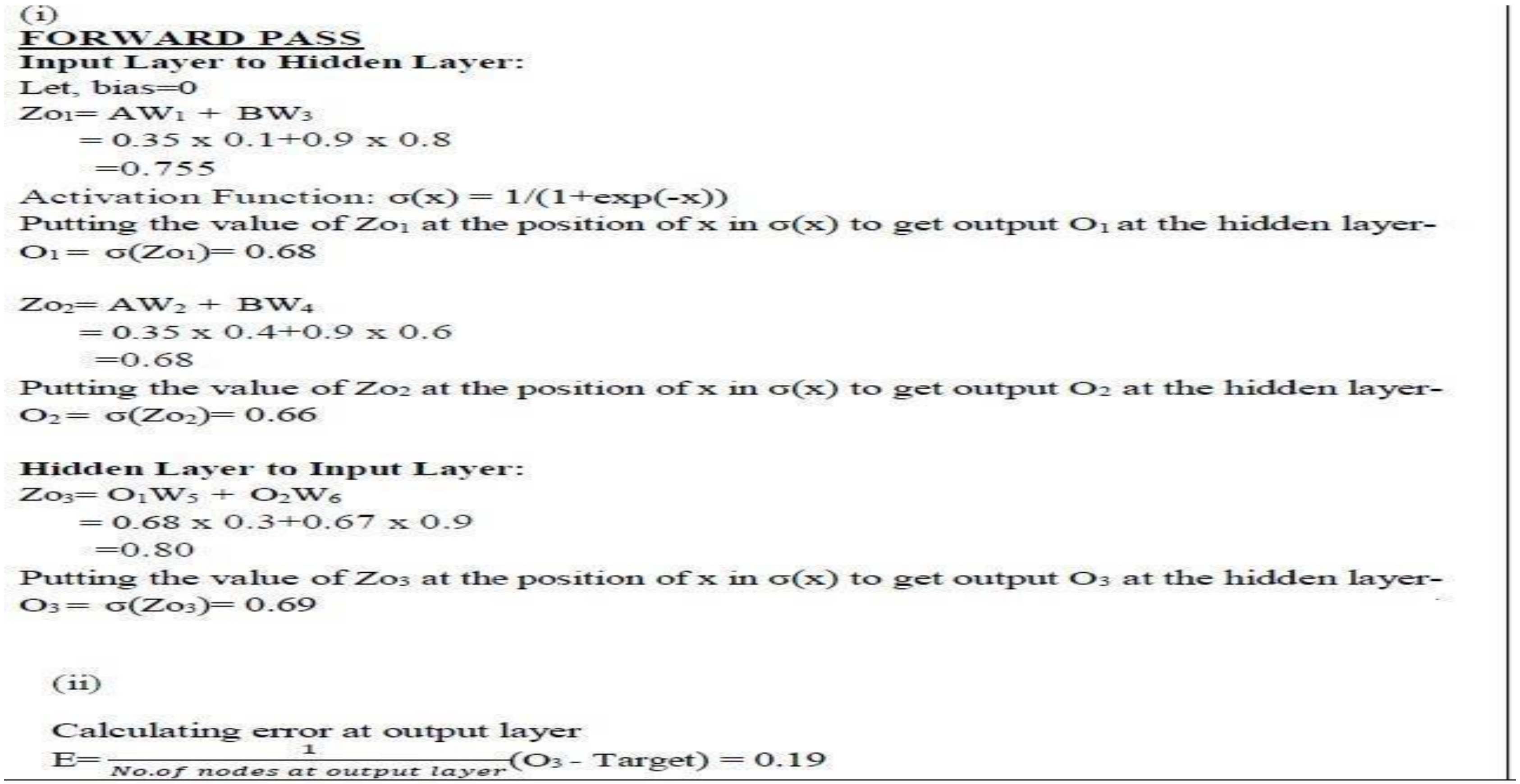
Consider the backpropagation neural network as shown below, assume that the neurons have a sigmoid activation function, do the following:

(i) Perform a forward pass on the network.

(ii) Perform a reverse pass (training) once (target = 0.5, ∝=**1**) and find the new weights.

(iii). Perform a further forward pass and comment on the result.





(i)

FORWARD PASS

Input Layer to Hidden Layer:

Let, bias=0

$Zo_1 = AW_1 + BW_3$

$= 0.35 \times 0.1 + 0.9 \times 0.8$

$= 0.755$

Activation Function: $\sigma(x) = 1/(1+\exp(-x))$

Putting the value of Zo_1 at the position of x in $\sigma(x)$ to get output O_1 at the hidden layer-

$O_1 = \sigma(Zo_1) = 0.68$

$Zo_2 = AW_2 + BW_4$

$= 0.35 \times 0.4 + 0.9 \times 0.6$

$= 0.68$

Putting the value of Zo_2 at the position of x in $\sigma(x)$ to get output O_2 at the hidden layer-

$O_2 = \sigma(Zo_2) = 0.66$

Hidden Layer to Input Layer:

$Zo_3 = O_1W_5 + O_2W_6$

$= 0.68 \times 0.3 + 0.67 \times 0.9$

$= 0.80$

Putting the value of Zo_3 at the position of x in $\sigma(x)$ to get output O_3 at the hidden layer-

$O_3 = \sigma(Zo_3) = 0.69$

(ii)

Calculating error at output layer

$E = \frac{1}{No. of\ nodes\ at\ output\ layer}(O_3 - \text{Target}) = 0.19$

<u>REVERSE PASS</u>

Finding the error at hidden layer due to weight W_5, W_6

$$\frac{dE}{dW5} = \frac{dE}{dO3} \text{ x } \frac{dO3}{dZO3} \text{ x } \frac{dZO3}{dW5} = 0.0276$$

$$\frac{dE}{dW6} = \frac{dE}{dO3} \text{ x } \frac{dO3}{dZO3} \text{ x } \frac{dZO3}{dW6} = 0.027$$

Calculating error at the input layer

$$\frac{dE}{dW1} = \frac{dE}{dO1} \text{ x } \frac{dO1}{dZO1} \text{ x } \frac{dZO1}{dW1} = 0.00092$$

$$\frac{dE}{dW2} = \frac{dE}{dO2} \text{ x } \frac{dO2}{dZO2} \text{ x } \frac{dZO2}{dW2} = 0.0028$$

$$\frac{dE}{dW3} = \frac{dE}{dO1} \text{ x } \frac{dO1}{dZO1} \text{ x } \frac{dZO1}{dW3} = 0.00238$$

$$\frac{dE}{dW4} = \frac{dE}{dO2} \text{ x } \frac{dO2}{dZO2} \text{ x } \frac{dZO2}{dW4} = 0.0073$$

Update the weight, $\alpha = 1$

$W_1 = W_1 - \alpha \frac{dE}{dW1} = 0.1 - 0.00092 = 0.099$

$W_2 = W_2 - \alpha \frac{dE}{dW2} = 0.4 - 0.0028 = 0.397$

$W_3 = W_3 - \alpha \frac{dE}{dW3} = 0.8 - 0.00238 = 0.797$

$W_4 = W_4 - \alpha \frac{dE}{dW4} = 0.6 - 0.0073 = 0.592$

$W_5 = W_5 - \alpha \frac{dE}{dW5} = 0.3 - 0.0276 = 0.272$

$W_6 = W_6 - \alpha \frac{dE}{dW6} = 0.9 - 0.027 = 0.873$

(iii)

FURTHER FORWARD PASS

Input Layer to Hidden Layer:

$Zo_1 = AW_1 + BW_3$

$= 0.35 \times 0.099 + 0.9 \times 0.797$

$= 0.75195$

Activation Function: $\sigma(x) = 1/(1+\exp(-x))$

Putting the value of Zo_1 at the position of x in $\sigma(x)$ to get output O_1 at the hidden layer-

$O_1 = \sigma(Zo_1) = 0.679$

$Zo_2 = AW_2 + BW_4$

$= 0.35 \times 0.397 + 0.9 \times 0.592$

$= 0.67175$

Putting the value of Zo_2 at the position of x in $\sigma(x)$ to get output O_2 at the hidden layer-

$O_2 = \sigma(Zo_2) = 0.662$

Hidden Layer to Output Layer:

$Zo_3 = O_1W_5 + O_2W_6$

$= 0.679 \times 0.272 + 0.662 \times 0.873$

$= 0.7626$

Putting the value of Zo_3 at the position of x in $\sigma(x)$ to get output O_3 at the hidden layer-

$O_3 = \sigma(Zo_3) = 0.681$

Now, we will find the error at output layer,

$E = (O_3 - E) = 0.18205$

When to Consider Neural Network

- Input is high-dimensional discrete or real-valued (e.g., raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is *unimportant*

Examples:

- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Financial prediction

THANK YOU