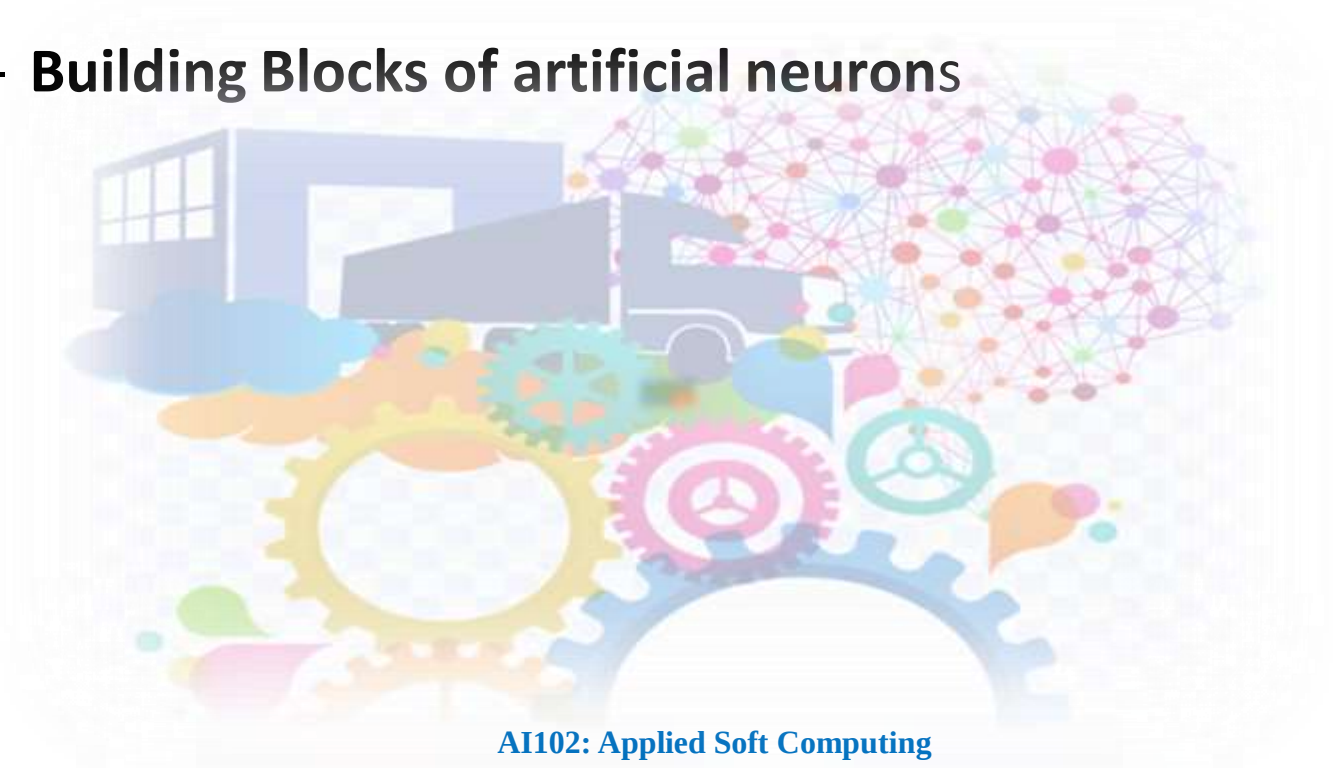
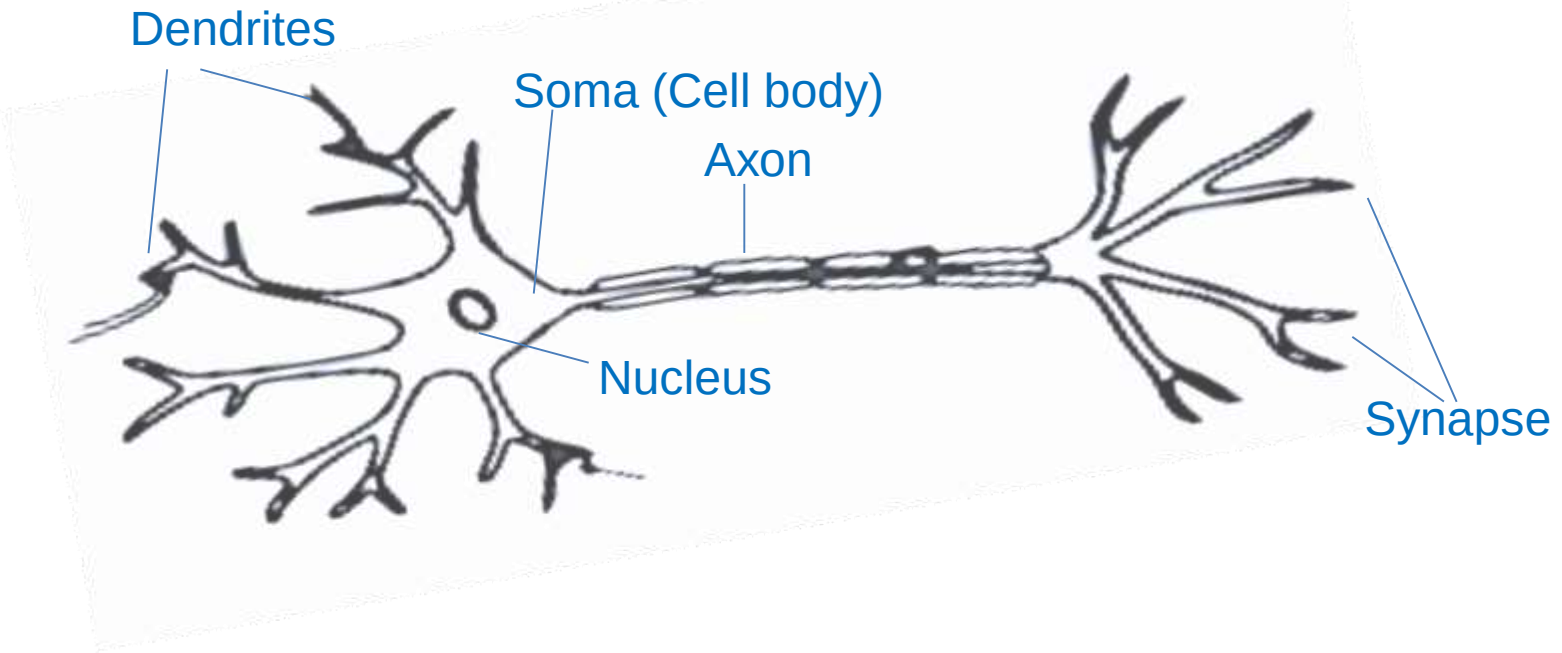
Today's Topics

- **Biological Neural Network and Artificial Neural Network (ANN)**
- **ANN Architecture**
- **Building Blocks of artificial neurons**

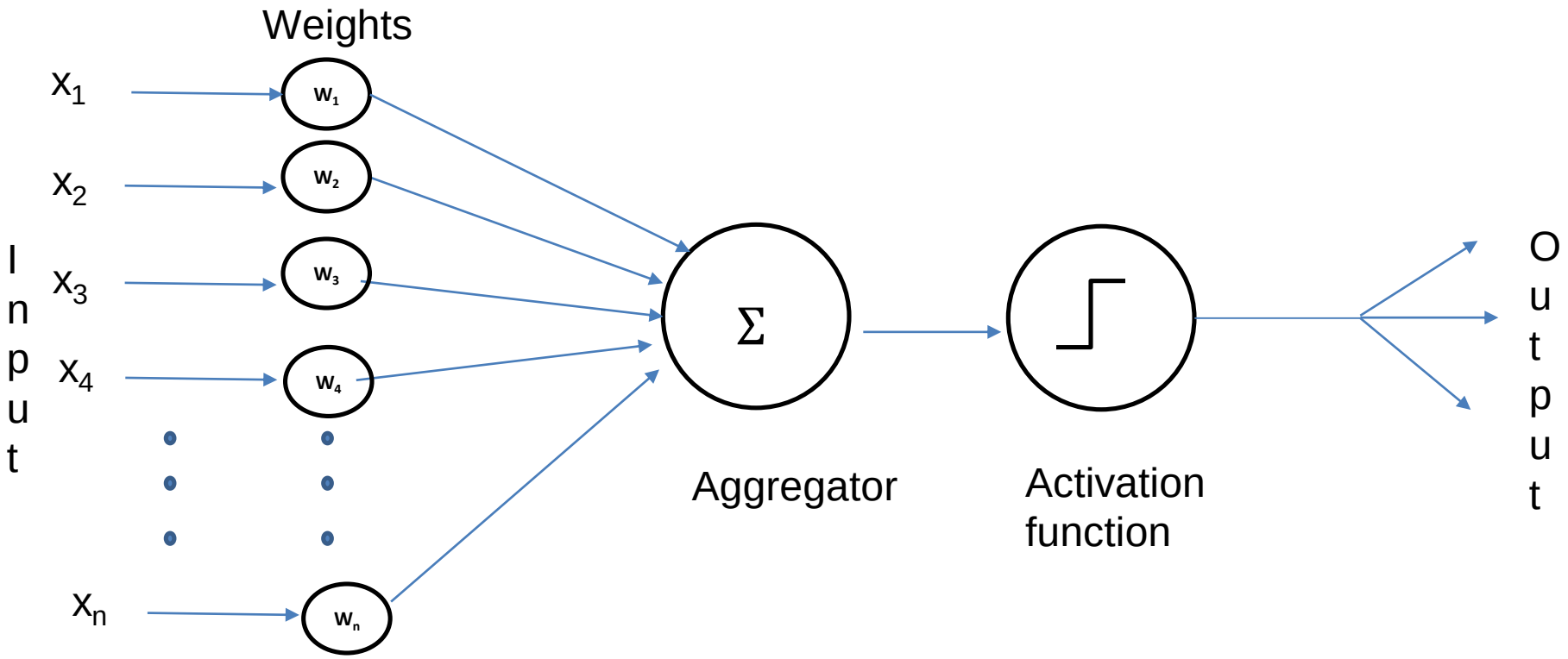


Biological Neuron



Dendrite: responsible for getting incoming signals from outside
Soma : cell body responsible for the processing of input signals and decision making to fire an output signal
Axon: connection for transmitting processed signals from neuron to relevant cells
Synapse: connection between an axon and dendrites of other neurons

Artificial Neuron (Perceptron)



Functions of Perceptron:

1.Takes inputs from the input layer

2.Weighs them separately

3. Does sum of weighted inputs

4.Pass this sum through a nonlinear function to produce output.

Artificial Neurons vs Biological Neurons

CHITKARA UNIVERSITY

Artificial Neural Network (ANN)	Biological Neural Network (BNN)
Processing speed is fast as compared to Biological Neural Network. Cycle time for execution is in nanoseconds.	They are slow in processing information. Cycle time for execution is in milliseconds.
It can perform massive parallel operations simultaneously like BNN.	It can perform massive parallel operations simultaneously.
Size and complexity depends on the application chosen but it is less complex than BNN.	Size and complexity of BNN is more than ANN with 10^{11} neurons and 10^{15} interconnections.
Information is stored in contiguous memory locations.	Information is stored in interconnections or in synapse strength.

Artificial Neurons vs Biological Neurons

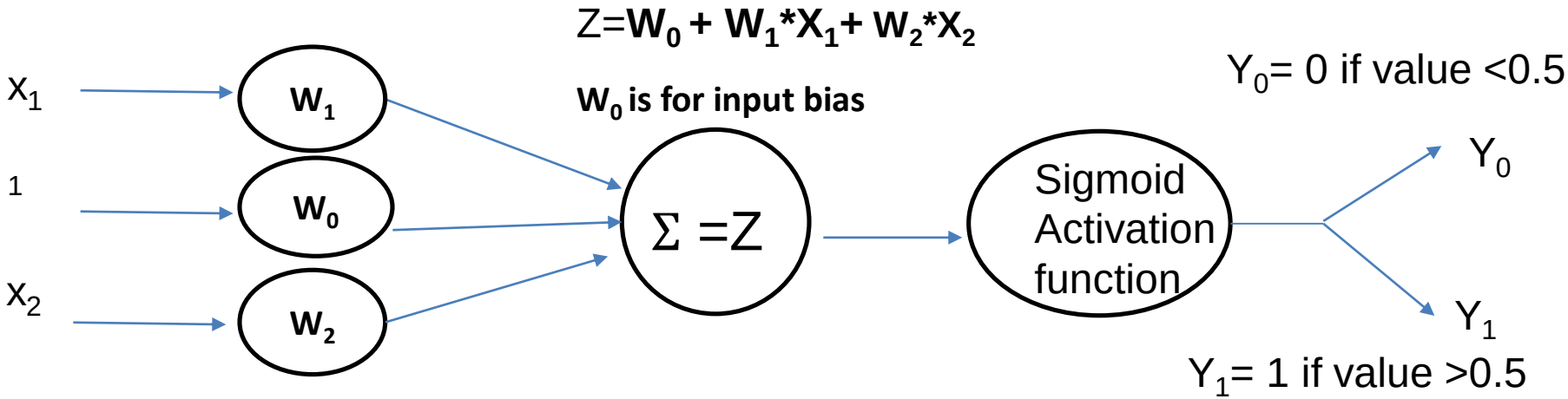
CHITKARA UNIVERSITY

Artificial Neural Network (ANN)	Biological Neural Network (BNN)
To store new information the old information is deleted if there is shortage of storage.	Any new information is stored in interconnection and the old information is stored with lesser strength.
There is no fault tolerance in ANN. The corrupted information cannot be processed.	It has fault tolerance capability. It can store and retrieve information even if the interconnection is disconnected.
The control unit processes the information.	The chemical present in neurons does the processing.

Implementing Logic gate through ANN

CHITKARA UNIVERSITY

OR GATE



X_1	X_2	Z	Desired Output	W_0	W_1	W_2
0	0	W_0	Y_0 (0)	-1	Any value	Any Value
0	1	$W_0 + W_2*X_2$	Y_1 (1)	-1	Any Value	2
1	0	$W_0 + W_1*X_1$	Y_1 (1)	-1	2	2
1	1	$W_0 + W_1*X_1 + W_2*X_2$	Y_1 (1)	-1	2	2

Basic entities of ANN Models

There are three basic entities of ANN Models

- Interconnections
- Learning rules for adjusting connection weights
- Activation functions

Interconnections

Single neuron concept can be extended to multiple neurons to form layers eg. Input Layer or Output Layer

The arrangement of neurons in the form of layers is called network architecture.

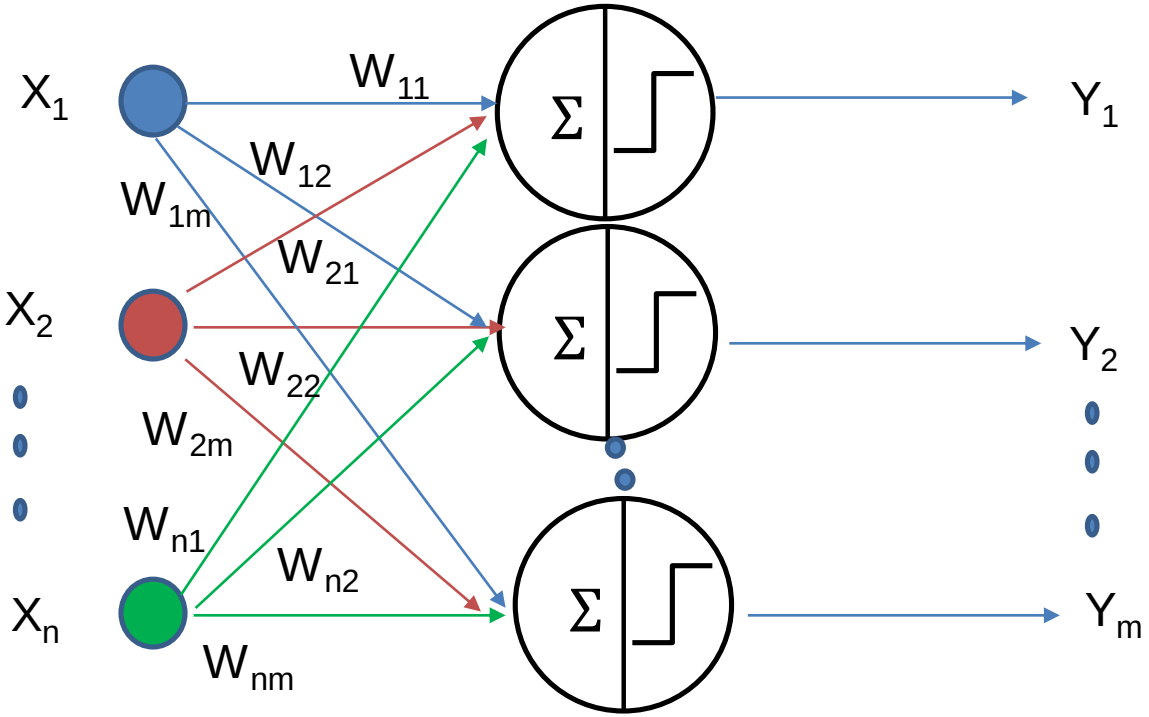
The basic types of architectures are:

- Single Layer Feed Forward
- Multiple Layer Feed Forward
- Recurrent Network

Single Layer Feed Forward ANN

- Single Layer Feed Forward ANN

  The implementation of logic gate through ANN can be extended to multiple neurons



Single Layer Feed Forward ANN

- It has single output layer
- The input is only fed forward
- There is no feedback, no neuron in the output layer is an input to same or the preceding layer (No backward feed)
- Input and Output Layers are boundary layers so cannot be called true layers

Single Layer Feed Forward ANN

Input Matrix = $[X_1 \;\; X_2 \;\ldots\; X_n \;]$

Weight Matrix = $\begin{bmatrix} W_{11} & W_{12} \ldots W_{1m} \\ W_{21} & W_{22} \ldots W_{2m} \\ & \cdot \\ & \vdots \\ W_{n1} & W_{n2} \ldots W_{nm} \end{bmatrix}$

Output Matrix = $[Y_1 \;\; Y_2 \ldots Y_m \;]$

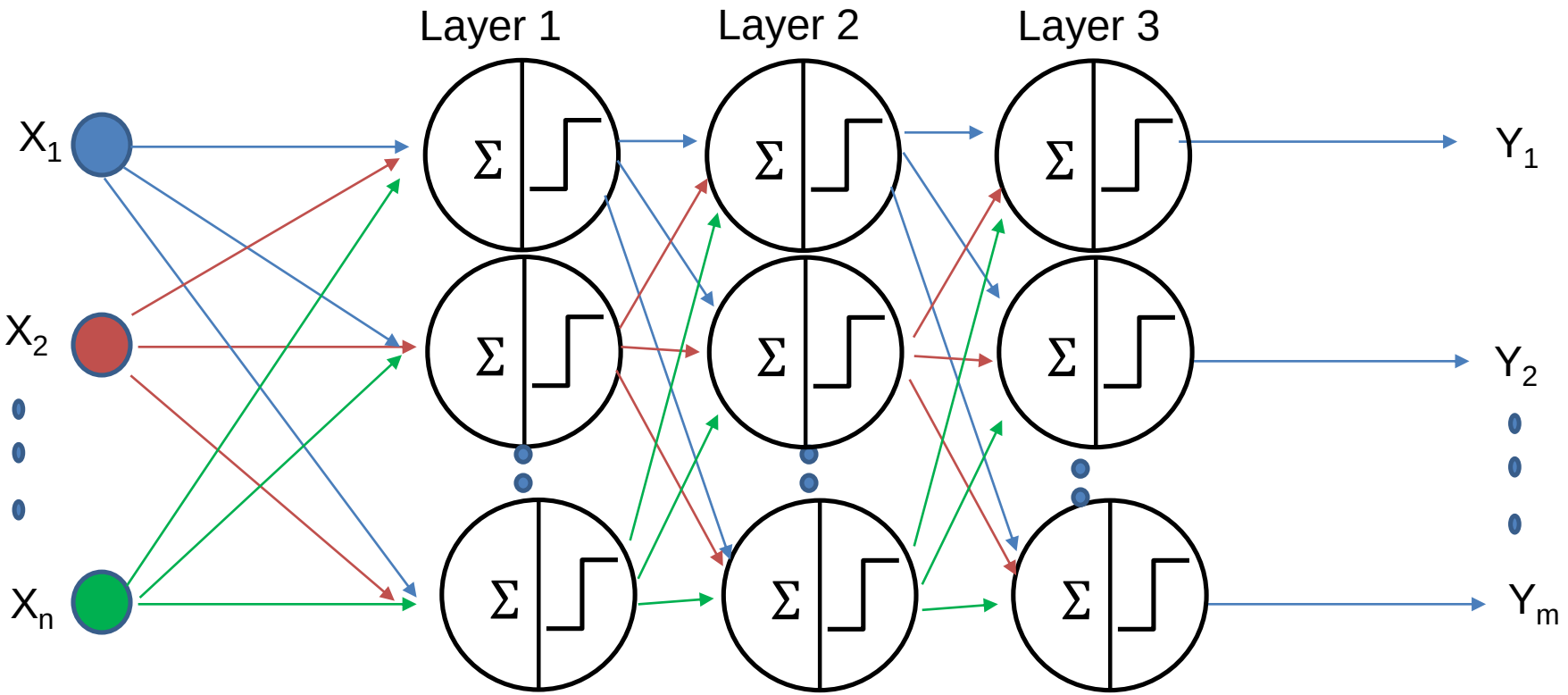
K_{th} output = $Y_k = \sum_{i=1}^{n} W_{ik*} X_i + \theta_k$ *where* θ *is the bias*

Multi Layer Feed Forward ANN

- It has multiple layers.
- Apart from input and output layers we have other layers between the two which are hidden.
- Intermediary computation is done by hidden layers.
- There is no feedback, no neuron in the output layer is an input to same or the preceding layer (No backward feed)
- Input and Output Layers are boundary layers so cannot be called true layers

Multiple Layer Feed Forward ANN

Multilayer ANN with input layer output layer and hidden layer



Multi Layer Feed Forward ANN

- Single Layer ANN is extended to Multi Layer ANN
- Input layer contains ***n*** neurons, hidden layer contains ***p*** neurons and output layer contains ***m*** neurons
- It is ***n-p-m*** Multilayer Feed Forward ANN

Multi Layer Feed Forward ANN

Input Matrix fed to first layer = $[X_1 \ X_2 \ \dots \ X_n \ ]$

Weight Matrix fed to first layer W1, to second layer W2 and to third layer is W3.

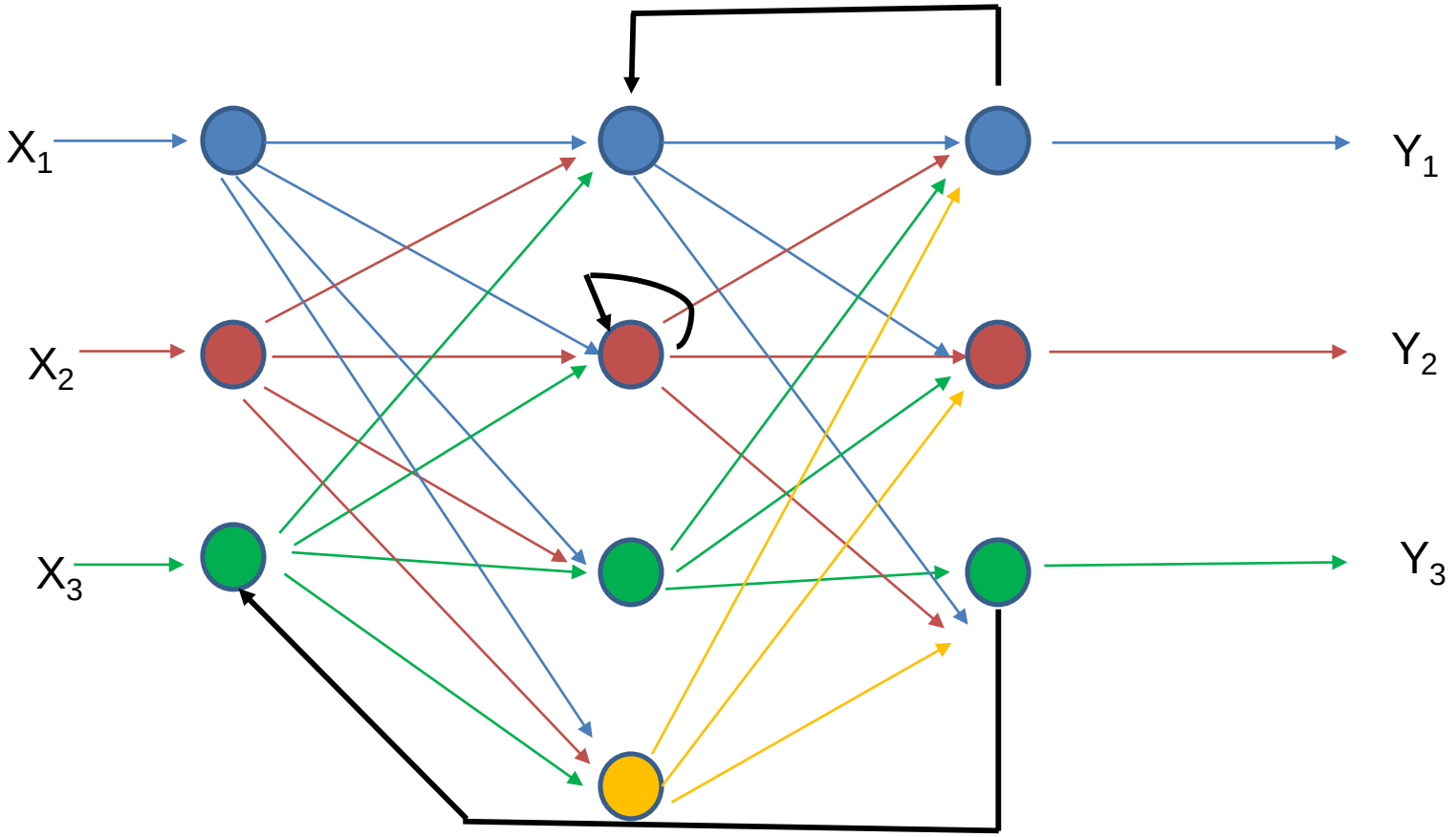
Output Matrix after layer 3 is = $[Y_1 \ Y_2 \ \dots Y_m \ ]$

Layer 2 is hidden layer.

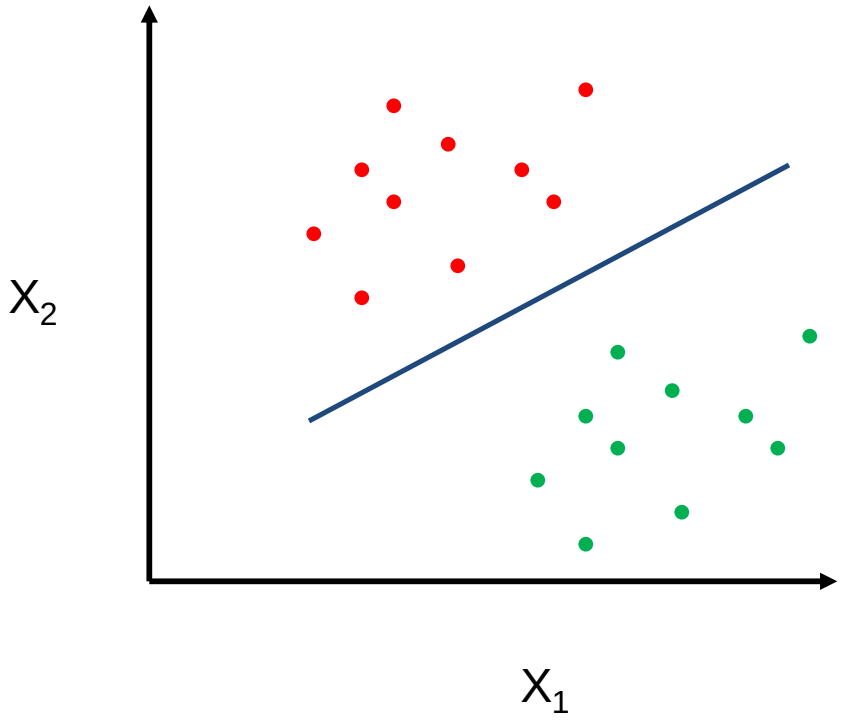
The output of any layer can be found accordingly.

Recurrent ANN

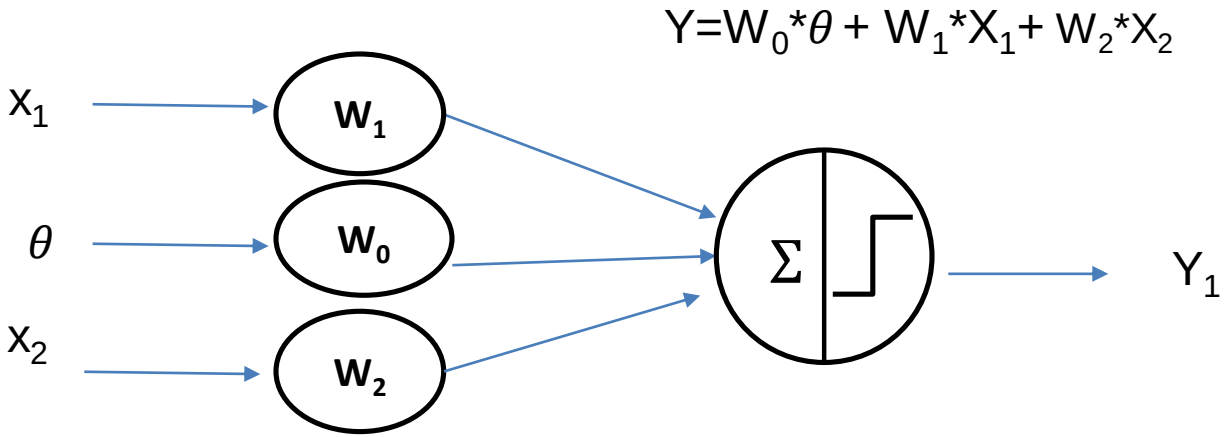
- It has atleast one feedback loop in atleast one layer.
- There can be neurons with self feedback loop as well.



Linearly Separable Data

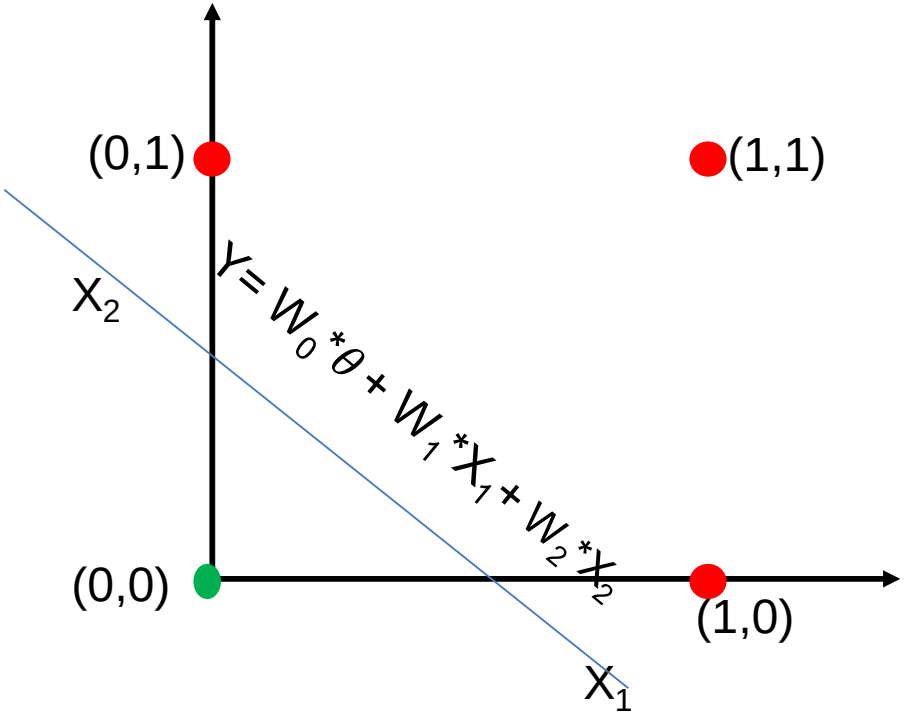


Implementing Logic gate through ANN



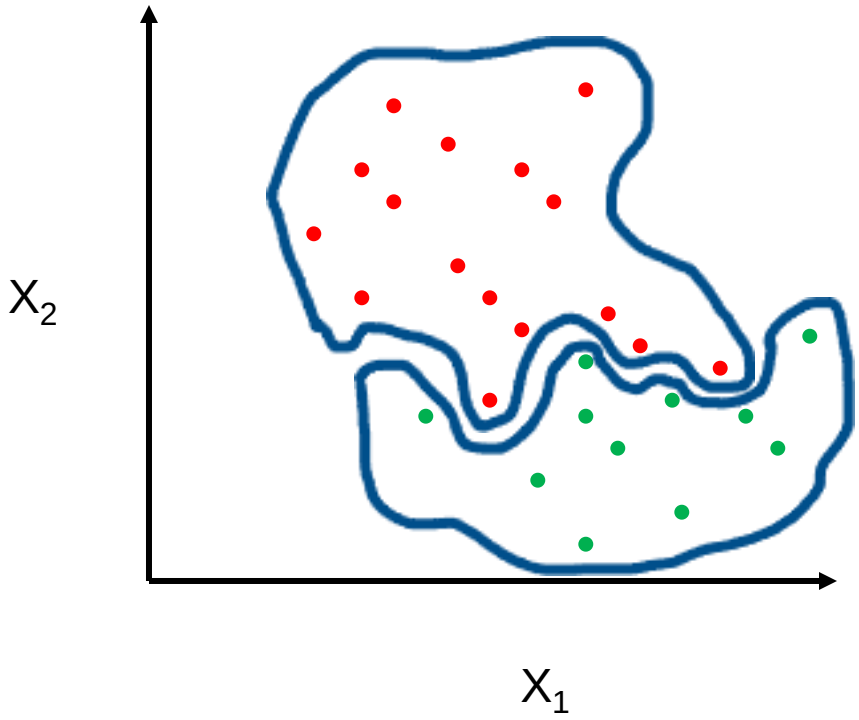
Linearly Separabale Data

OR GATE



Requires Single Layer Architecture

Non-Linearly Separable Data

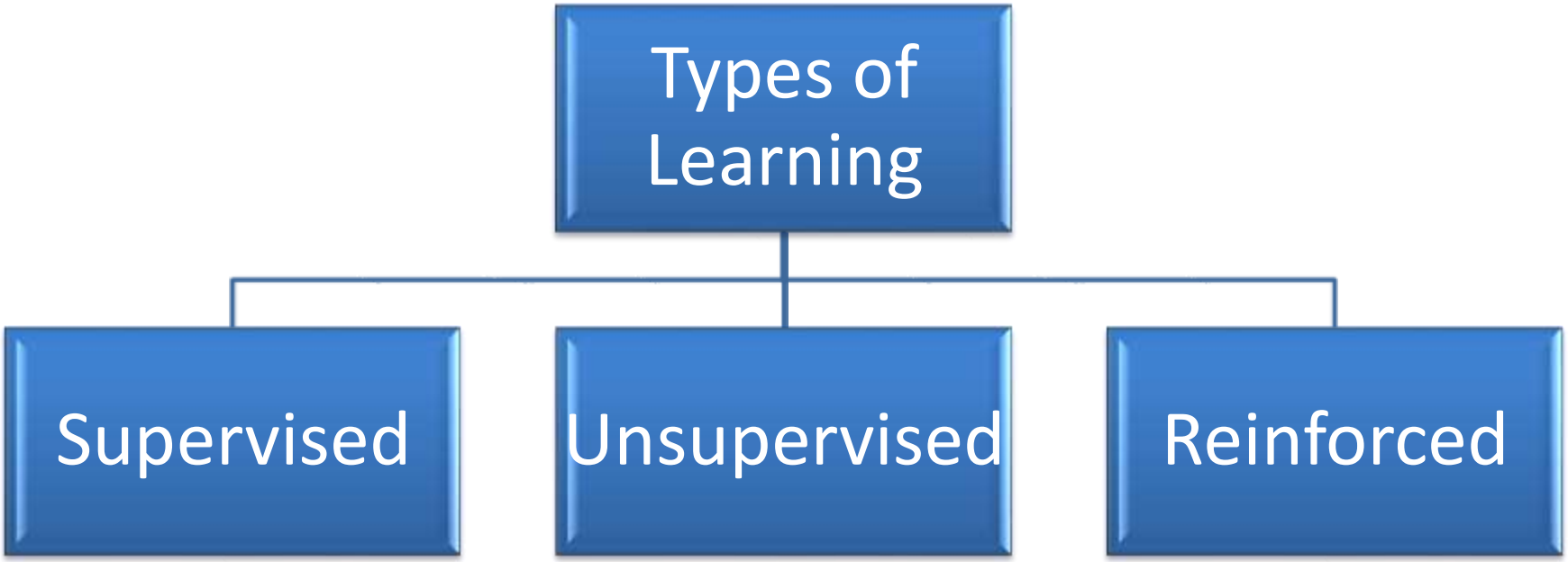


Requires Multi Layer Architecture

Learning in ANN

- Deciding number of hidden layers and number of neurons (perceptrons) in each layer
- Deciding and updating weights, biases and other parameters.
- Optimization of parameters

Types of Learning



Supervised Learning

- Learning performed with the help of supervisor.
- Each input vector has corresponding desired output vector.
- The combination of input vector and desired output vector is training pair.
- Error is detected by comparing actual output with the target output.
- Adjust weights accordingly.
- Network trained by this method needs supervisor for error minimization.

Unsupervised Learning

- Learning performed without the supervisor.
- Output vector is not known
- No error detection
- Adaptation to input patterns

Reinforced Learning

- Similar to supervised learning.
- Exact output vector is not known but critic information is there.
- Learning based on the critic information is reinforced learning.
- The feedback sent is reinforced feedback.

Activation Functions

- Activation function helps to obtain exact output.
- It can of linear or non-linear type.

Types of Activation Functions:

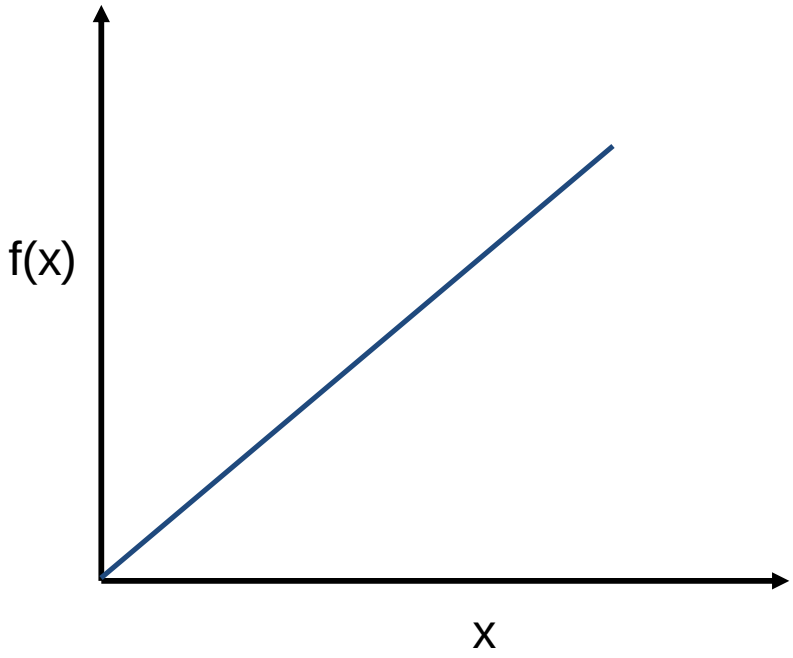
- Identity function
- Binary Step function
- Bipolar Step function
- Sigmoidal function
- Ramp function

Activation Functions

- **Identity function**

  It's a linear function. The output is same as input.

  f(x) = x for all x



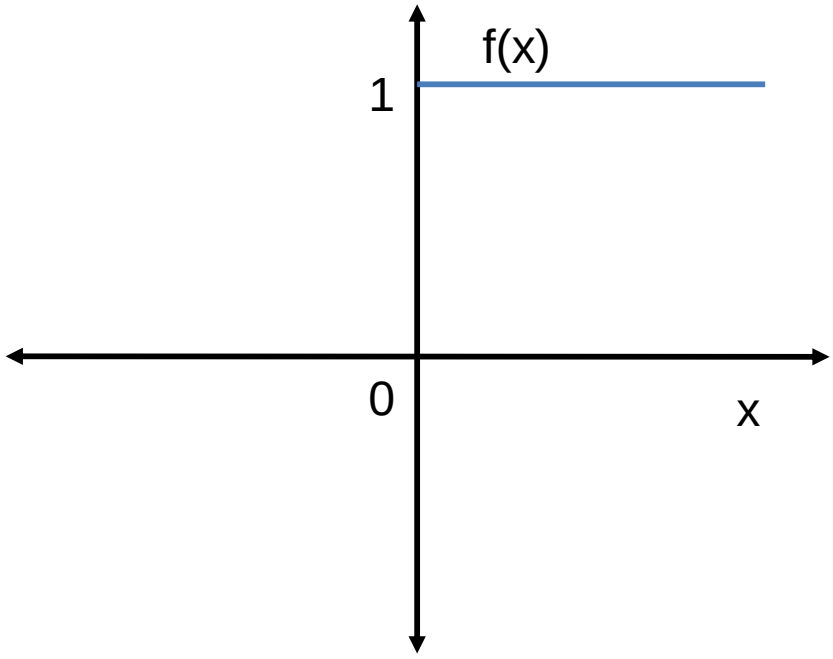
Activation Functions

- **Binary Step function**

  It is used in single-layer networks to convert input to binary output.

$$f(x) = \begin{cases} 1 \ \ if \ \ x \geq \theta \\ 0 \ \ if \ \ x < \theta \end{cases}$$

where θ is threshold value



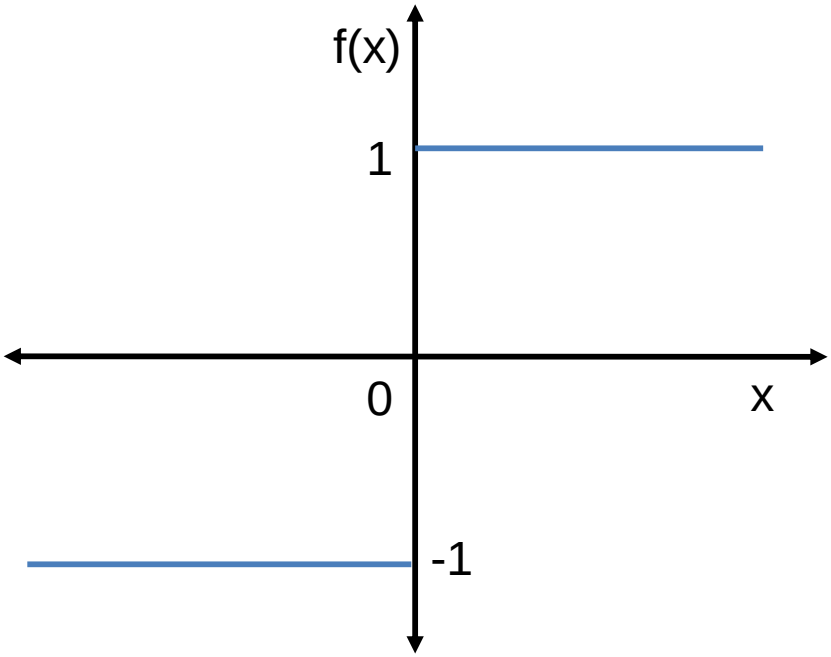
Activation Functions

- **Bipolar Step function**

  It is used in single-layer networks to convert input to bipolar output.

$$f(x) = \begin{cases} 1 \ \ if \ x \geq \theta \\ -1 \ \ if \ \ x < \theta \end{cases}$$

where θ is threshold value



Activation Functions

- **Sigmoidal function :** It is Widely used in back propagation Nets.

  It is of two types

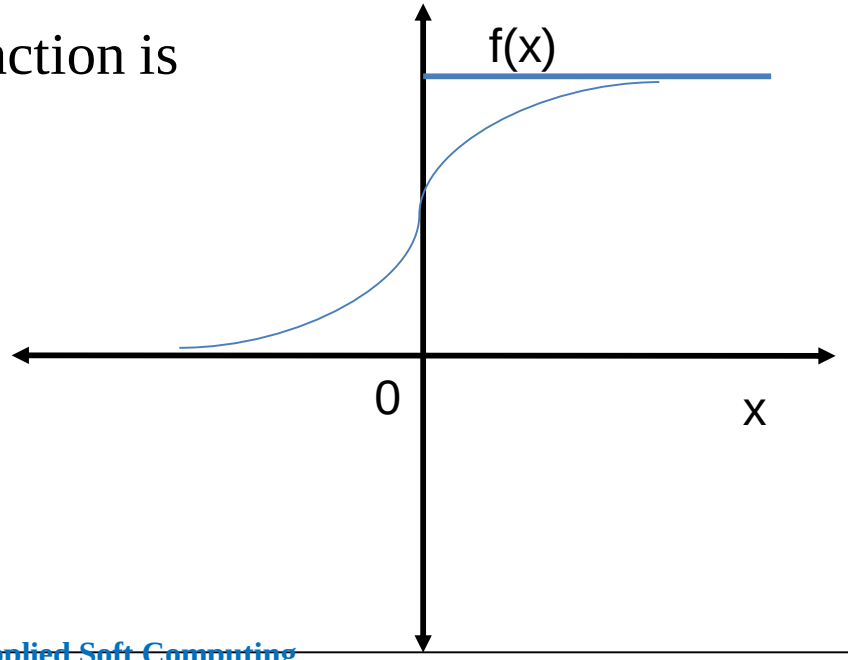
  - **Binary Sigmoid function: It is also known as logistic or unipolar sigmoid function.**

    The range of this function is 0 to 1.

    $f(x) = \frac{1}{1+ e^{-\lambda x}}$ λ is steepness parameter,

    the derivative of this function is

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



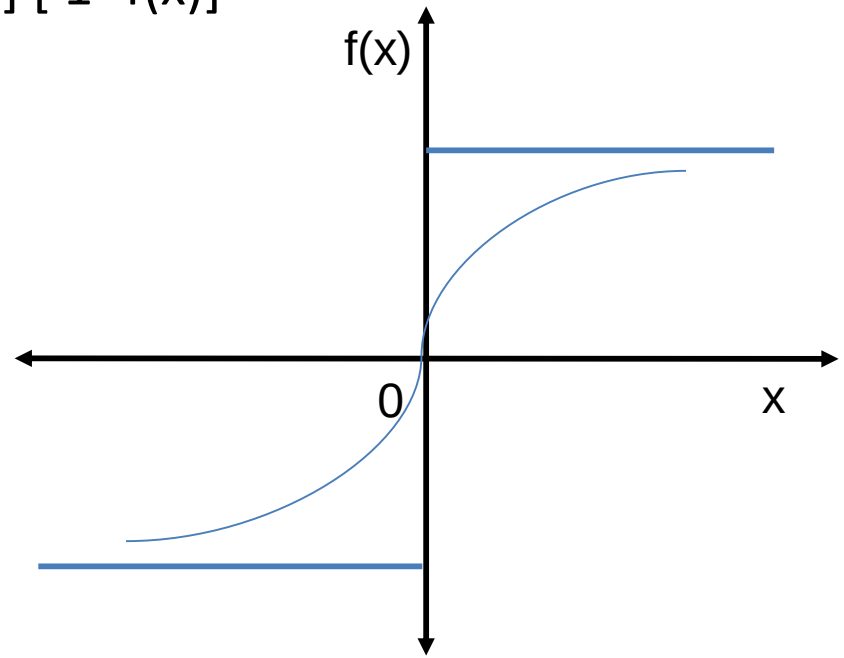
Activation Functions

- **Sigmoidal function**
  - **Bipolar Sigmoid function**

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

λ is steepness parameter, the derivative of this function is

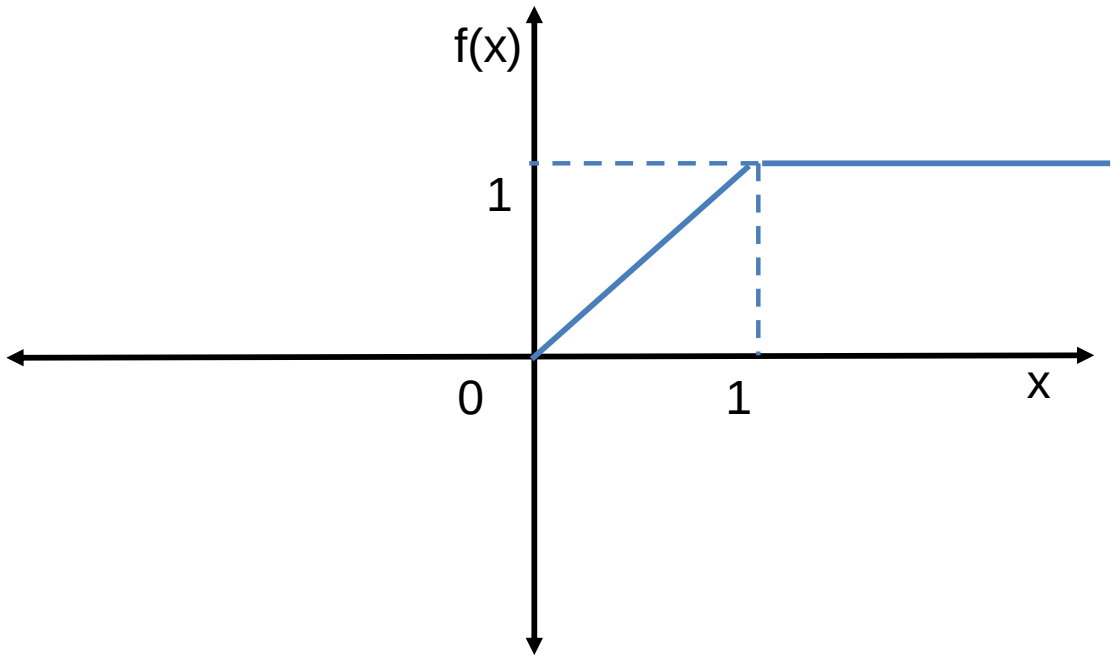
$$f'(x) = \frac{\lambda}{2} [\, 1 + f(x)] [\, 1- f(x)]$$



Activation Functions

- **Ramp Function**

$$f(x) = \begin{cases} 1 \ \ if \ x > 1 \\ x \ \ \ if \ \ 0 \leq x \leq 1 \\ 0 \ \ if \ \ x < 0 \end{cases}$$



Basic Concepts of training

- Neurons are arranged in single layer with multiple inputs and connected weights
- Single neuron termed as perceptron is trained
- Same training algorithm is applied to other neurons in the layer
- Learning is achieved by adjusting the weights associated with the inputs

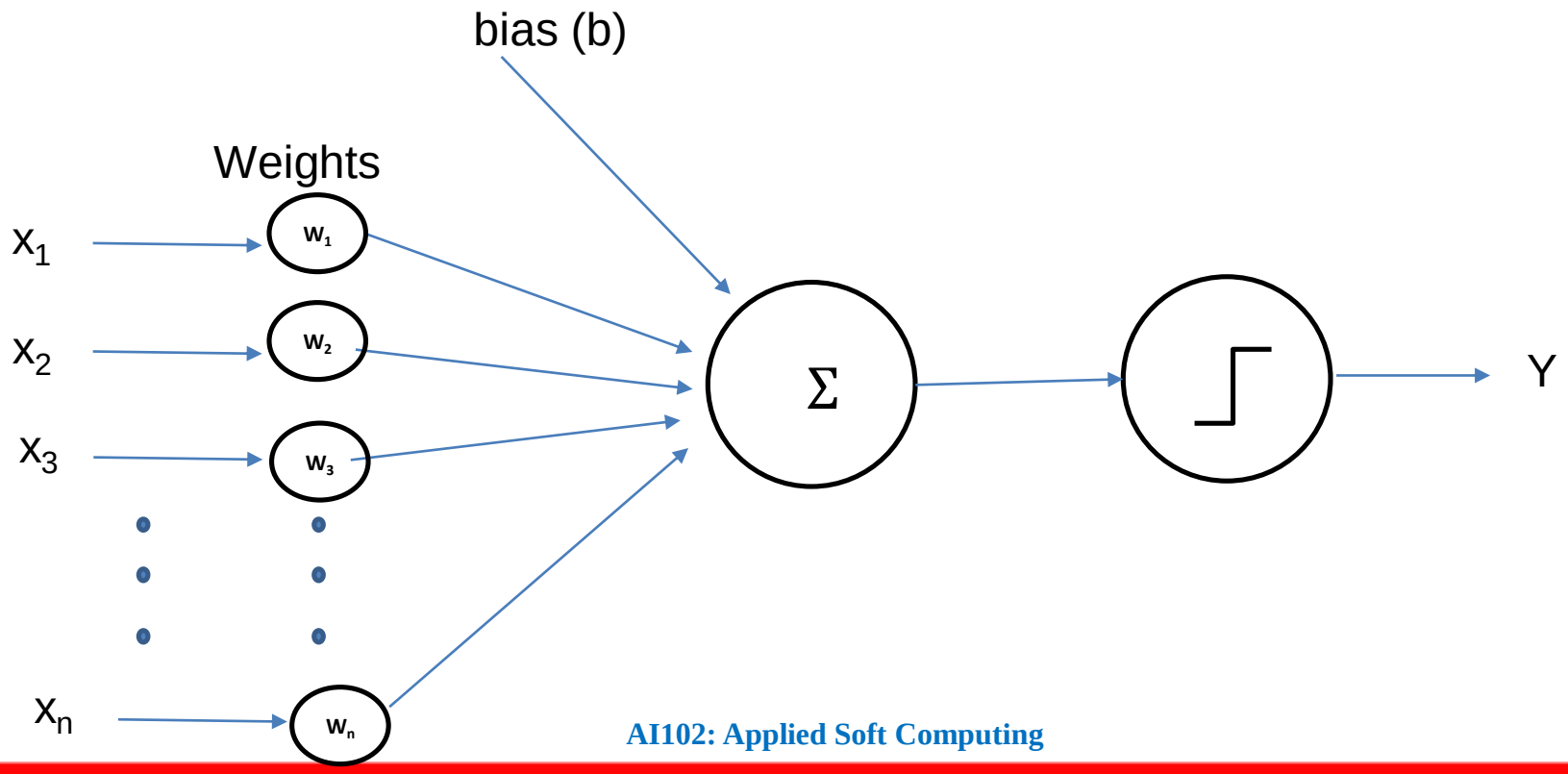
Architecture of Single Layer Feed Forward ANN

Consider a perceptron with n+1 inputs consisting of $x_0, x_1, x_2 \ldots x_n$

X and Y are input and output vectors

W is the weight matrix consisting of $w_0, w_1, w_2 \ldots w_n$

$x_0 * w_0$ is bias represented by b



Training Single Layer Feed Forward ANN with single output

CHITKARA UNIVERSITY

Algorithm :

Step 0: Initialize $w_1, w_2 \ldots w_n$ and bias b to some random values

Also initialize α ($0 < \alpha \leq 1$), which is learning rate.

For simplicity $\boldsymbol{\alpha}$ is set to 1 , Y is taken as single output **y**

Step 1: Perform steps 2-6 until the final stopping condition applies

Step 2: Perform steps 3-5 for each training pair

Step 3: Apply the inputs to the input units (neurons) in the layer

Step 4: Find output of the network

z (aggregated input) = b + $\sum_{i=1}^{n} w_{ik*} x_i$

Apply activation function (binary step) and calculate output

y (observed output)= f(z) = $\begin{cases} 1 & if \ z \geq 0 \\ 0 & if \ z < 0 \end{cases}$

Training Single Layer Feed Forward ANN with single output

Step 5: Adjust weights and bias

Compare observed output *(y)* with target output *(t)*

if $y \neq t$ then

$$w_{i(new)} = w_{i(old)} + \alpha(t)x_i$$

$$b_{(new)} = b_{(old)} + \alpha(t)$$

else

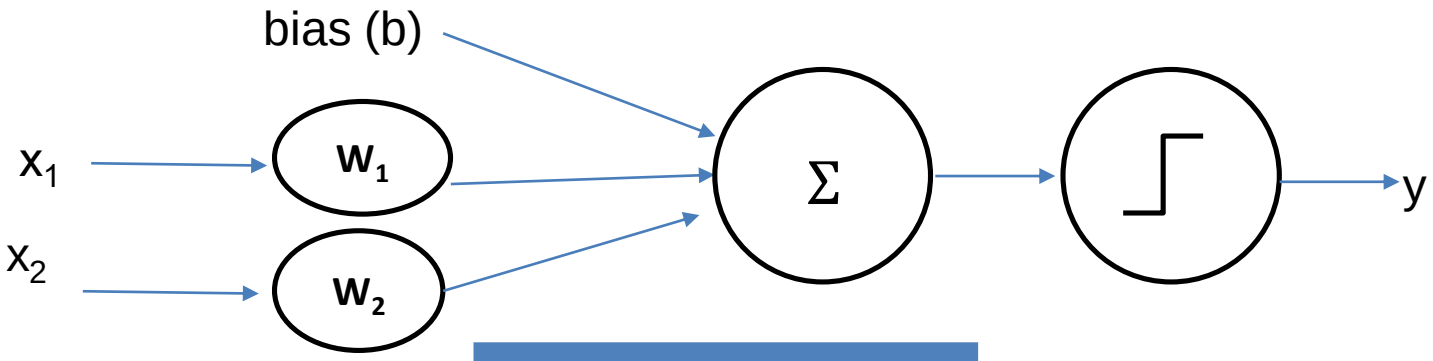
$$w_{i(new)} = w_{i(old)}$$

$$b_{(new)} = b_{(old)}$$

Step 6: Train the network until there is no weight change. This is stopping condition. If condition is not met then go to step 2.

Example: Training ANN for AND Function

- Implement AND function using perceptron networks for bipolar inputs and targets



OR Gate Truth Table		
x_1	x_2	Target t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Example : Training ANN for AND Function

CHITKARA UNIVERSITY

Terms used:

Input $\boldsymbol{x_i}$, weights $\boldsymbol{w_i}$ *(where i=1,2)*, bias ***b***, observed output ***y***, target output ***t***, aggregated input ***z,*** learning rate α

Assumptions :

1. Learning rate α is 1
2. Inputs x_1 and x_2 will have values as per the truth table of OR gate
3. Output y is single output
4. Activation function is bipolar function

$$f(z) = \begin{cases} 1 \ \ if \ z > 0 \\ 0 \ \ if \ \ z = \ 0 \\ -1 \ \ if \ \ z < \ 0 \end{cases}$$

Example : Training ANN for AND Function

Initialize w_1, w_2 and b to 0, α is set to 1

Input		Target Output	Aggregated Output	Observed Output	Weight Changes			Weights		
x_1	x_2	t	z	y	Δw_1	Δw_2	Δb	w_1 (0)	w_2 (0)	b (0)
1	1	1	0	0	1	1	1	1	1	1
1	-1	-1	1	1	-1	1	-1	0	2	0
-1	1	-1	2	1	1	-1	-1	1	1	-1
-1	-1	-1	-3	-1	0	0	0	1	1	-1

Weight Changes are not zero for all the training pairs
Run another Iteration (EPOCH)

Example : Training ANN for AND Function

CHITKARA UNIVERSITY

EPOCH 2

Input		Target Output	Aggregated Output	Observed Output	Weight Changes			Weights		
x_1	x_2	t	z	y	Δw_1	Δw_2	Δb	w_1 (1)	w_2 (1)	b (-1)
1	1	1	1	1	0	0	0	1	1	-1
1	-1	-1	-1	-1	0	0	0	1	1	-1
-1	1	-1	-1	-1	0	0	0	1	1	-1
-1	-1	-1	-3	-1	0	0	0	1	1	-1

Weight Changes are zero for all the training pairs
This will be the last EPOCH

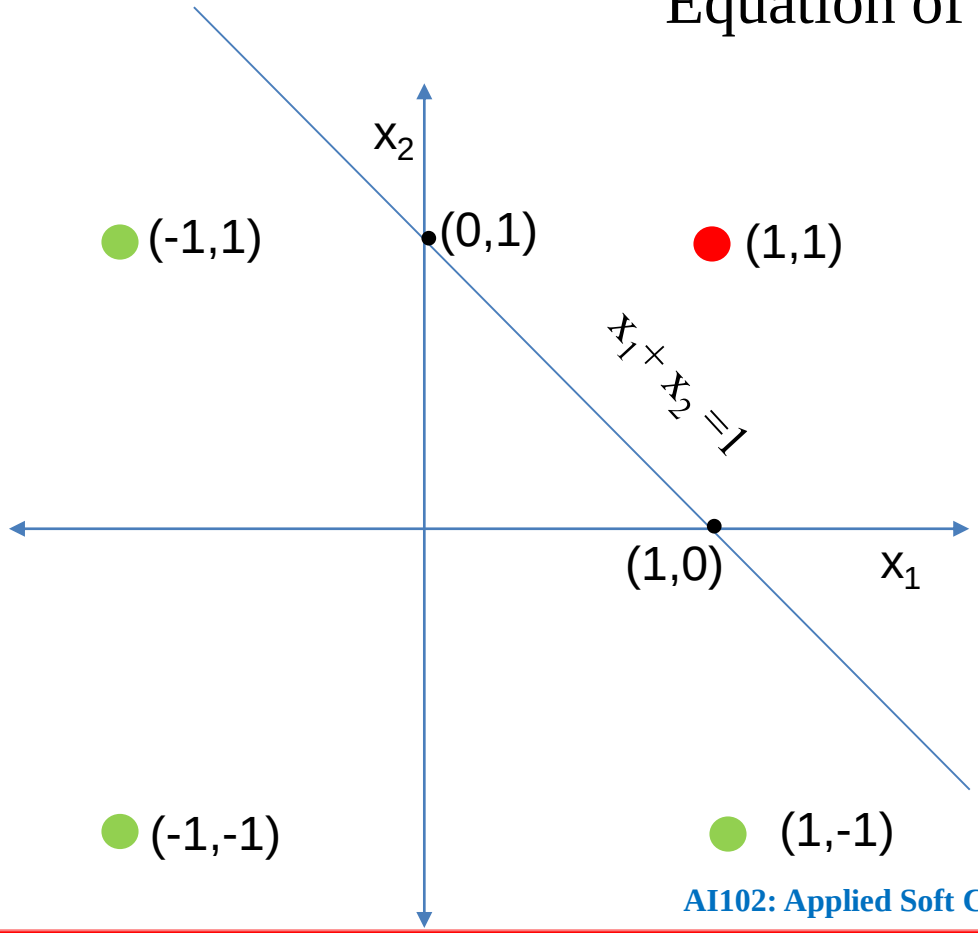
w_1 =1, w_2 =1, b =-1

Example : Training ANN for AND Function

Considering equation: $y = b + w_1 * x_1 + w_2 * x_2$

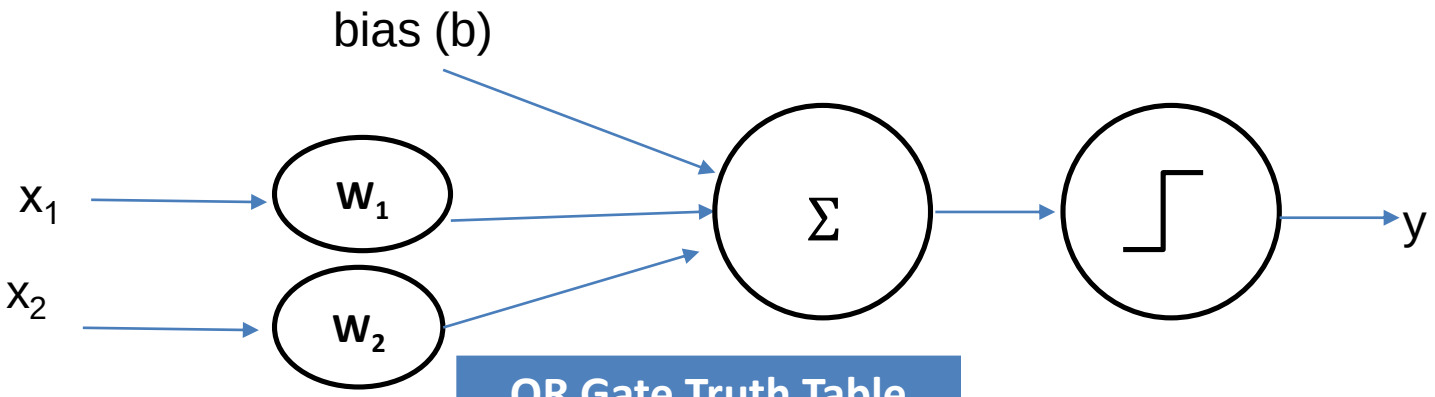
Substituting values of $\boldsymbol{w_1 = 1, w_2 = 1, b = -1}$

Equation of line becomes $x_1 + x_2 = 1$



Question: Training ANN for OR Function

- Implement OR function with binary inputs and bipolar targets using perceptron training algorithm.



OR Gate Truth Table		
x_1	x_2	Target t
1	1	1
1	0	1
0	1	1
0	0	-1

Question : Training ANN for OR Function

Terms used:

Input $\boldsymbol{x_i}$, weights $\boldsymbol{w_i}$ *(where i=1,2)*, bias ***b***, observed output ***y***, target output ***t***, aggregated input ***z,*** learning rate α

Assumptions :

1. Learning rate α is 1
2. Inputs x_1 and x_2 will have values as per the truth table of OR gate
3. Output y is single output
4. Activation function is binary step function

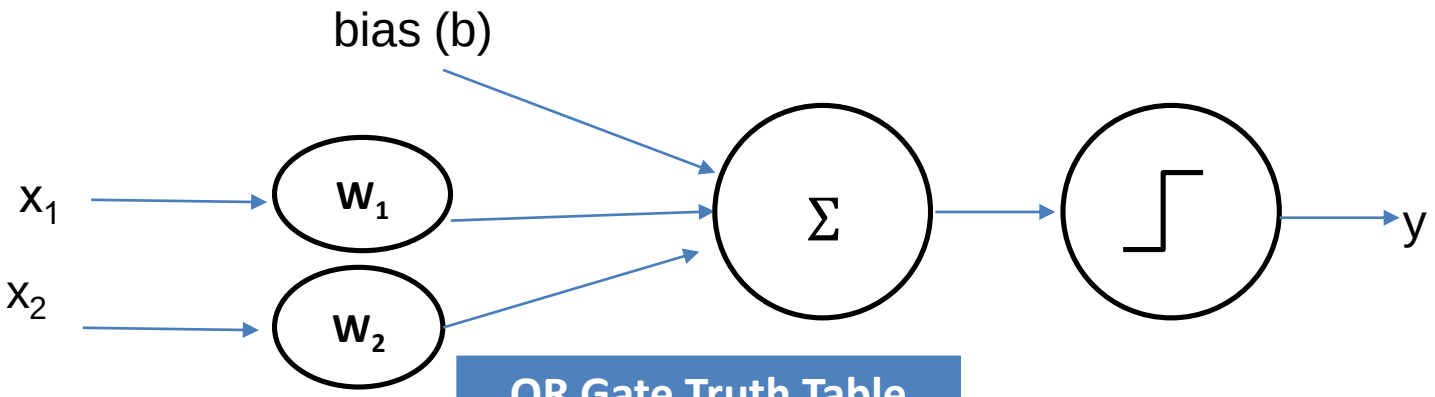
$$f(z) = \begin{cases} 1 \ \ if \ z > 0 \\ 0 \ \ if \ \ z \leq 0 \end{cases}$$

<u>In the same question change the activation function as given below, taking rest of the things same and the find the solution.</u>

$$f(z) = \begin{cases} 1 \ \ if \ z > 0 \\ 0 \ \ if \ \ z = 0 \\ -1 \ \ if \ \ z < 0 \end{cases}$$

Example: Training ANN for OR Function

- Implement OR function with binary inputs and bipolar targets using perceptron training algorithm.



OR Gate Truth Table		
x_1	x_2	Target t
1	1	1
1	0	1
0	1	1
0	0	-1

Example : Training ANN for OR Function

Terms used:

Input $\boldsymbol{x_i}$, weights $\boldsymbol{w_i}$ *(where i=1,2)*, bias ***b***, observed output ***y***, target output ***t***, aggregated input ***z,*** learning rate α

Assumptions :

1. Learning rate α is 1
2. Inputs x_1 and x_2 will have values as per the truth table of OR gate
3. Output y is single output
4. Activation function is binary step function

$$f(z) = \begin{cases} 1 \ \ if \ z > 0 \\ 0 \ \ if \ \ z \leq 0 \end{cases}$$

Example : Training ANN for OR Function

Initialize w_1, w_2 and b to 0, α is set to 1

Input		Target Output	Aggregated Output	Observed Output	Weight Changes			Weights		
x_1	x_2	t	z	y	Δw_1	Δw_2	Δb	w_1 (0)	w_2 (0)	b (0)
1	1	1	0	0	1	1	1	1	1	1
1	0	1	2	1	0	0	0	1	1	1
0	1	1	2	1	0	0	0	1	1	1
0	0	-1	1	1	0	0	-1	1	1	0

Weight Changes are not zero for all the training pairs
Run another Iteration (EPOCH)

Example : Training ANN for OR Function

EPOCH 2

Input		Target Output	Aggregated Output	Observed Output	Weight Changes			Weights		
x_1	x_2	t	z	y	Δw_1	Δw_2	Δb	w_1 (1)	w_2 (1)	b (0)
1	1	1	1	1	0	0	0	1	1	0
1	0	1	1	1	0	0	0	1	1	0
0	1	1	1	1	0	0	0	1	1	0
0	0	-1	0	0	0	0	-1	1	1	-1

Weight Changes are not zero for all the training pairs
Run another Iteration (EPOCH)

Example : Training ANN for OR Function

EPOCH 3

Input		Target Output	Aggregated Output	Observed Output	Weight Changes			Weights		
x_1	x_2	t	z	y	Δw_1	Δw_2	Δb	w_1 (1)	w_2 (1)	b (-1)
1	1	1	1	1	0	0	0	1	1	-1
1	0	1	0	0	1	0	1	2	1	0
0	1	1	1	1	0	0	0	2	1	0
0	0	-1	0	0	0	0	-1	2	1	-1

Weight Changes are not zero for all the training pairs
Run another Iteration (EPOCH)

Example : Training ANN for OR Function

EPOCH 4

Input		Target Output	Aggregated Output	Observed Output	Weight Changes			Weights		
x_1	x_2	t	z	y	Δw_1	Δw_2	Δb	w_1 (2)	w_2 (1)	b (-1)
1	1	1	2	1	0	0	0	2	1	-1
1	0	1	1	1	0	0	0	2	1	-1
0	1	1	0	0	0	0	0	2	2	0
0	0	-1	0	0	0	0	-1	2	2	-1

Weight Changes are not zero for all the training pairs
Run another Iteration (EPOCH)

Example : Training ANN for OR Gate

CHITKARA UNIVERSITY

EPOCH 5

Input		Target Output	Aggregated Output	Observed Output	Weight Changes			Weights		
x_1	x_2	t	z	y	Δw_1	Δw_2	Δb	w_1 (2)	w_2 (2)	b (-1)
1	1	1	3	1	0	0	0	2	2	-1
1	0	1	1	1	0	0	0	2	2	-1
0	1	1	1	1	0	0	0	2	2	-1
0	0	-1	-1	0	0	0	-1	2	2	-2

Weight Changes after 5th EPOCH is not zero for all the training pairs
For the convergence we may consider the solution after 5th EPOCH

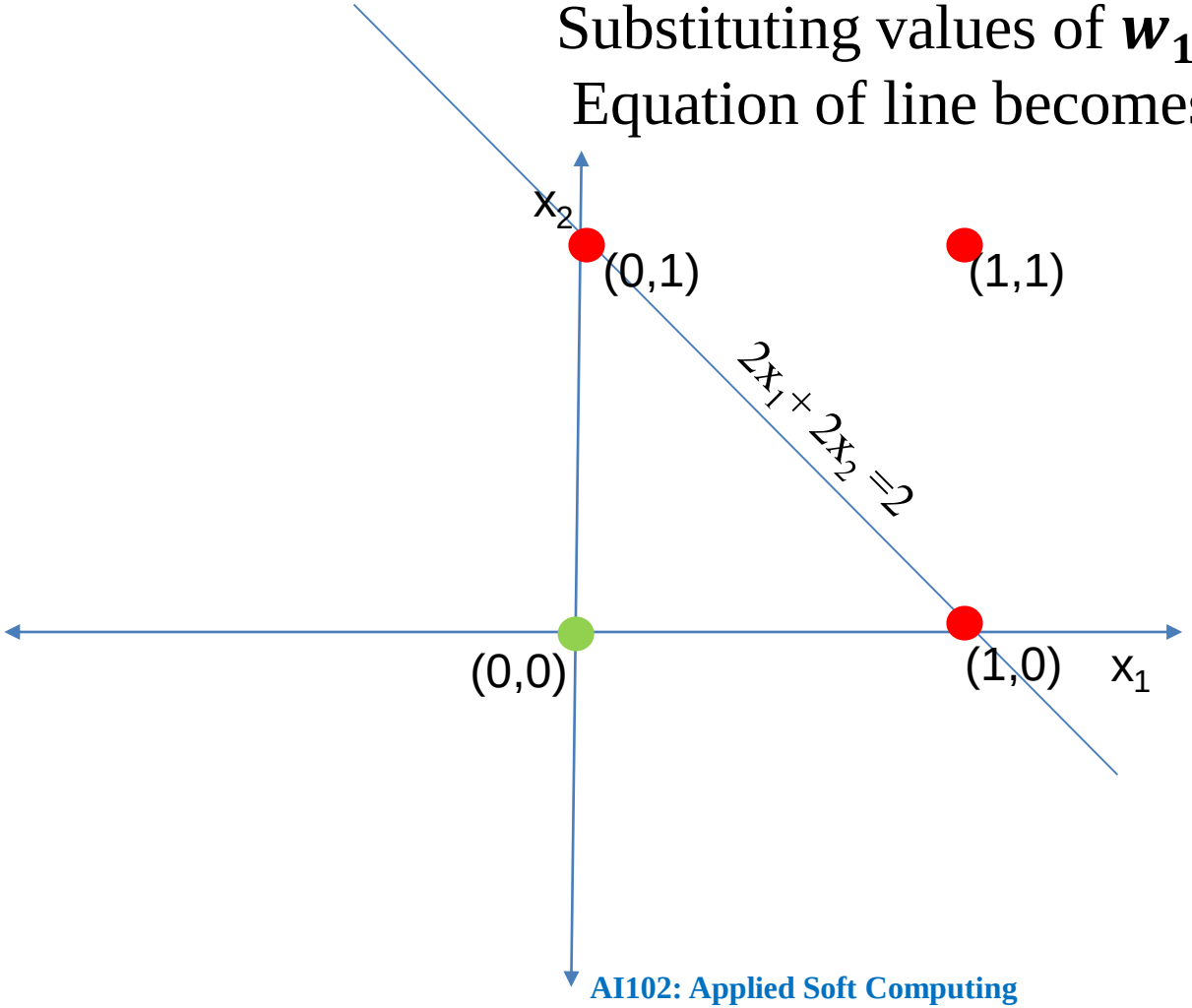
w_1 =2, w_2 =2, b =-2

Example : Training ANN for AND Function

Considering equation: $y = b + w_1 * x_1 + w_2 * x_2$

Substituting values of $\boldsymbol{w_1 = 2, w_2 = 2, b = -2}$

Equation of line becomes $x_1 + x_2 = 1$



Question: Training ANN for OR Function

Terms used:

Input $\boldsymbol{x_i}$, weights $\boldsymbol{w_i}$ *(where i=1,2)*, bias ***b***, observed output ***y***, target output ***t***, aggregated input ***z,*** learning rate α

Assumptions :

1. Learning rate α is 1
2. Inputs x_1 and x_2 will have values as per the truth table of OR gate
3. Output y is single output
4. Activation function is bipolar function

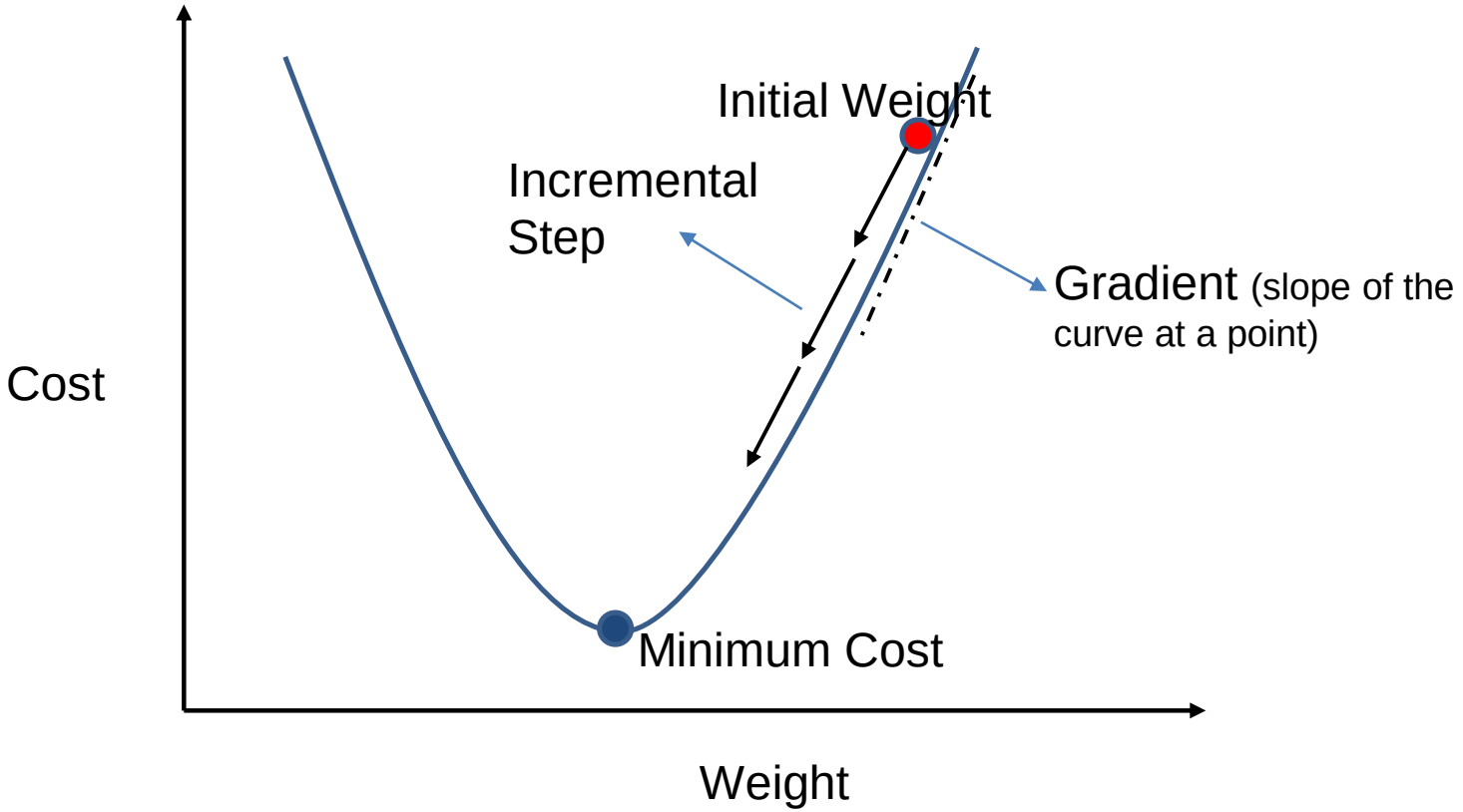
$$f(z) = \begin{cases} 1 \ \ if \ z > 0 \\ 0 \ \ if \ \ z = \ 0 \\ -1 \ \ if \ \ z < \ 0 \end{cases}$$

Basics of Backpropagation

- Applied to multi-layer feed-forward network with continuous differentiable activation functions.
- The underlying concept of weight update is gradient-descent method.
- The networks using backpropagation algorithm is known as back propagation network (BPN)

Basics of Gradient-Descent Learning

- Type of Supervised Learning
- Optimization algorithm to train neural network
- Based on minimizing error in terms of weights and activation function
- The activation function should be continuous differentiable



Gradient-Descent Learning

$$w_i = w_i - \alpha \frac{\delta(e)}{\delta(w_i)}$$

w_i - weight

α – learning rate [0,1]

$\frac{\delta(e)}{\delta(w_i)}$ - partial derivative for rate of change of error w.r.t weight on cost function

Error gradient can be found either by least mean square or by back propagation

Gradient-Descent (GD) Learning

- Learning rate α scales the gradient and thus controls the step size. It has strong influence on performance.
- Smaller the learning rate longer the GD converges, or it may reach maximum iteration before reaching the optimum point.
- If learning rate is too large the algorithm may not converge to the optimal point.
- Following are the steps for Gradient Descent algorithm:

  Step 1: calculate gradient at a particular point

  Step 2: make a scaled step in the opposite direction to the gradient

  Step 3: repeat steps 2 and 3 until one of the below criteria is met:

  » iterations reach to maximum number

  » step size becomes smaller than the tolerance

Training of Back Propagation Network (BPN)

Stages of training BPN

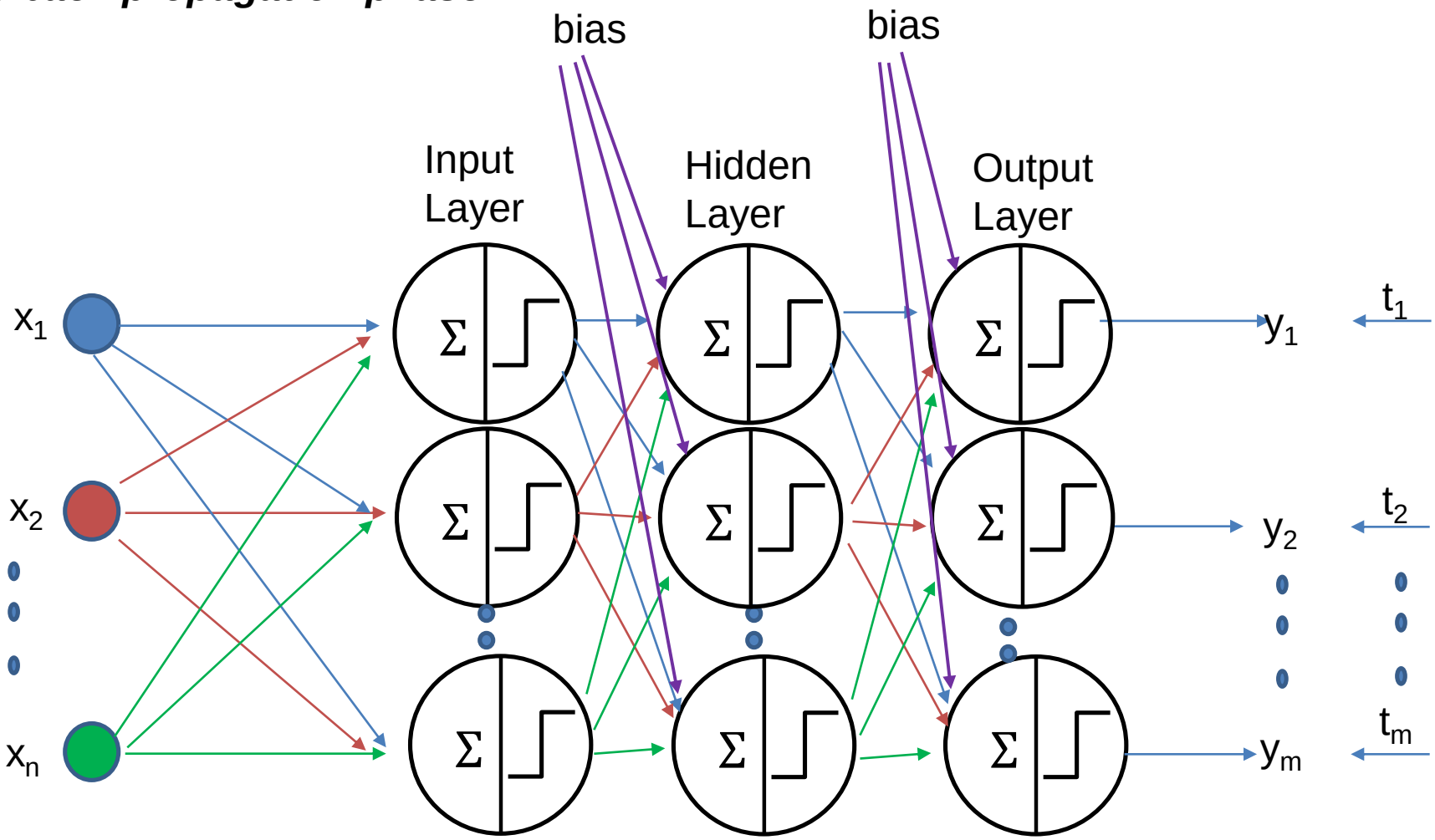
Stage 1: feed forward of input training pattern

Stage 2: calculation of error and back propagation of error

Stage 3: updation of weights

Architecture of BPN

For simplicity, figure is presenting only feed forward phase and not the back propagation phase



Architecture of BPN

Features of Back Propagation Neural Network

- Multilayer architecture consisting of Input Layer, Output Layer and Hidden Layer(s)
- Neurons present in hidden layer and output layer have biases
- In feedforward phase signals are sent in forward direction and in back propagation phase signals are sent in reverse direction
- Input or output can be either binary (0,1) or bipolar (-1,1)
- Activation function should be such that it is increasing monotonically and is differentiable

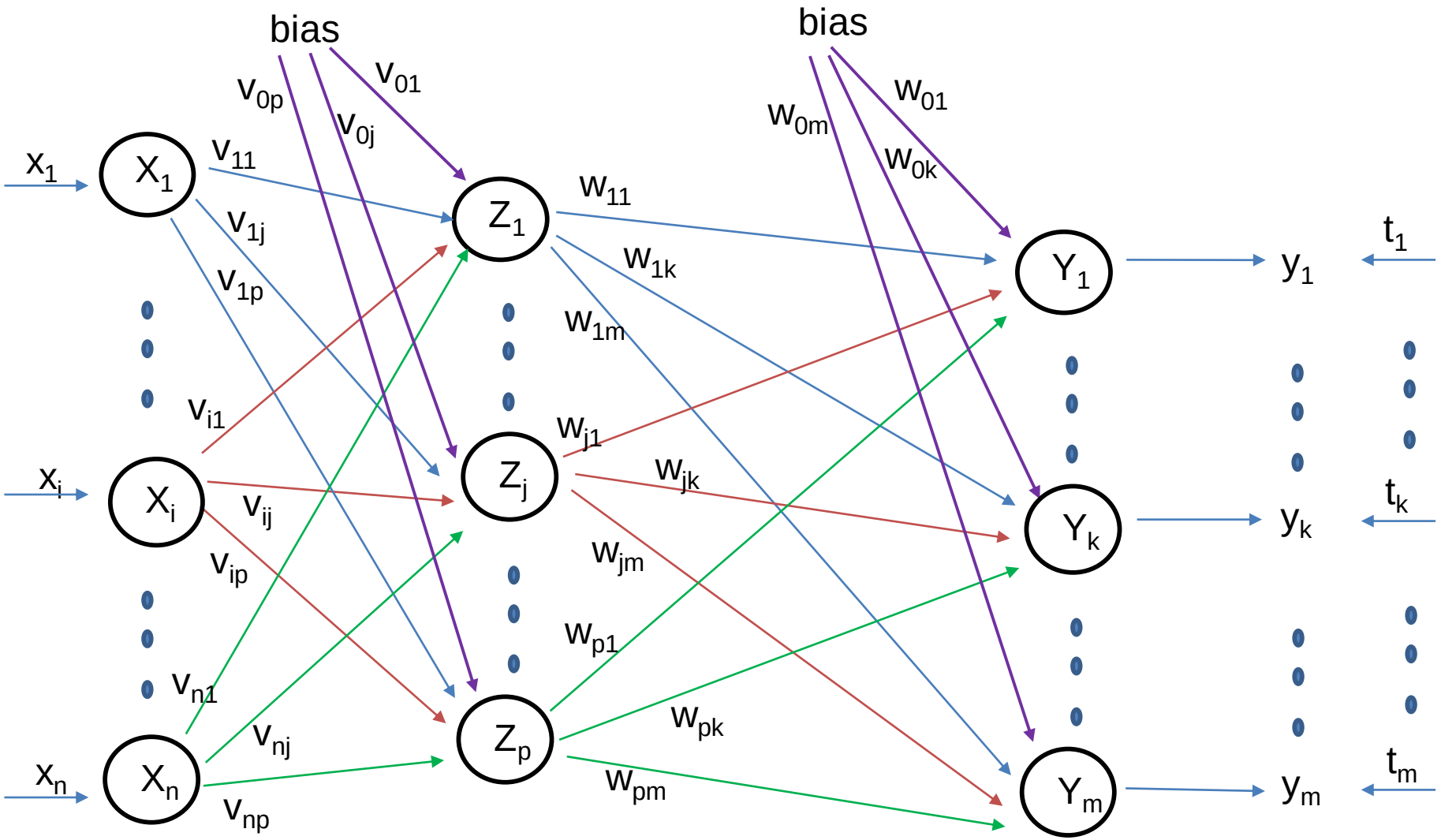
Architecture of BPN

Features of Back Propagation Neural Network

- Multilayer architecture consisting of Input Layer, Output Layer and Hidden Layer(s)
- Neurons present in hidden layer and output layer have biases
- In feedforward phase signals are sent in forward direction and in back propagation phase signals are sent in reverse direction
- Input or output can be either binary (0,1) or bipolar (-1,1)
- Activation function should be such that it is increasing monotonically and is differentiable

Architecture of BPN

For simplicity feed forward information flow is shown in figure



Terms used in Algorithm for training BPN

CHITKARA UNIVERSITY

x - input training vector (x_1 ,… x_i , … x_n)

t - target output vector (t_1 ,… t_k , … t_m)

x_i – i_{th} input unit ; z_j – j_{th} hidden unit ; y_k – k_{th} output unit

v_{0j} – bias on the j_{th} hidden unit

w_{0k} – bias on the k_{th} output unit

v_{ij} – weight from i_{th} input unit to j_{th} hidden unit

w_{jk} - weight from j_{th} hidden unit to k_{th} output unit

z_{inj} - aggregated input fed to z_j

y_{ink} - aggregated input fed to y_k

δ_k – error correction weight adjustment for w_{jk} due to error at output unit y_k , which is back propagated to hidden units that feed into y_k

δ_j – error correction weight adjustment for v_{ij} due to error at hidden unit z_j , which is back propagated to input units that feed into z_j

Training BPN

CHITKARA UNIVERSITY

Algorithm :

Step 0: Initialize weights, bias to some random values

Also initialize α ($0 < \alpha \leq 1$), which is learning rate.

For simplicity $\boldsymbol{\alpha}$ is set to 1

Step 1: Perform steps 2-9 until the final stopping condition applies

Step 2: Perform steps 3-8 for each training pair

Feed Forward Phase

Step 3: Each input unit receives input signal x_i (i=1 to n) and sends it to hidden unit

Step 4: Each hidden unit z_j (j=1 to p) sums the weighted input signals and calculates the aggregated input

$$z_{inj} \text{ (aggregated input)} = v_{0j} + \sum_{i=1}^{n} v_{ij} * x_i$$

Apply activation function (sigmoidal function) and calculate output

$$z_j \text{ (observed output)} = f(z_{inj})$$

Training BPN

Step 4 continues: Send output z_j of the hidden unit to the input of output layer unit

Step 5: For each output unit y_k (k=1 to m), calculate aggregated input

$$y_{ink} \text{ (aggregated input)} = w_{0k} + \sum_{j=1}^{p} w_{jk*} x_i$$

Apply activation function (sigmoidal function) and calculate output

$$y_k \text{ (observed output)} = f(y_{ink})$$

Back Propagation Phase

Step 6: Each output unit y_k (k=1 to m) receives target pattern corresponding to the input training pattern and computes the error correction term:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

Training BPN

Step 6 continues: On the basis of calculated error correction, find the change in weights and bias

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

$$\Delta w_{0k} = \alpha\, \delta_k$$

Send δ_k backwards to the hidden layer.

Step 7: Each hidden unit z_j (j=1 to p) aggregates its delta inputs received from output units

$$\delta_{inj} = \sum_{k=1}^{m} w_{jk*}\, \delta_k$$

The term δ_{inj} is multiplied with derivative of $f(z_{inj})$ to calculate error term $\delta_j = \delta_{inj}\, f'(z_{inj})$

On the basis of calculated δ_j, find the change in weights and bias

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

$$\Delta v_{0j} = \alpha\, \delta_j$$

Training BPN

CHITKARA UNIVERSITY

Weight and Bias Updation

Step 8 : Each output unit y_k (k=1 to m), updates bias and weights:

$$w_{jk(new)} = w_{jk(old)} + \Delta w_{jk}$$

$$w_{0k(new)} = w_{0k(old)} + \Delta w_{0k}$$

Each hidden unit z_j (j=1 to p), updates its bias and weights

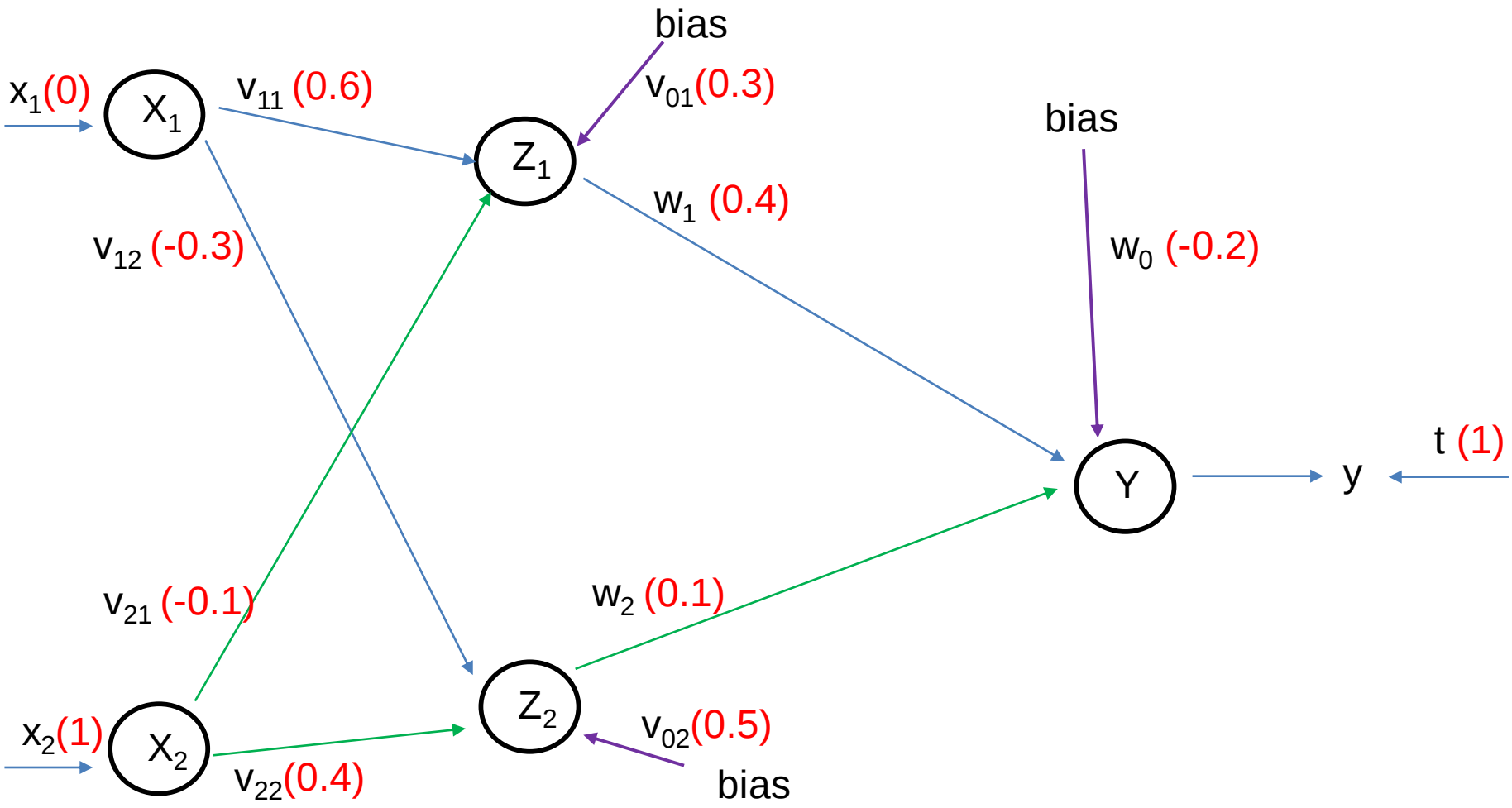
$$v_{ij(new)} = v_{ij(old)} + \Delta vij$$

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

Step 9: Check for stopping condition. It may be specified number of epochs reached or when the observed output equals target output

Example of training BPN

Using BPN, find new weights for the net shown in Figure below. It is presented with input pattern [0,1] and the target output is 1. Use a learning rate α =0.25 and binary sigmoidal activation function.



Example of Training BPN

Initial weights are $[v_{11}, v_{21}, v_{01}] = [0.6, -0.1, 0.3]$

$[v_{12}, v_{22}, v_{02}] = [-0.3, 0.4, 0.5]$

$[w_1, w_2, w_0] = [0.4, 0.1, -0.2]$

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

Feed Forward Phase:

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

z_{in2} (aggregated input) $= v_{02} + x_1 v_{12} + x_2 v_{22} = 0.5 + 0 \times (-0.1) + 1 \times (0.4) = 0.9$

Apply activation function

$$z_1 = f(z_{in1}) = \frac{1}{1+e^{-z_{in1}}} = \frac{1}{1+e^{-(0.2)}} = 0.5498$$

$$z_2 = f(z_{in2}) = \frac{1}{1+e^{-z_{in2}}} = \frac{1}{1+e^{-(0.9)}} = 0.7109$$

$y_{in} = w_0 + z_1 w_1 + z_2 w_2 = (-0.2) + 0.5498 \times 0.4 + 0.7109 \times 0.1 = 0.9101$

Example of Training BPN

Apply activation function

$$y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-(0.09101)}} = 0.5227$$

***Back Propagation Phase*:**

Calculate error ($\delta_k = (t_k - y_k) f'(y_{ink})$)

As there is only one output so k is 1

$$\delta_1 = (t - y) f'(y_{in})$$

$$f'(y_{in}) = f(y_{in})\,[1 - f(y_{in})] = 0.5227\,(1-0.5227) = 0.2495$$

$$\delta_1 = (1 - 0.5227) \times 0.2495 = 0.1191$$

Find the change in weights between hidden layer and output layer

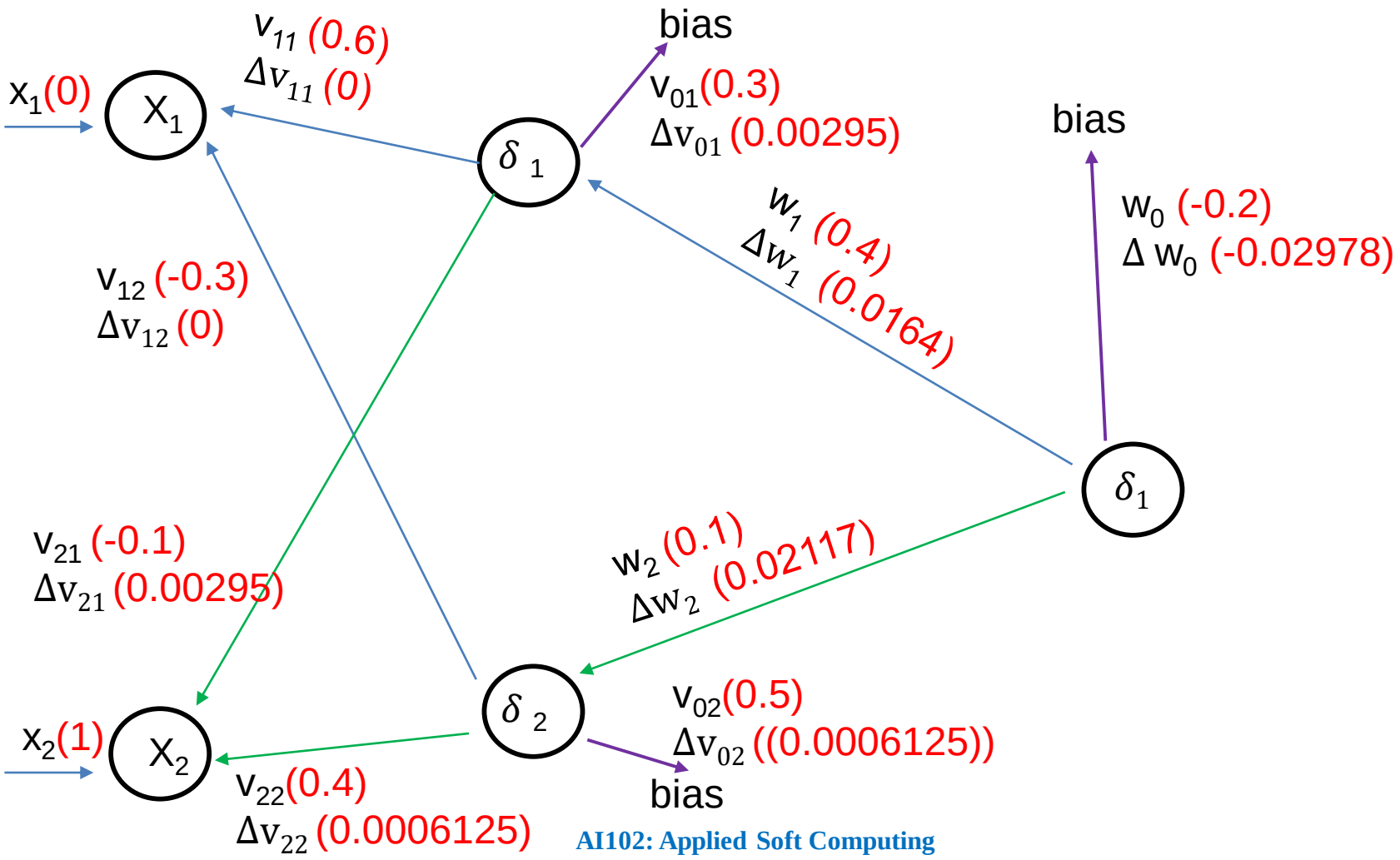
$$\Delta w_1 = \alpha\, \delta_1\, z_1 = 0.25 \times 0.1191 \times 0.5498 = 0.0164$$

$$\Delta w_2 = \alpha\, \delta_1\, z_2 = 0.25 \times 0.1191 \times 0.7109 = 0.02117$$

$$\Delta w_0 = \alpha\, \delta_1 = 0.25 \times 0.1191 = 0.02978$$

Example of training BPN

Back Propagation Phase (Representing old Weights and change in weights)
Change weights accordingly for next iteration



Example of Training BPN

Calculate error between input and hidden layer ($\delta_j = \delta_{inj}\, f'(z_{inj})$)

$\delta_{inj} = \sum_{k=1}^{m} w_{jk*}\, \delta_k$ (j=1 to 2), (k is 1)

$\delta_{in1} = w_{1\,x}\, \delta_1 = 0.4 \times 0.1191 = 0.04764$

$\delta_{in2} = w_{2\,x}\, \delta_1 = 0.1 \times 0.1191 = 0.01191$

Error $\delta_1 = \delta_{in1}\, f'(z_{in1})$ where $f'(z_{in1}) = f(z_{in1})\,[1- f(z_{in1})]$

$f'(z_{in1}) = 0.5498(1-0.5498) = 0.2475$

$\delta_1 = 0.04764 \times 0.2475 = 0.0118$

Similarly $\delta_2 = \delta_{in2}\, f'(z_{in2})$ where $f'(z_{in2}) = f(z_{in2})\,[1- f(z_{in2})]$

$f'(z_{in2}) = 0.7109(1-0.7109) = 0.2055$

$\delta_2 = 0.01191 \times 0.2055 = 0.00245$

Example of Training BPN

Find the change in weights between input layer and hidden layer

$\Delta v_{11} = \alpha\ \delta_1\ x_1 = 0.25 \text{ x } 0.0118 \text{ x } 0 = 0$

$\Delta v_{21} = \alpha\ \delta_1\ x_2 = 0.25 \text{ x } 0.0118 \text{ x } 1 = 0.00295$

$\Delta v_{01} = \alpha\ \delta_1 = 0.25 \text{ x } 0.0118 = 0.00295$

$\Delta v_{12} = \alpha\ \delta_2\ x_1 = 0.25 \text{ x } 0.00245 \text{ x } 0 = 0$

$\Delta v_{22} = \alpha\ \delta_2\ x_2 = 0.25 \text{ x } 0.00245 \text{ x } 1 = 0.0006125$

$\Delta v_{02} = \alpha\ \delta_2 = 0.25 \text{ x } 0.0118 = 0.0006125$

Training BPN

CHITKARA UNIVERSITY

Weight and Bias Updation

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

$v_{12(new)} = v_{12(old)} + \Delta v_{12} = -0.3 + 0 = -0.3$

$v_{21(new)} = v_{21(old)} + \Delta v_{21} = -0.1 + 0.00295 = -0.09705$

$v_{22(new)} = v_{22(old)} + \Delta v_{22} = 0.4 + 0.0006125 = 0.4006125$

$w_{1(new)} = w_{1(old)} + \Delta w_1 = 0.4 + 0.0164 = 0.4164$

$w_{2(new)} = w_{2(old)} + \Delta w_2 = 0.1 + 0.02117 = 0.12117$

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

$v_{02\textbf{(new)}} = v_{02\textbf{(old)}} + \Delta v_{02} = 0.5 + 0.0006125 = 0.5006125$

$w_{0(new)} = w_{0(old)} + \Delta w_0 = -0.2 + 0.02978 = -0.17022$

Training BPN

CHITKARA UNIVERSITY

Weight and Bias Updation

Initial weights are $[v_{11}, v_{21}, v_{01}] = [0.6, -0.1, 0.3]$

$[v_{12}, v_{22}, v_{02}] = [-0.3, 0.4, 0.5]$

$[w_1, w_2, w_0] = [0.4, 0.1, -0.2]$

Final weights are $[v_{11}, v_{21}, v_{01}] = [0.6, -0.09705, 0.30295]$

$[v_{12}, v_{22}, v_{02}] = [-0.3, 0.4006125, 0.5006125]$

$[w_1, w_2, w_0] = [0.4164, 0.12117, -0.17022]$

Application of ANN

Case Study: A Fusion Approach of Multispectral Images with Synthetic Aperture Radar (SAR) Image for Flood Area Analysis

Objective: To assess the area destroyed by flood

- Multispectral Image technique – Monochrome images of same area
- SAR Image – 2D imaging of 3D objects

Application of ANN

CHITKARA UNIVERSITY

Image of flood area obtained by fusion of the images

Application of ANN

Image Classification using ANN Computing

- Image Fusion :
  - Fusion images help the feature extraction and recognition
  - Two categories of Image Fusion: Spatial Domain and **Spectral Domain**

IHS (Intensity Hue Saturation) Model for image fusion

RGB Color Space is transformed to IHS Model

Difference in gray value of pixel in image before and after flood is added to intensity

The flood area will be emphasized and non-flood area is depressed

IHS Model is inversely transformed to RGB

Application of ANN

ANN techniques was used for image recognition.

Multi-layer perceptron neural network based on back-propagation is used.

For this specific case, 3 layers back propagation neural network is taken

Sigmoid function was taken as an activation function

Error reduction was done using gradient descent technique

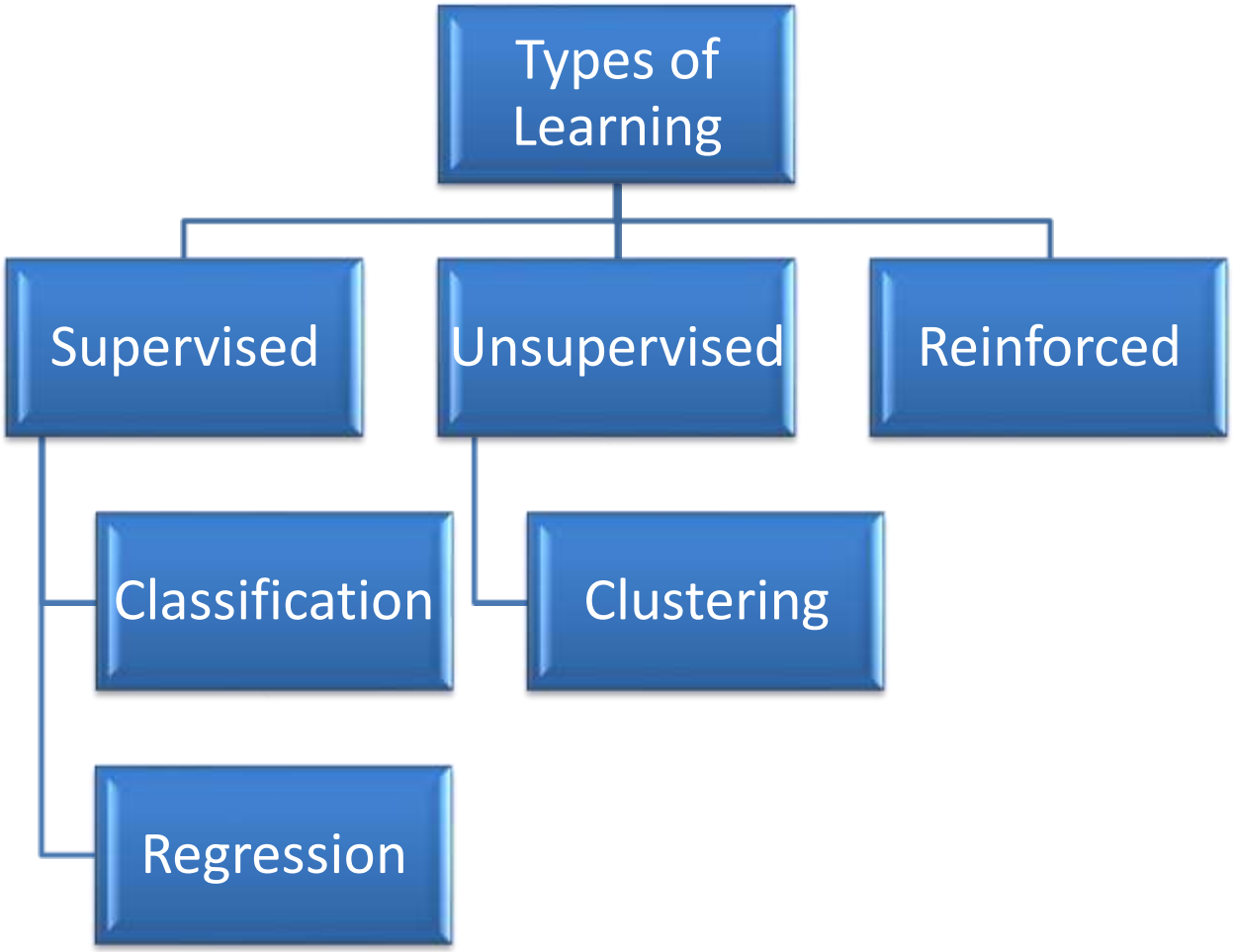
Methodology:

16 bits data obtained was reduced to 8 bits to obtain 256 values of intensity.

Flood assessment with fusion of data and without fusion was taken.

Accuracy with fusion data was more as compared to other.

Types of Learning

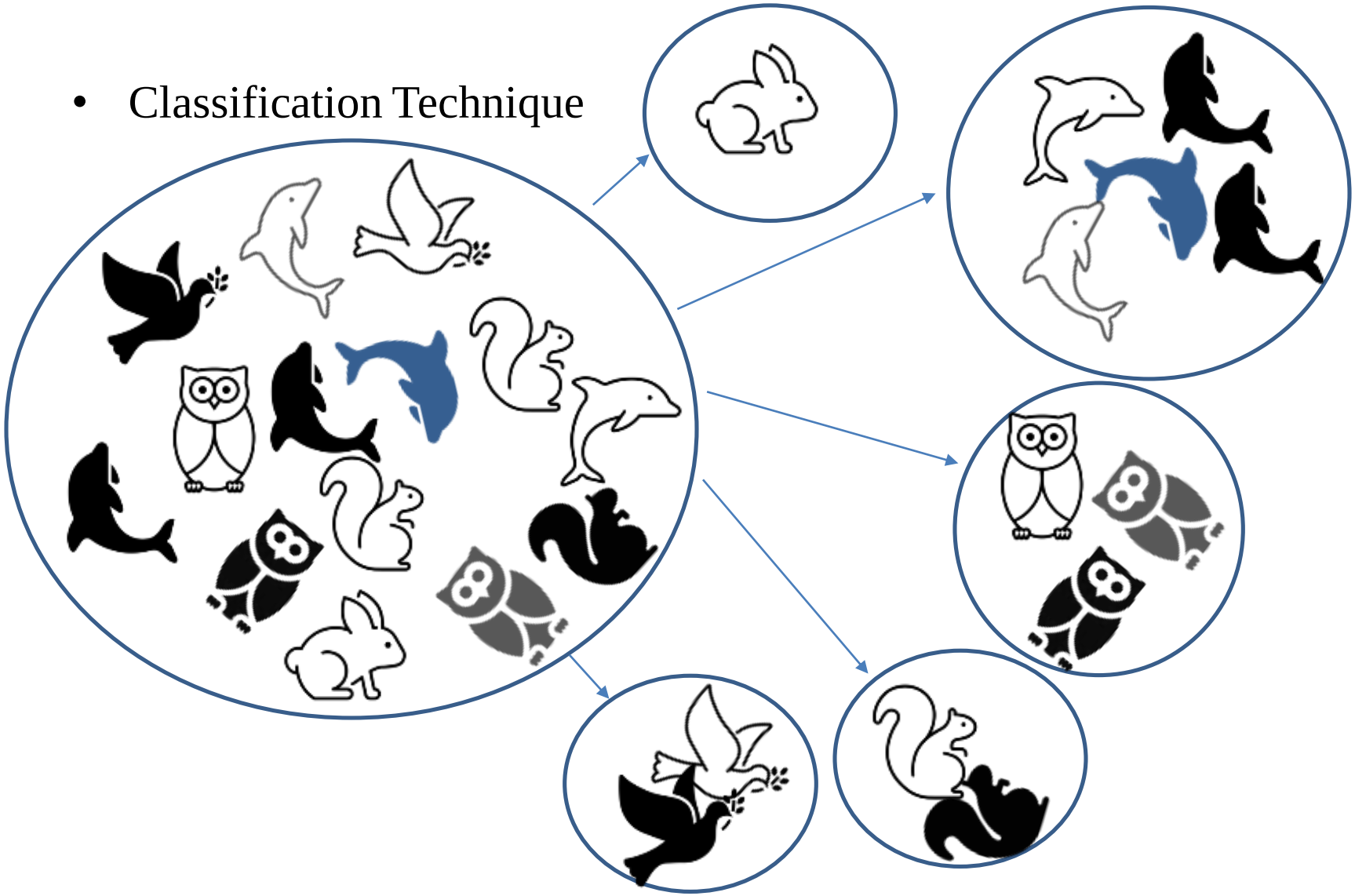


Supervised Learning

- Regression Technique

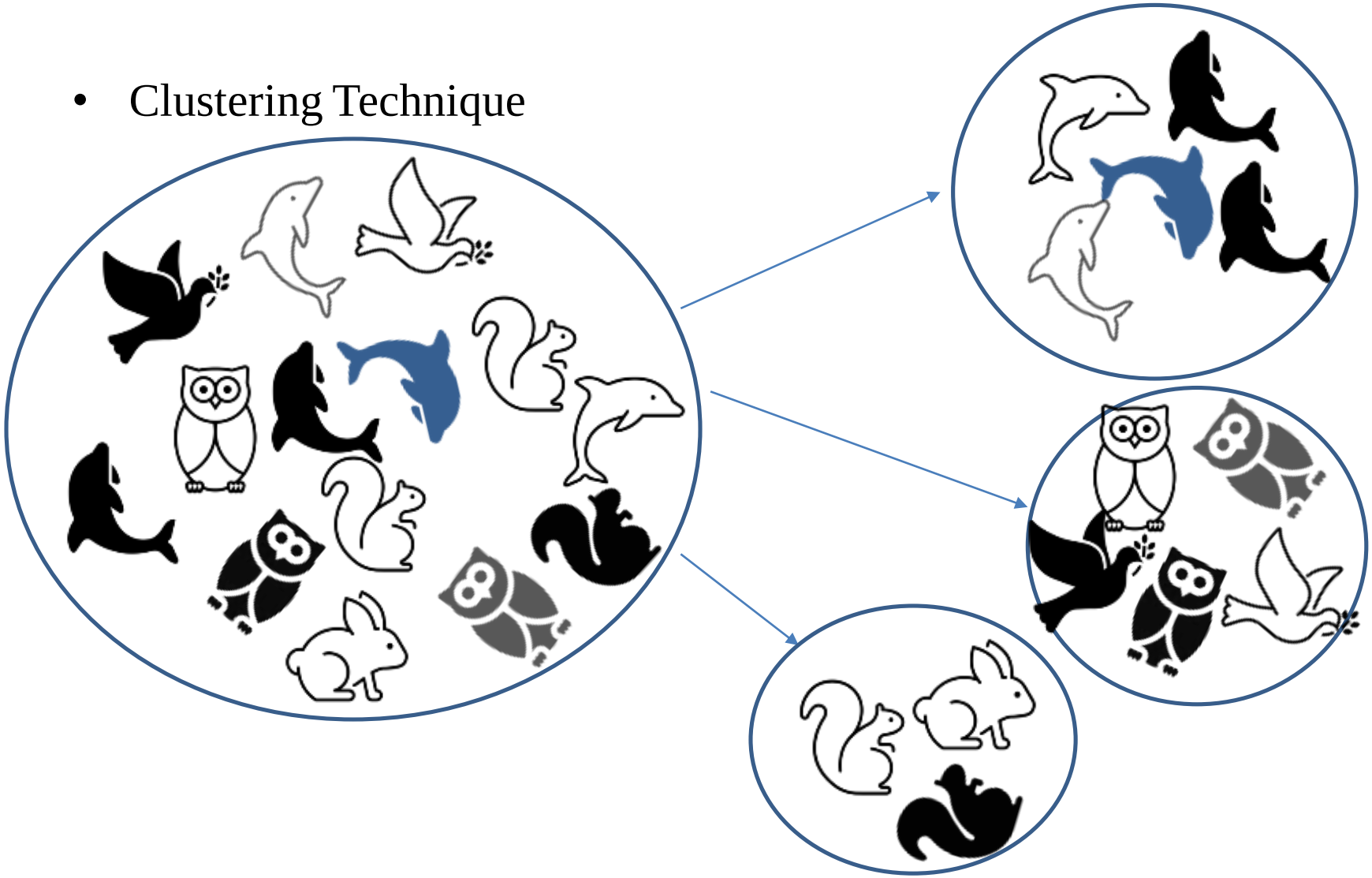
Supervised Learning

- Classification Technique



Unsupervised Learning

- Clustering Technique



Reinforced Learning