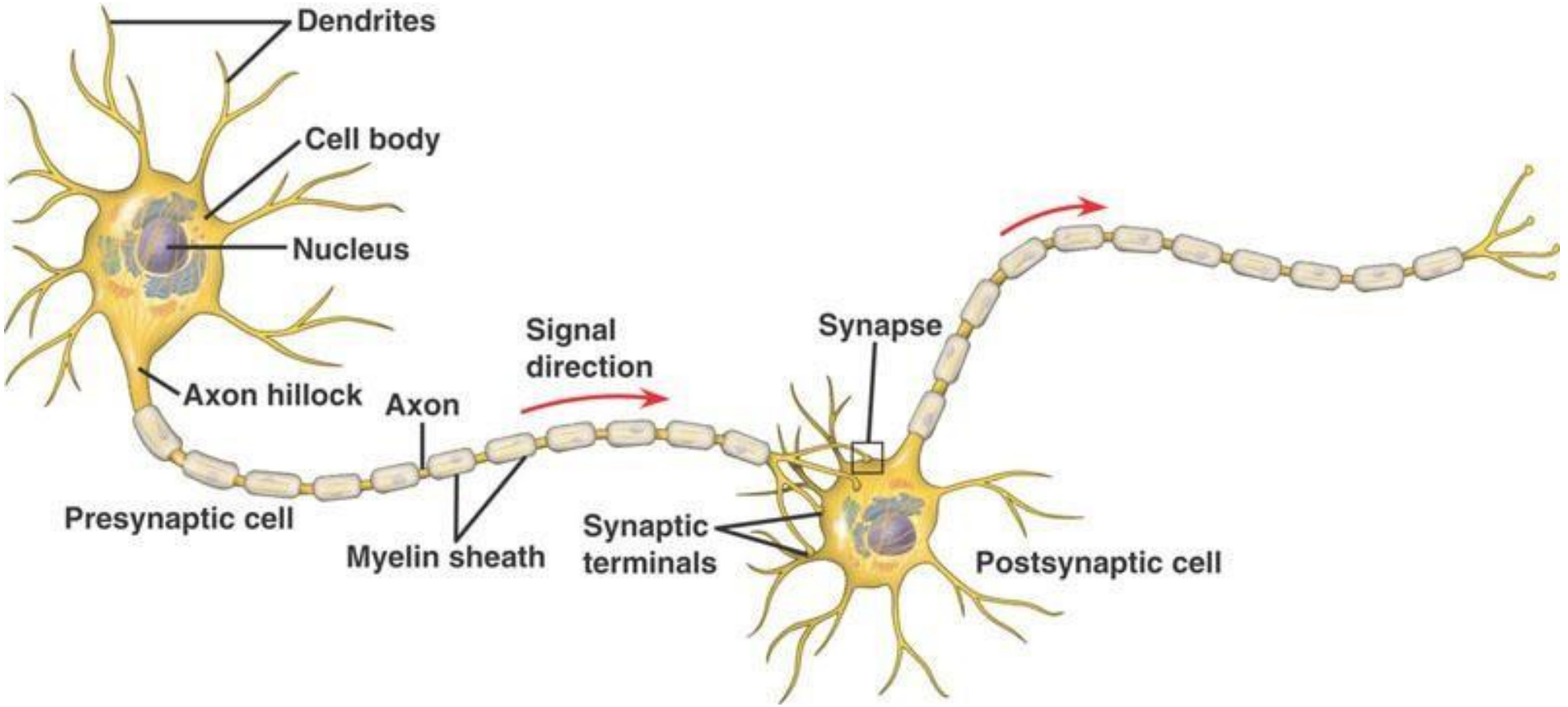
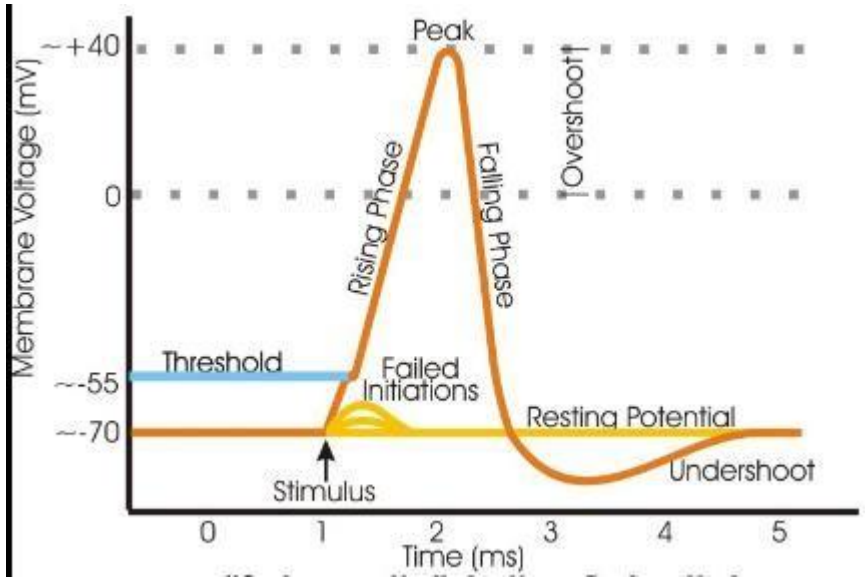
Overview of Neural Network

Sarbin Sayami
Central Department of Computer Science and Information Technology
Kirtipur, Kathmandu

Extended view of neuron



Excitation and Inhibition



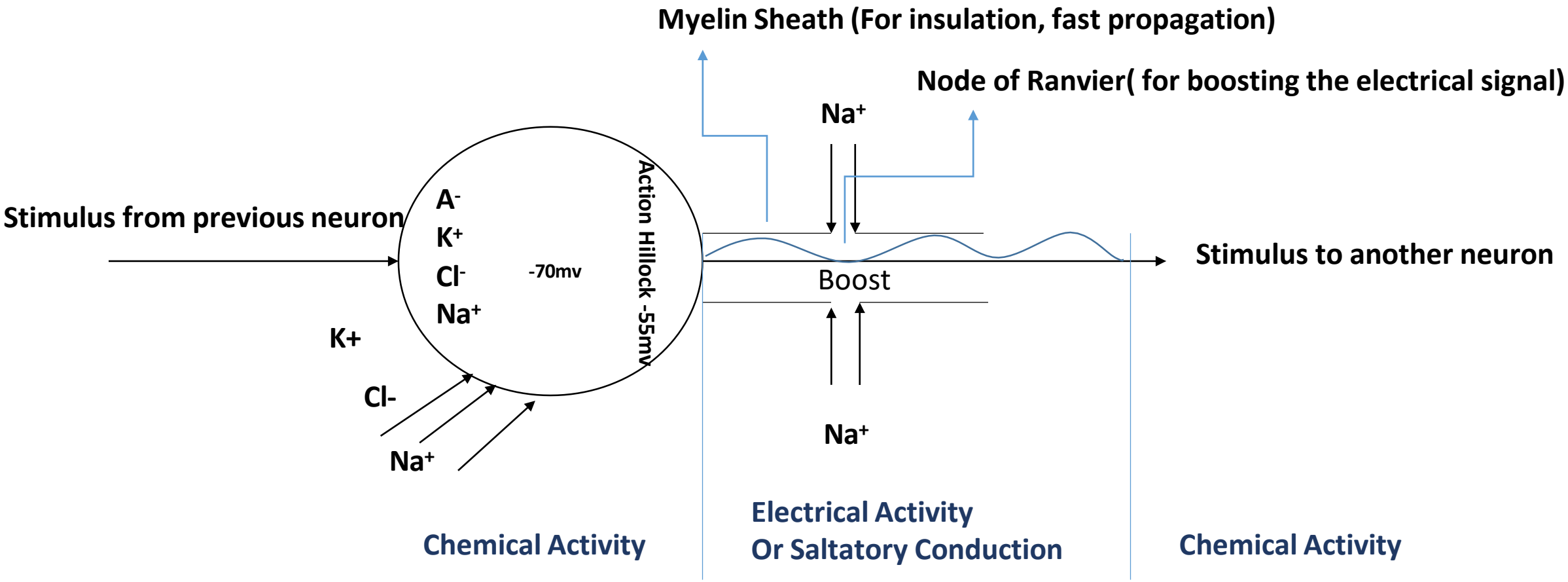
The voltage inside the neuron is -70 mv (resting potential) and upon getting stimulus from sensory organs, its potential might

- Undershoot: Inhibitory neurons (No fire)
- Overshoot : Excitatory neurons (fire) , once it reaches or crosses -55mv which the required threshold

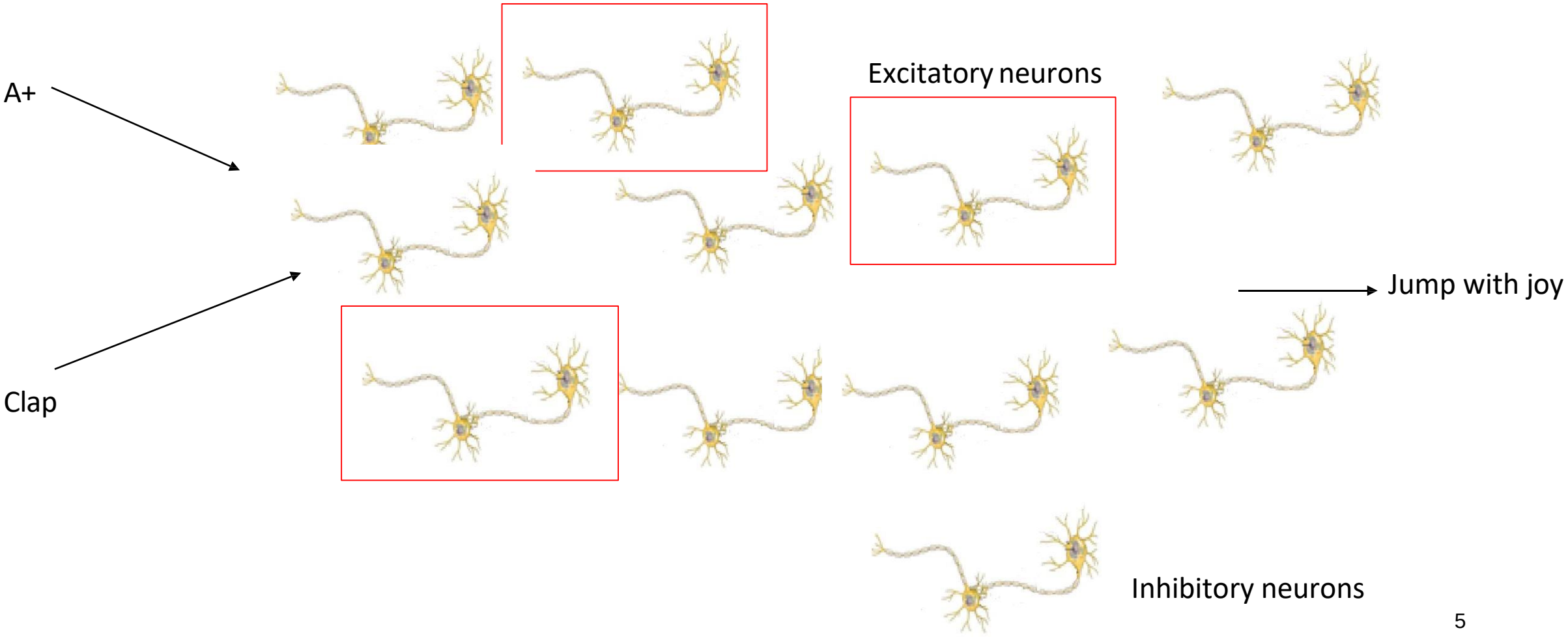
It works on "ALL OR NONE" principle.

i.e. There are no big or small action potentials in one nerve cell - all action potentials are the same size. Therefore, the neuron either does not reach the threshold or a full action potential is fired

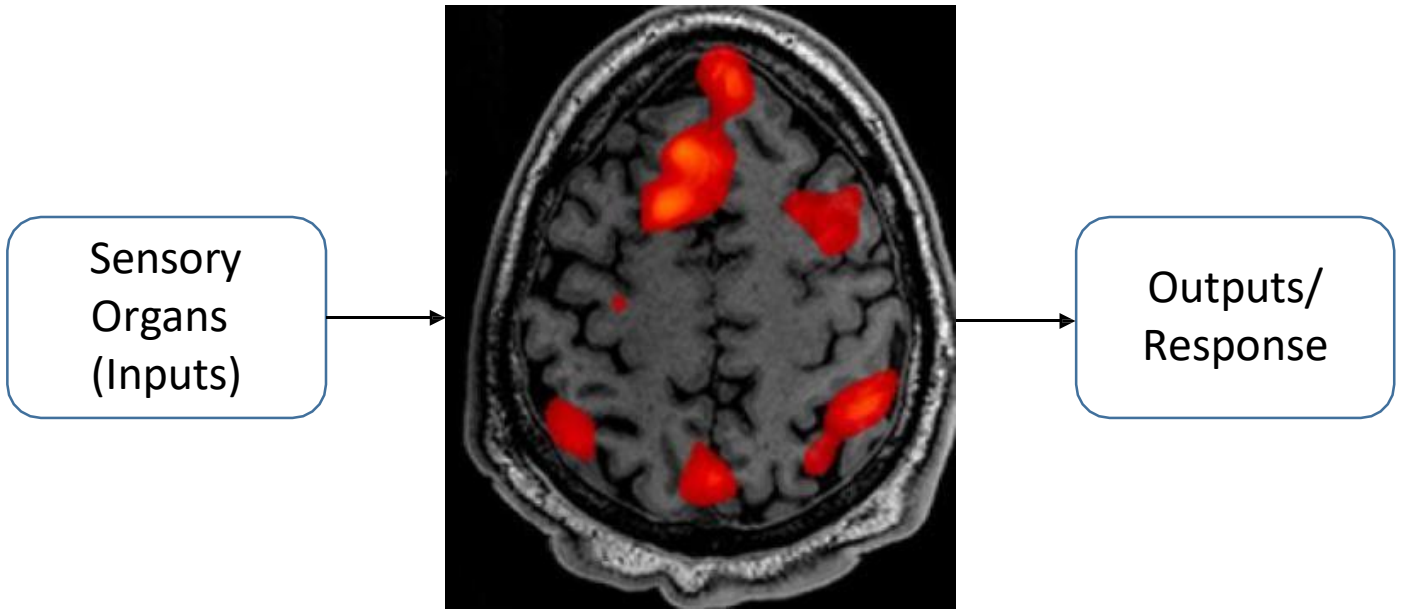
Excitation and Inhibition(continued)



Excitation and Inhibition(continued)

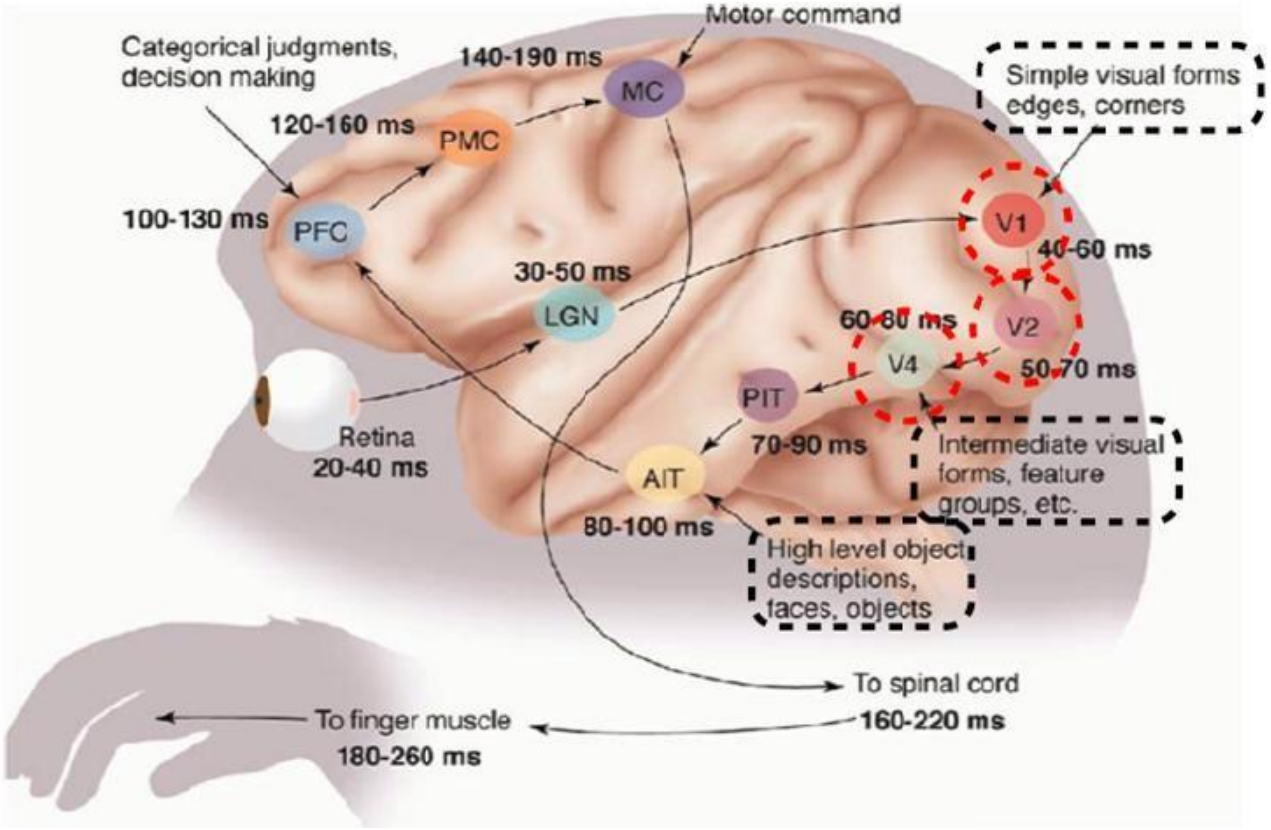


Excitation and Inhibition(continued)



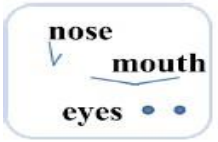
Red colors means active region, excitatory neurons, others are inactive regions, inhibitory neurons

Hierarchical processing of visual cortex of brain



AIT - Auditory Integration Training
LGN - Lateral Geniculate Nucleus
PFC - Prefrontal Cortex
PMC - Pontine Micturition Center
PIT - Posterior Inferotemporal area
V1,V2... - Visual Areas

Layer 1: detect edges & corners



Layer 2: form feature groups

Layer 3: detect high level objects, faces, etc.

Computational properties of the Human Brain

- Even though the complete architecture (physiological) of brain is available, the fundamental computation is a mystery.
- Integrates and stores experiences
- Considers new experiences in the context of stored experiences
- Makes accurate predictions about new situations on the basis of previously self- organized experiences.

Computational properties of the Human Brain (continued)

- Does not require perfect information
- Represents a fault-tolerant architecture
- Seems to have available, perhaps unused, neurons ready for use
- Homeostatic in nature
- It is not well equipped to solve roots of high-order polynomials, invert large-dimension matrices, and solve complex sets of differential equations as its inherent property is “Survival of the fittest”.

Computational properties of the Human Brain (continued)

- An average human brain has around 100 billion neurons
- In reality, there is not just a single neuron but there is a massively parallel interconnected network of neurons
- The sense organs relay signals /information to the lowest layer of neurons
- Some of these neurons may fire (in red) in response to this signals/ information and in turn relay information to other connected neurons
- These neurons may also fire (in red) and the process continues eventually resulting in a response (jump with joy in this case)

Computational properties of the Human Brain (continued)

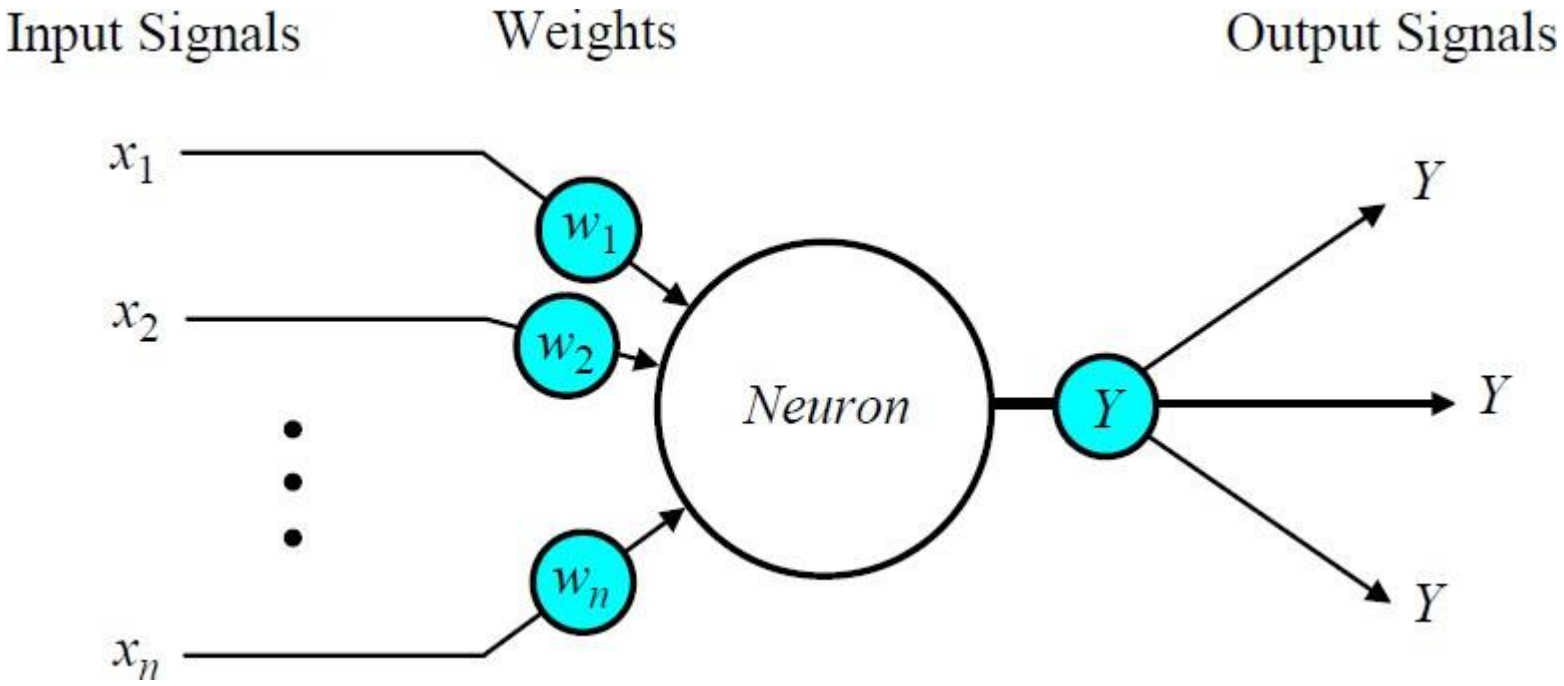
- The firing takes place if the potential exceeds the required threshold
- Actually there is division of work in the massively parallel network
- Therefore, each neuron may perform a certain role or respond to a certain stimulus
- Similarly, each layer within the network may perform certain roles

Analogy of brain and Artificial Neural Network

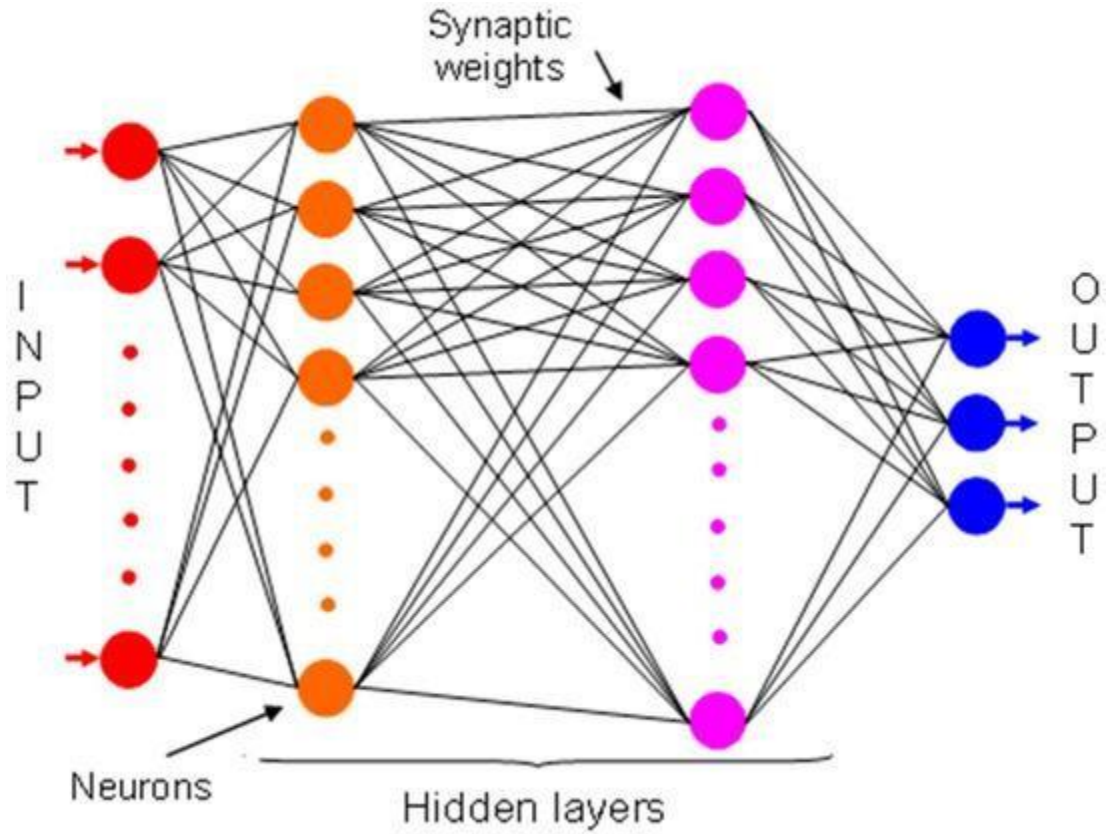
	Brain	Artificial Neural Network
Structure	Connection, layers, Hierarchies	Input, hidden, output layers, number of neurons within these layers and connection
Learning	Importance of signals, Excitatory and Inhibitory characteristics, Integration of signals	Weight, thresholding, activation function(fire or not fire),learning rules to adjust the weights to minimize error of output

Analogy of brain and Artificial Neural Network (continued)

Biological Neural Network	***Artificial Neural Network***
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight



ANN Structure



History of Artificial Neural Network

Date	Researcher Name	Achievements
427 – 322 B.C.	Plato and Aristotle	Conceived theories about the brain and the thinking
1596 - 1650	Descartes	Influenced by the platonic ideas proposed that mind and matter are two different kinds of substance, of opposite nature, able to exist in an independent way, which have the possibility to interact
1936	Alan Turing	Studied the brain as a form to see the world of computing
1943 (First Wave of Interest)	Warren McCulloch and Walter Pitts	Created the theory about functioning of neurons, Linear Threshold
1949	Donald Hebb	Established the connection between psychology and physiology, Hebbian Learning (Cell Assemblies)
1950s	Marvin Minsky	Re-inforcement Learning

History of Artificial Neural Network

Date	Researcher Name	Achievements
1957 (second Wave of Interest)	Frank Rosenblatt	Developed the Perceptron, the first Artificial Neural Network
1959	Bernard Widrow and Marcial Hoff	Created the Adaline Model
1969 (Silent Period)	Marvin Minsky and Seymour Papert	Published the book Perceptron where rigorous analysis of the limitation of Perceptron was carried out provoking a disappointment for the use of artificial neural networks
Silent Period (Worked on the Sidelines)	Stephen Grossberg, Teuvo Kohonen, Shun-Ichi Amari	
1982	James Anderson, Kunihiko Fukushimica, Teuvo Kohonen, and John Hopfield	Made important works that allowed the rebirth of the interest for the artificial neural networks. Reunion U.S – Japan: Begin the development of what was known as thinking computers for their application in robotics
1980s (Third Wave of Interest)	Hopfield	Hopfield's Network
1980s	Le Cun, Parker and the group of Rumelhart, Hilton and Williams	Back-propagation
1990s		Hybrid Model

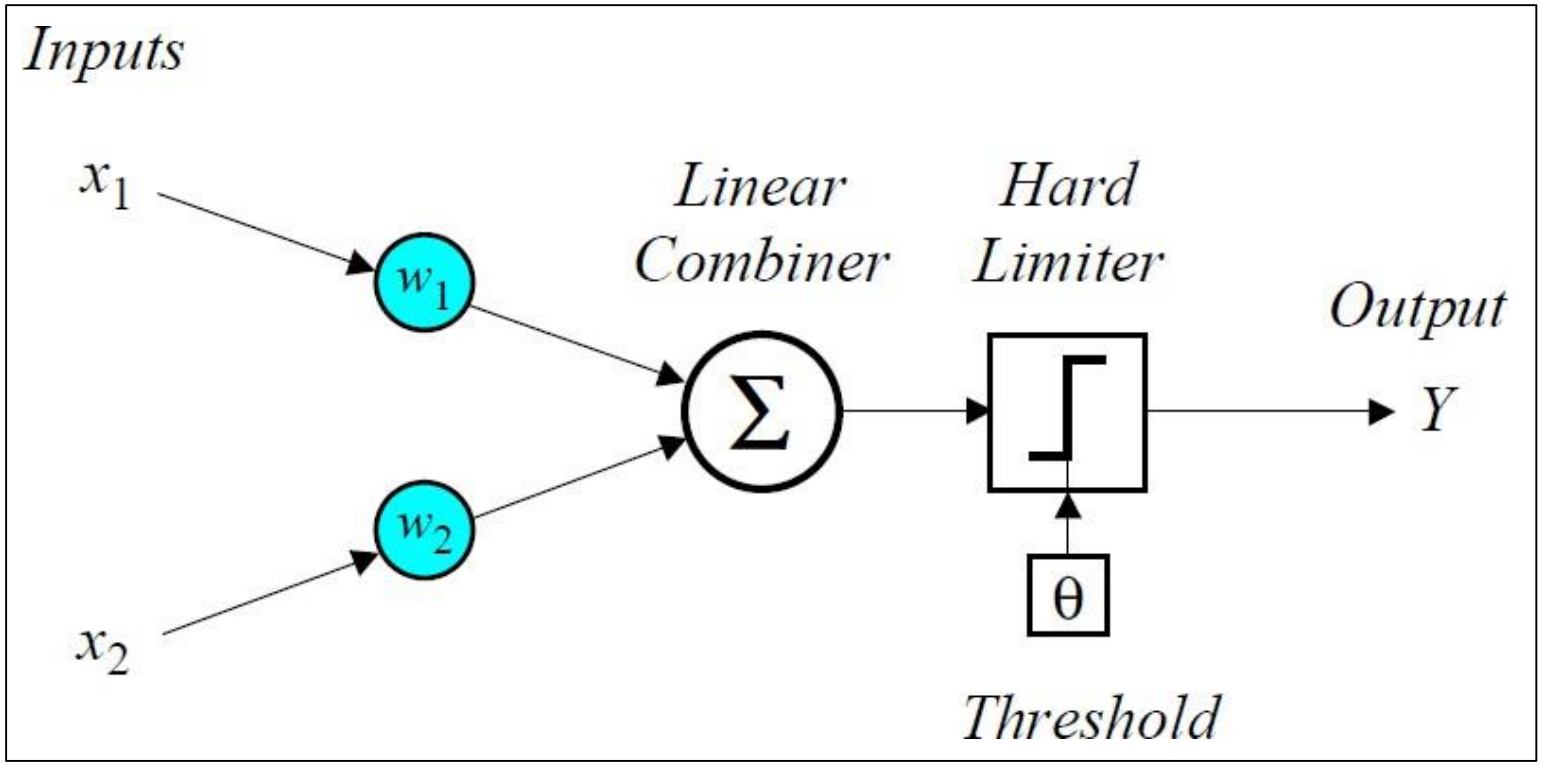
Mcculloch and Pitts Neuron

Assumptions:

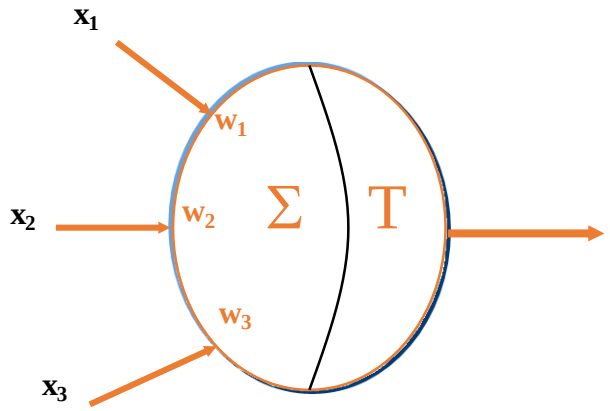
1. The activity of the neuron is an “all-or-none” process
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron
3. The only significant delay within the nervous system is synaptic delay
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time
5. The structure of the net does not change with time

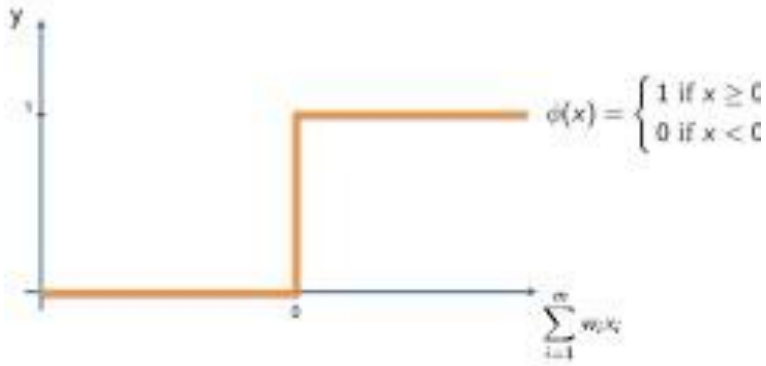
Mcculloch and Pitts Neuron (continued)

- Neural events and the relations among them can be treated by means of propositional logic. It is found that the behaviors of every net can be described in these terms.



Mcculloch and Pitts Neuron (continued)





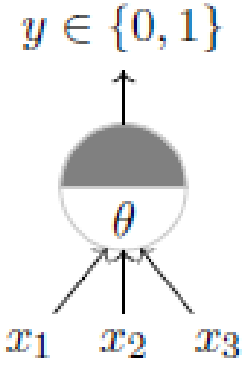
$$x_1w_1 + x_2w_2 + x_3w_3 >= T$$

weights [-1 , + 1]

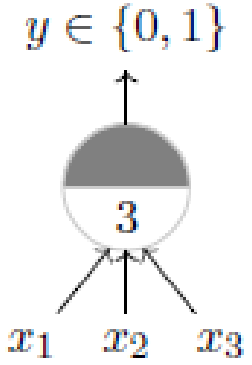
It has a threshold value, T

Its output is 1 (fired) or 0 (not fired)

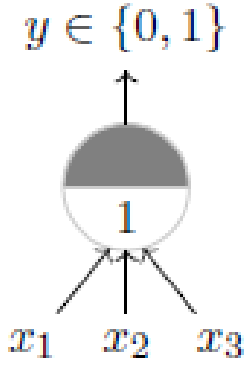
Mcculloch and Pitts Neuron (continued)



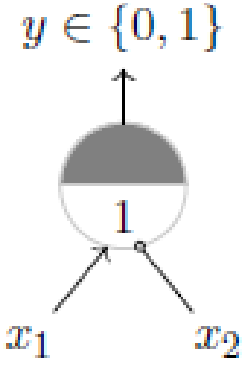
A McCulloch Pitts unit



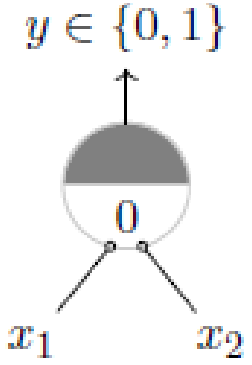
AND function



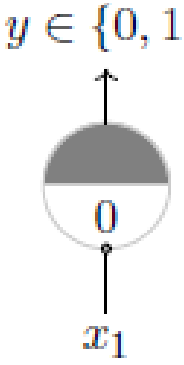
OR function



x_1 AND $!x_2$*



NOR function



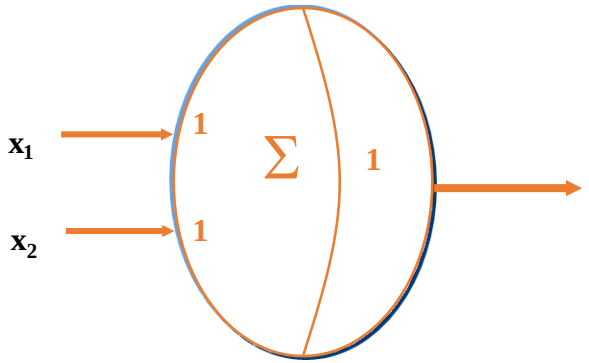
NOT function

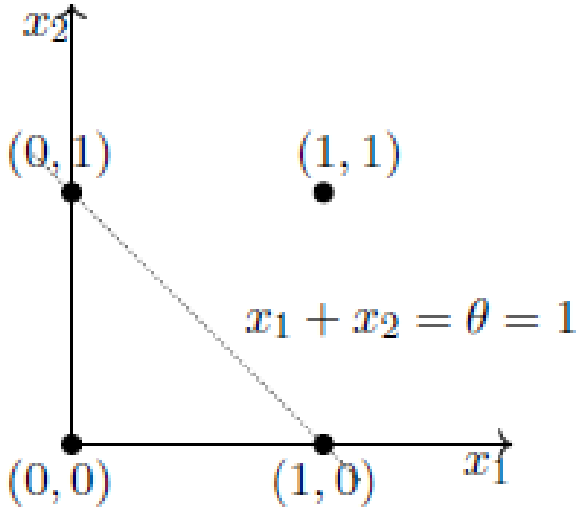
*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0

Mcculloch and Pitts Neuron (continued)

OR Example

x1	x2	F
0	0	0
0	1	1
1	0	1
1	1	1



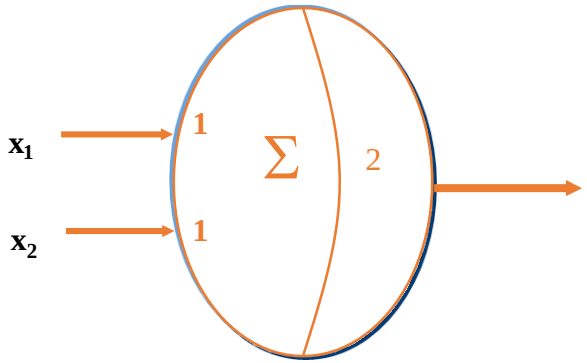


0*1+0*1=0,	0<1,	F=0
0*1+1*1=1,	1=1,	F=1
1*1+0*1=1,	1=1,	F=1
1*1+1*1=2,	2>1,	F=1

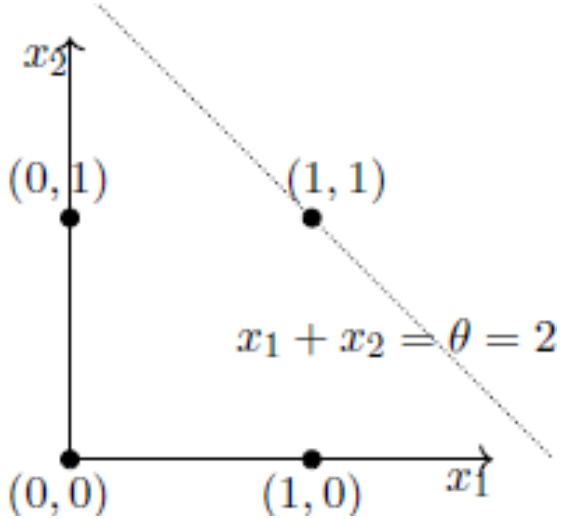
Mcculloch and Pitts Neuron (continued)

AND Example

x1	x2	F
0	0	0
0	1	0
1	0	0
1	1	1



0*1+0*1=0,	0<2,	F=0
0*1+1*1=1,	1<2,	F=0
1*1+0*1=1,	1<2,	F=0
1*1+1*1=2,	2=2,	F=1



Mcculloch and Pitts Neuron (continued)

Demerits

- Cannot realize EX-OR gate i.e. it cannot handle non linear data
- Can only handle Boolean inputs
- Does not have learning mechanism (i.e. no mechanism to update weights)

Hebbian Learning

Hebb's Postulate

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

- When a weight contributes to firing a neuron, the weight is increased.
- When a weight acts to inhibit the firing of a neuron, the weight is decreased

Hebbian Learning (continued)

- Supervised Hebbian Learning
  - Weights are strengthened by the desired response
- Unsupervised Hebbian Learning
  - Weights are strengthened by the actual response

Hebbian Learning (continued)

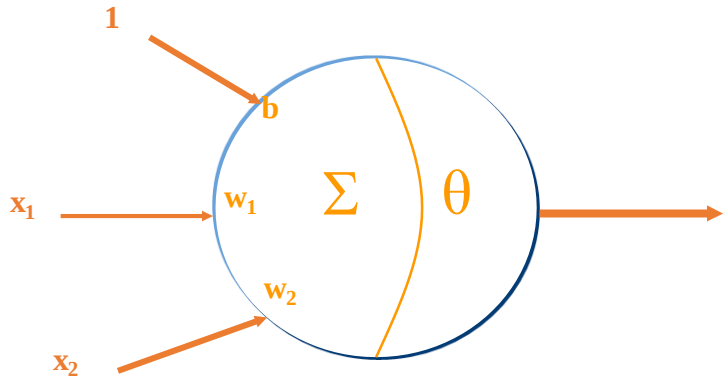
Supervised Hebbian Learning

- **Step 0**: initialize all weights to 0
- **Step 1**: Given a training input, s, with its target output, t, set the activations of the input units: $x_i = s_i$
- **Step 2**: Set the activation of the output unit to the target value: y = t, y is the desired output
- **Step 3**: Adjust the weights: $w_i(new) = w_i(old) + x_i y$
- **Step 4**: Adjust the bias (just like the weights): b(new) = b(old) + y

Hebbian Learning (continued)

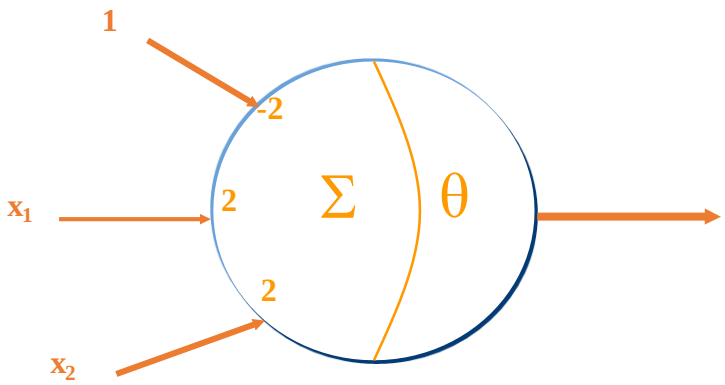
Gate Realization

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



Iteration	Input	$w_1(new) = w_1(old) + x_1y$	$w_2(new) = w_2(old) + x_2y$	$b(new) = b(old) + y$
1	1 1 1	**0 + 1*1 = 1**	**0 + 1*1 = 1**	**0 + 1 = 1**
2	1 -1 1	**1 + 1(-1) = 0**	**1 + (-1)(-1) = 2**	**1 + (-1) = 0**
3	-1 1 1	**0 + (-1)(-1) = 1**	**2 + 1(-1) = 1**	**0 + (-1) = -1**
4	-1 -1 1	**1 + (-1)(-1) = 2**	**1 + (-1)(-1) = 2**	**-1 + (-1) = -2**

Hebbian Learning (continued)



1*2 + 1*2 + 1*(-2) = 2 > 0 = 1

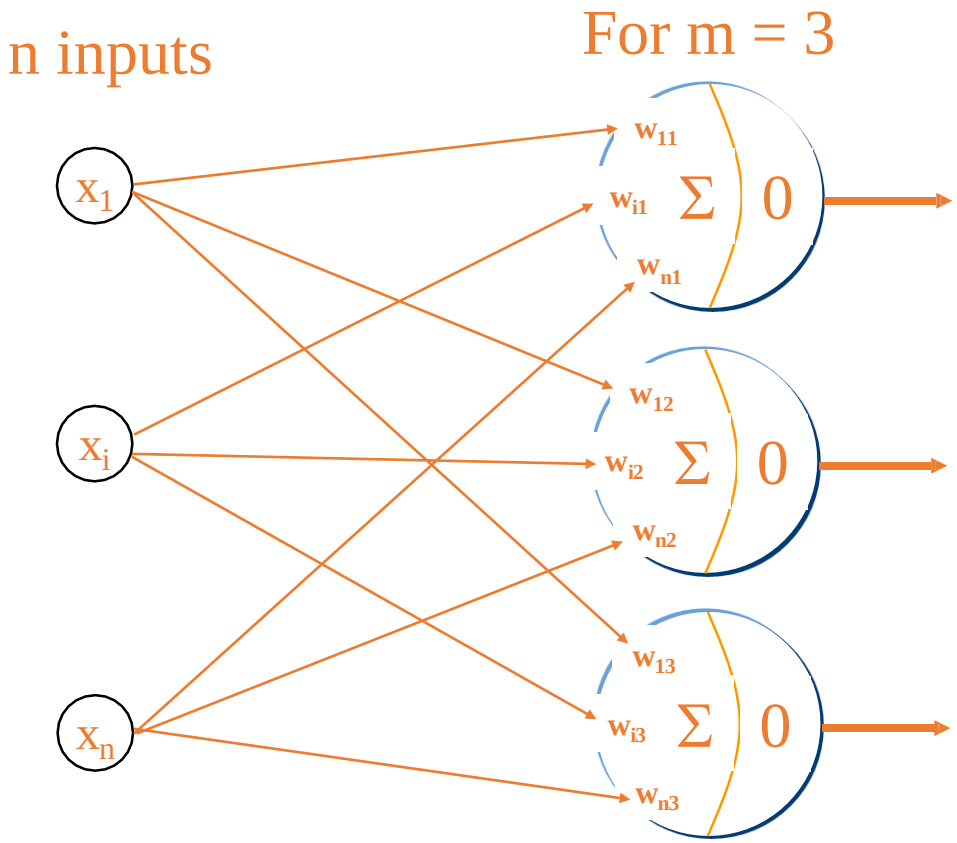
(-1)*2 + 1*2 +1*(-2) = -2 < 0 = -1

1*2 + (-1)*2 + 1*(-2) = -2 < 0 = -1

(-1)*2 + (-1)*2 + 1 *(-2) = -6 <0 = -1

Heteroassociative Architecture

- There are n input units and m output units with each input connected to each output unit.



Activation Functions:

For bipolar targets:

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

For binary targets:

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i <= 0 \end{cases}$$

Heteroassociative Architecture (continued)

- GOAL: build a neural network which will associate the following two sets of patterns

$s_1 = (1 \ -1 \ -1 \ -1)$ $t_1 = (1 \ -1 \ -1)$

$s_2 = (-1 \ 1 \ -1 \ -1)$ $t_2 = (1 \ -1 \ 1)$

$s_3 = (-1 \ -1 \ 1 \ -1)$ $t_3 = (-1 \ 1 \ -1)$

$s_4 = (-1 \ -1 \ -1 \ 1)$ $t_4 = (-1 \ 1 \ 1)$

The process will involve 4 input neurons and 3 output neurons

The algorithm involves finding the four outer products and adding them

Heteroassociative Architecture (continued)

Pattern pair 1:

$$\begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

Pattern pair 2:

$$\begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix}$$

Pattern pair 3:

$$\begin{bmatrix} -1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

Pattern pair 4:

$$\begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$$

Heteroassociative Architecture (continued)

- Add all four individual weight matrices to produce the final weight matrix:

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & -2 & -2 \\ 2 & -2 & 2 \\ -2 & 2 & -2 \\ -2 & 2 & 2 \end{bmatrix}$$

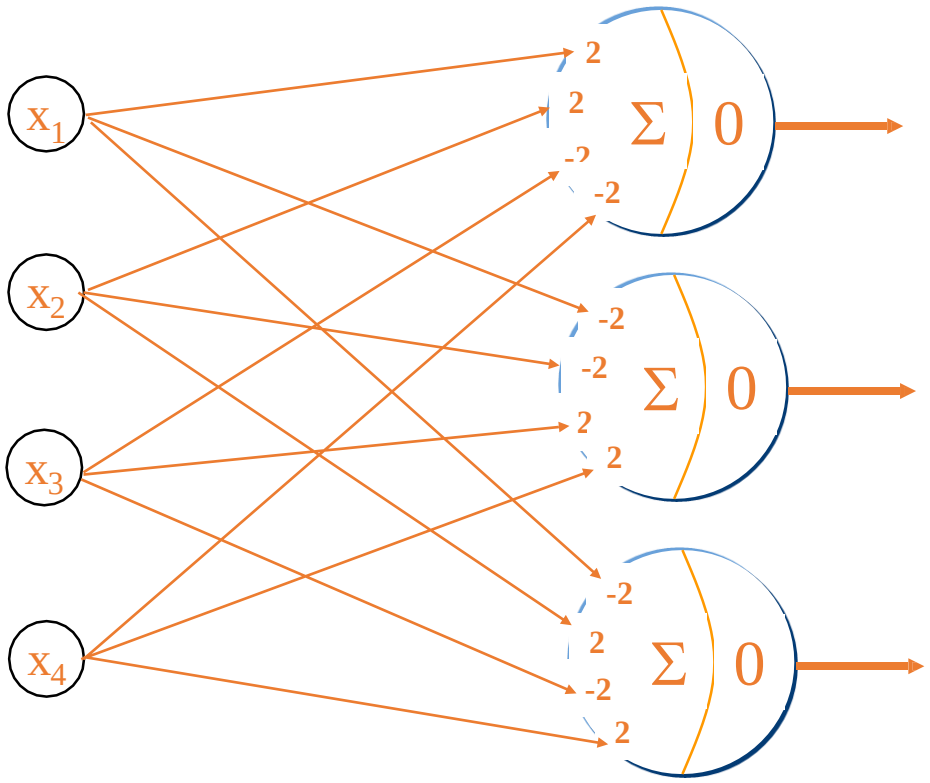
Each column defines the weights for an output neuron

Heteroassociative Architecture (continued)

- General Structure

$$\begin{bmatrix} 2 & -2 & -2 \\ 2 & -2 & 2 \\ -2 & 2 & -2 \\ -2 & 2 & 2 \end{bmatrix}$$

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

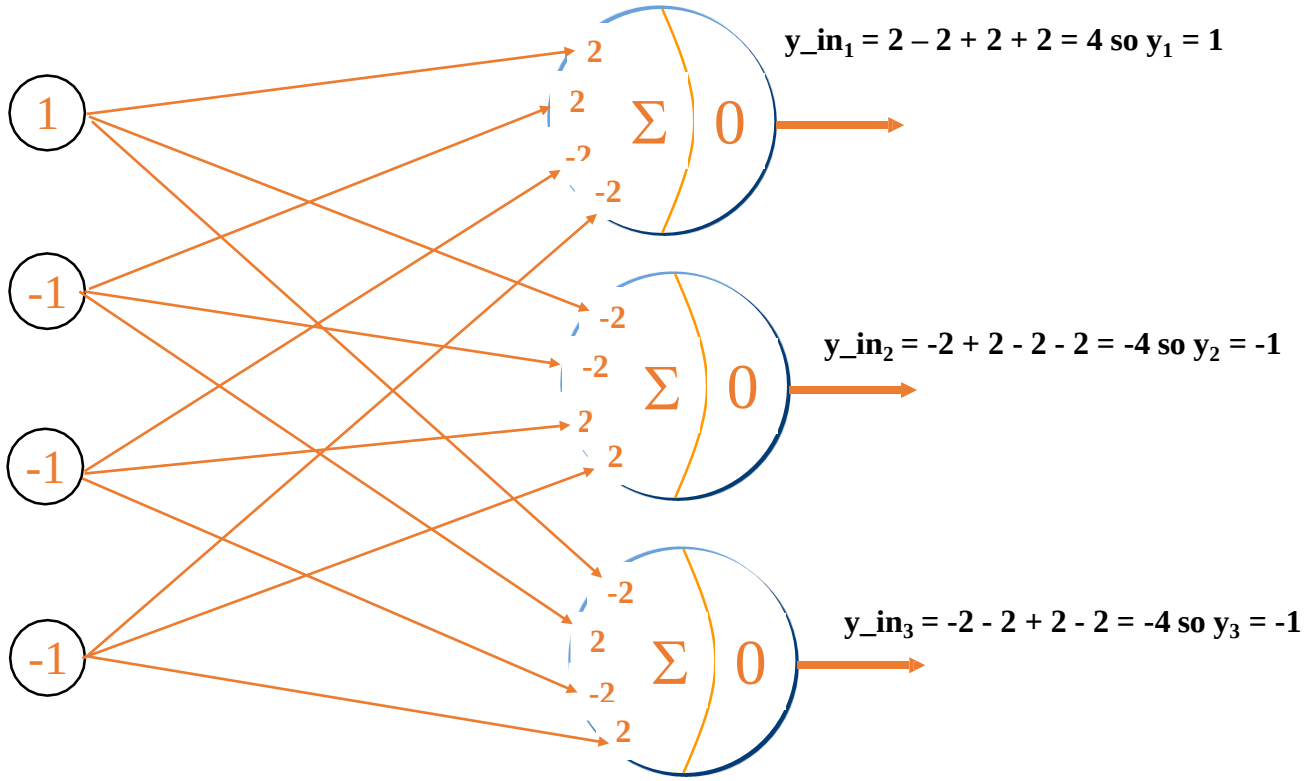


Heteroassociative Architecture (continued)

- First input pattern:

$s_1 = (\ 1\ \ -1\ \ -1\ \ -1)\quad t_1 = (\ 1\ \ -1\ \ -1)$

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

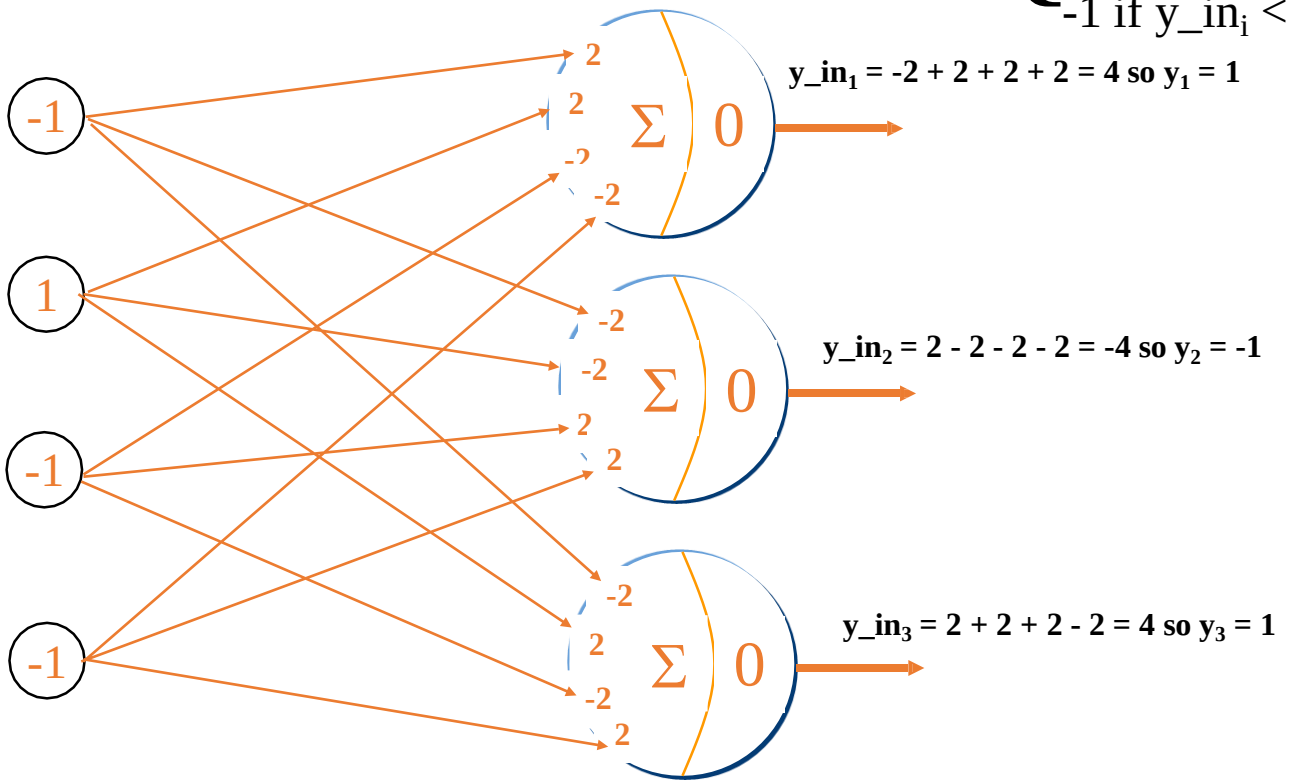


Heteroassociative Architecture (continued)

- Second input pattern:

$s_2 = (-1 \quad 1 \quad -1 \quad -1) \quad t_2 = (1 \quad -1 \quad 1)$

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

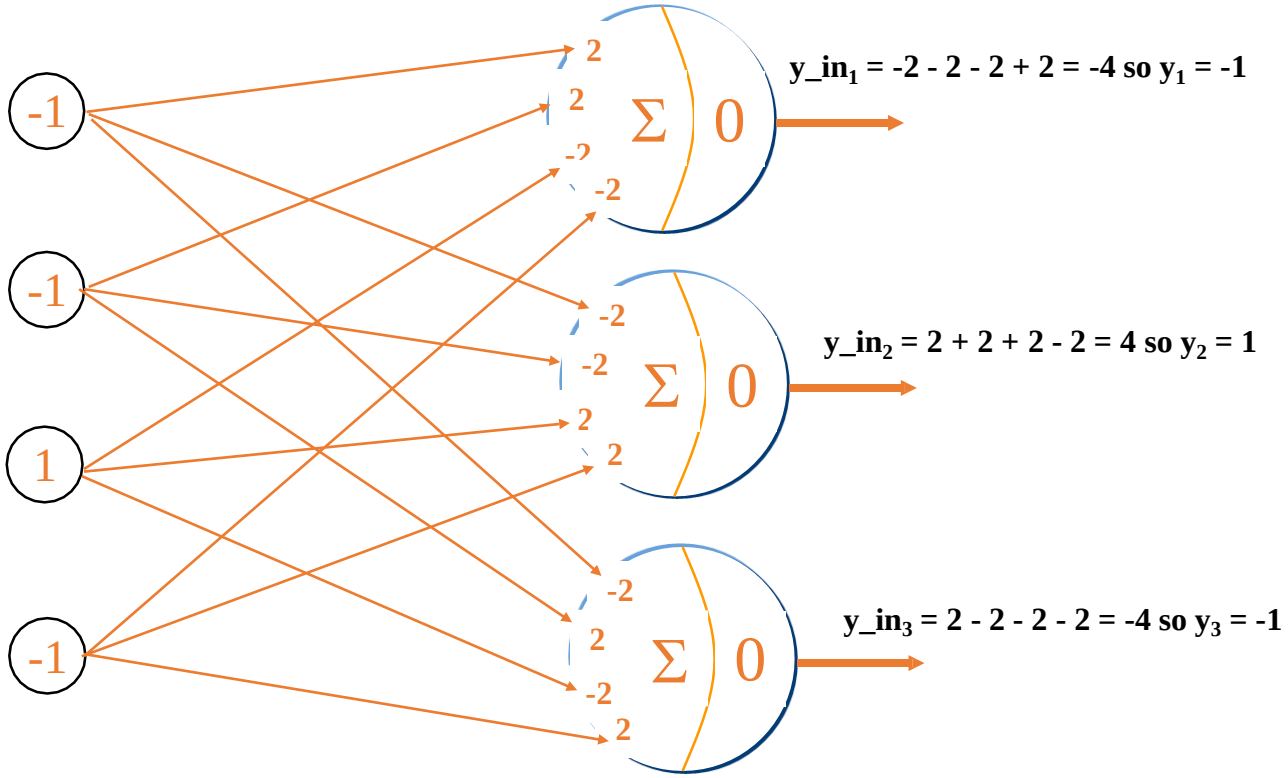


Heteroassociative Architecture (continued)

- Third input pattern:

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$

$s_3 = (-1 \quad -1 \quad 1 \quad -1) \quad t_3 = (-1 \quad 1 \quad -1)$

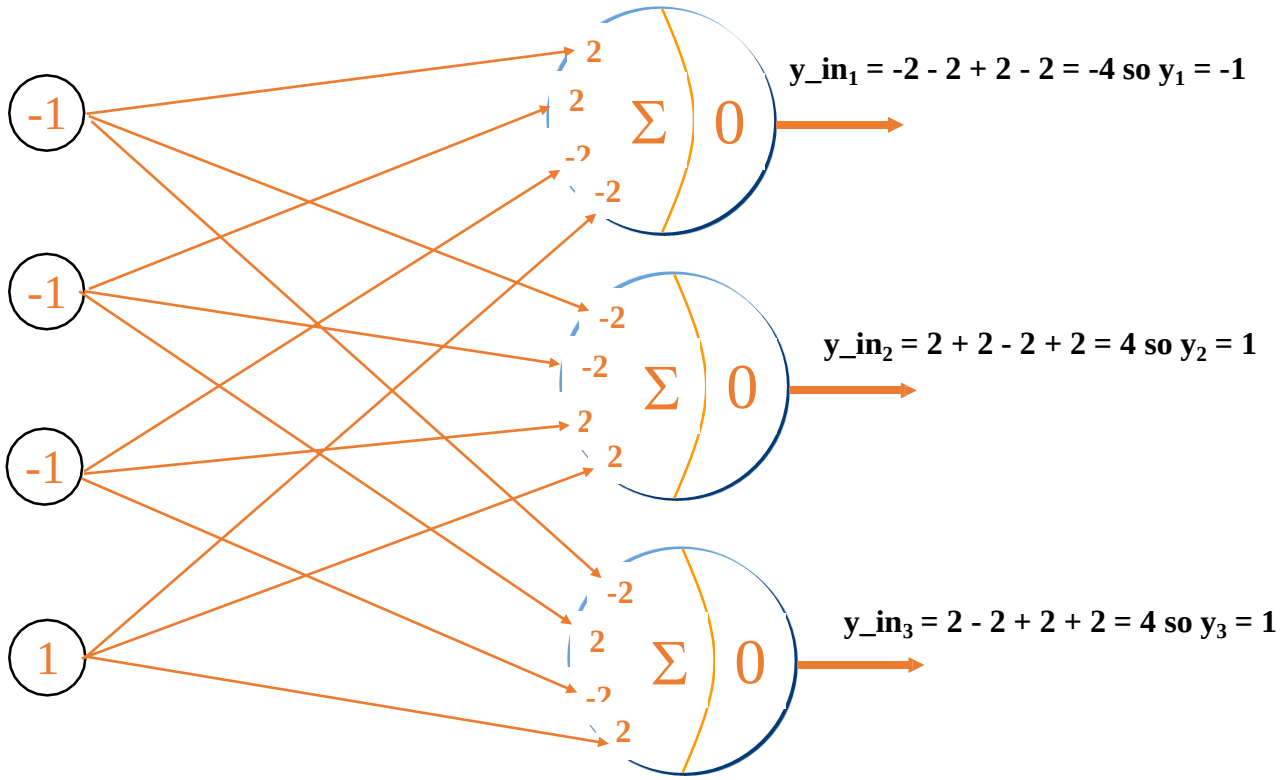


Heteroassociative Architecture (continued)

- Fourth input pattern:

$s_4 = (-1 \quad -1 \quad -1 \quad 1) \quad t_4 = (-1 \quad 1 \quad 1)$

$$y_i = \begin{cases} 1 \text{ if } y\_in_i > 0 \\ 0 \text{ if } y\_in_i = 0 \\ -1 \text{ if } y\_in_i < 0 \end{cases}$$



Perceptron

Rules for feedback

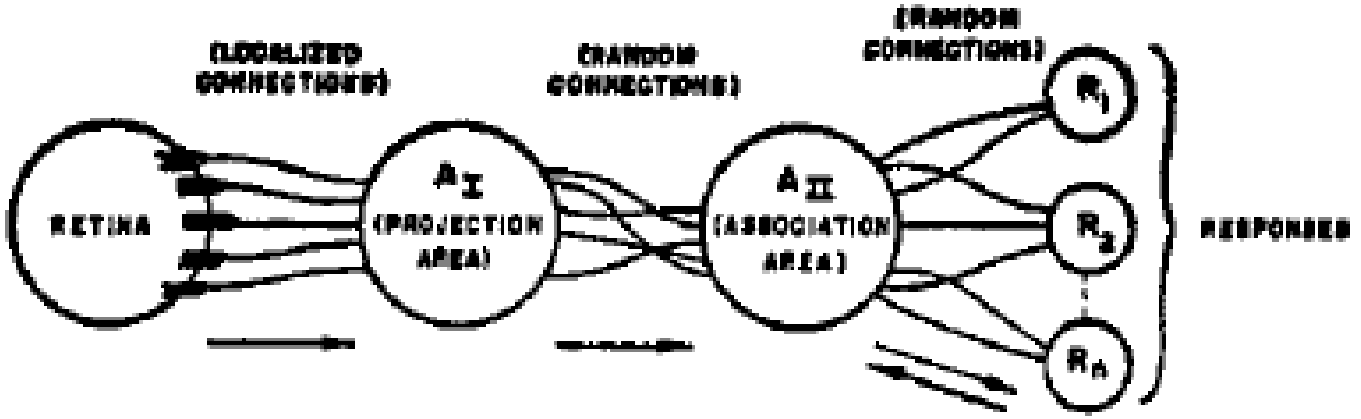
a) Each response has excitatory feedback connections to the cells in its own source-set

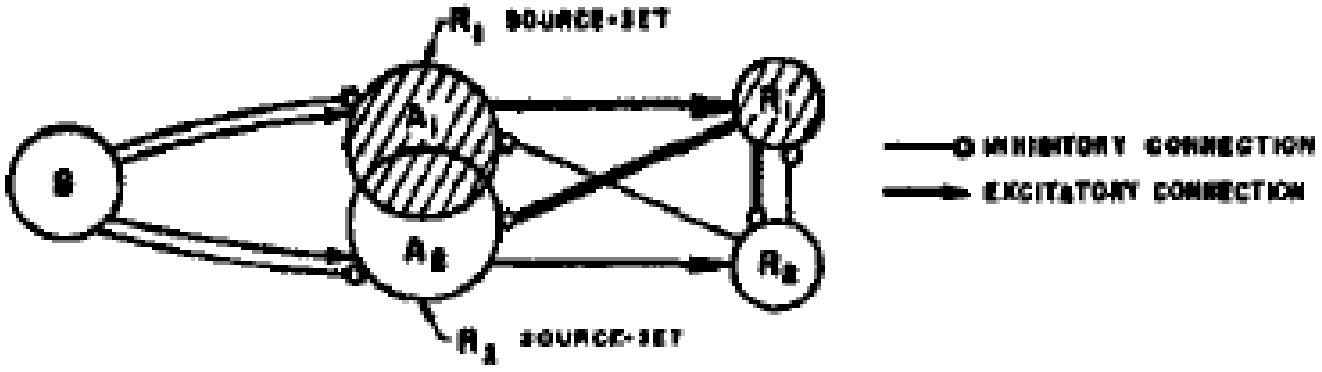
b) Each response has inhibitory feedback connections to the complement of its own source-set

Rule a) is not realistic as it creates bursting effects

Rule b) is more realistic

Perceptron (continued)





Perceptron (continued)

Pa – Expected proportion of A units activated by a stimulus of a given size

R – proportions of S-points activated by the stimulus

x – number of excitatory connections to each A-unit

y – number of inhibitoryconnections to each A-unit

Θ – threshold of A-unit

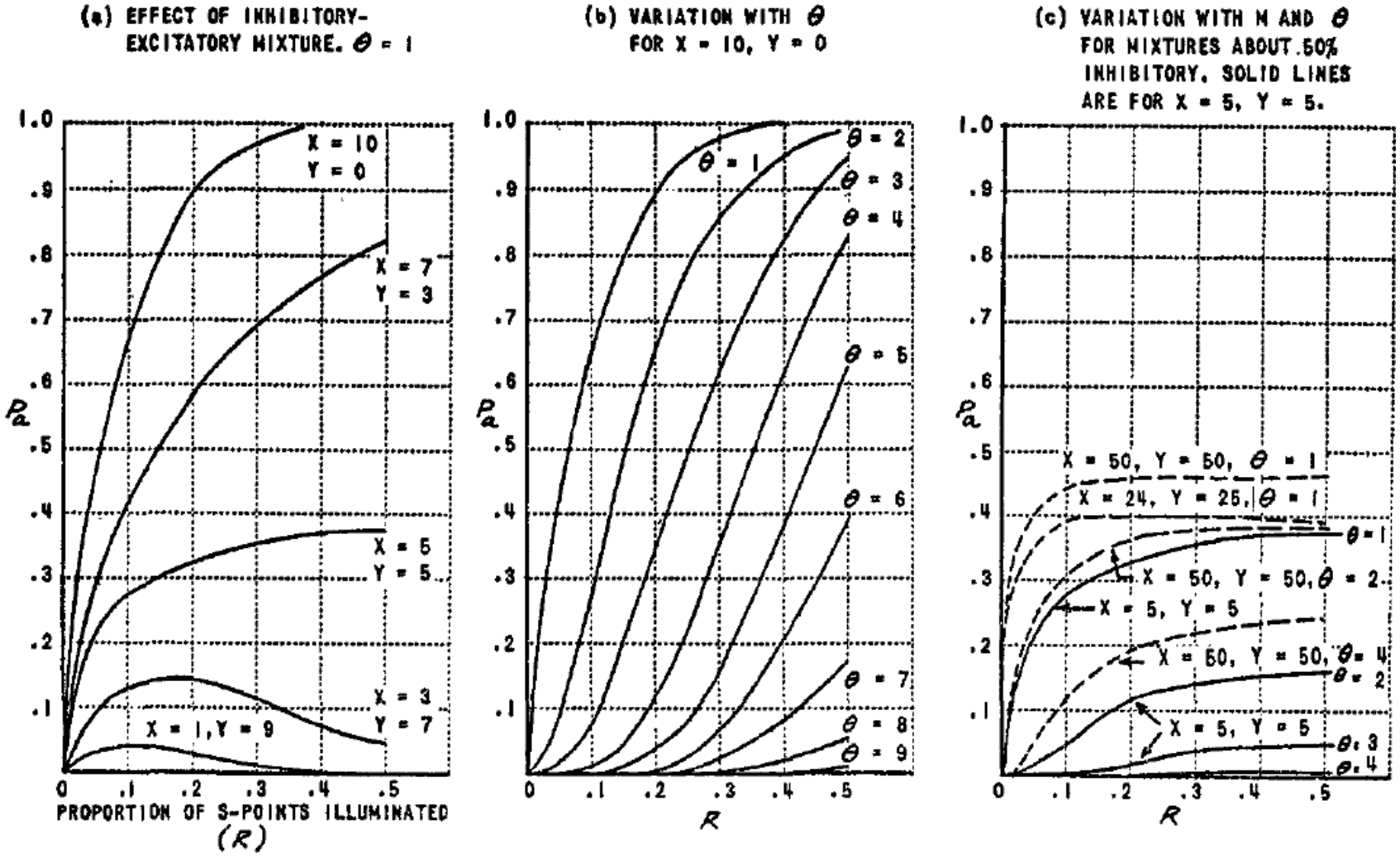
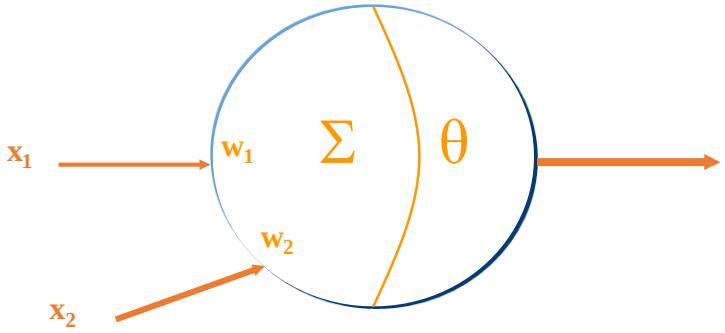


Fig: P_a as a function of retinal area illuminated

Types of perceptron

- Alpha
- Beta
- Gamma

Perceptron (continued)



Activation Function

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

Learning by Delta Rule

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

$$\Delta w_i(t) = \alpha . x_i(t) . e(t)$$

e(t) is the error

e(t)= Desired output – Actual output

α is the learning rate

Perceptron (continued)

Step 1: Set initial weights and threshold to random numbers in the range [-0.5,0.5]

Step 2: Y(t) = step function($\sum_{1=0}^{n} xi(t)$w$_i$(t) – θ), where n is the number of the perceptron inputs

Step 3: Calculate error e(t)= Desired output – Actual output

Step 4: If there is error, adjust the weights by using the Delta rule

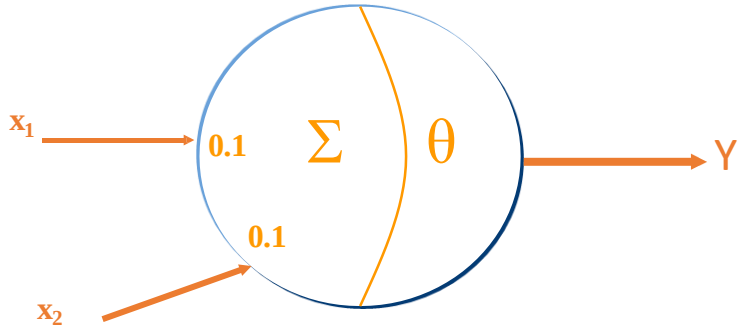
Step 5: Repeat step 2 to step 4 until convergence

Perceptron (continued)

Realization of The logical operation -AND

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$



ADALINE

- Variation on the Perceptron
  - inputs are +1 or -1
  - outputs are +1 or -1
  - uses a bias input
- Differences
  - trained using the Delta Rule which is also known as the least mean squares (LMS) or Widrow-Hoff rule
  - the activation function, during training is the identity function
  - after training the activation is a threshold function

ADALINE (continued)

ADALINE Algorithm

- **Step 0: initialize the weights to small random values and select a learning rate, α**
- **Step 1: for each input vector s, with target output, t set the inputs to s**
- **Step 2: compute the neuron inputs**
- **Step 3: use the delta rule to update the bias and weights**
- **Step 4: stop if the largest weight change across all the training samples is less than a specified tolerance, otherwise cycle through the training set again**

Neuron input

$$y\_in = b + \sum x_i w_i$$

Delta rule

$$b(new) = b(old) + \alpha(t - y\_in)$$
$$w_i(new) = w_i(old) + \alpha(t - y\_in)x_i$$

Activation Function

$$y = \begin{cases} 1 \text{ if } y\_in >= 0 \\ -1 \text{ if } y\_in < 0 \end{cases}$$

ADALINE (continued)

The Learning Rate, α

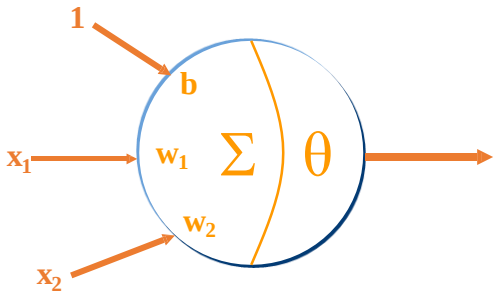
- The performance of an ADALINE neuron depends heavily on the choice of the learning rate
  - if it is too large the system will not converge
  - if it is too small the convergence will take to long

- Typically, α is selected by trial and error
  - typical range: $0.01 < \alpha < 10.0$
  - often start at 0.1
  - sometimes it is suggested that:
    $0.1 < n\alpha < 1.0$
    where n is the number of inputs

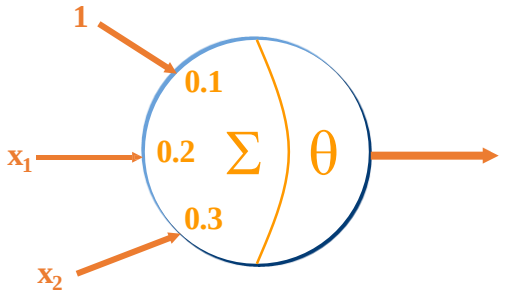
ADALINE (continued)

- Realization of AND function ,α = 0.1

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



Step 1:Initial Conditions: Set the weights to small random values:



ADALINE (continued)

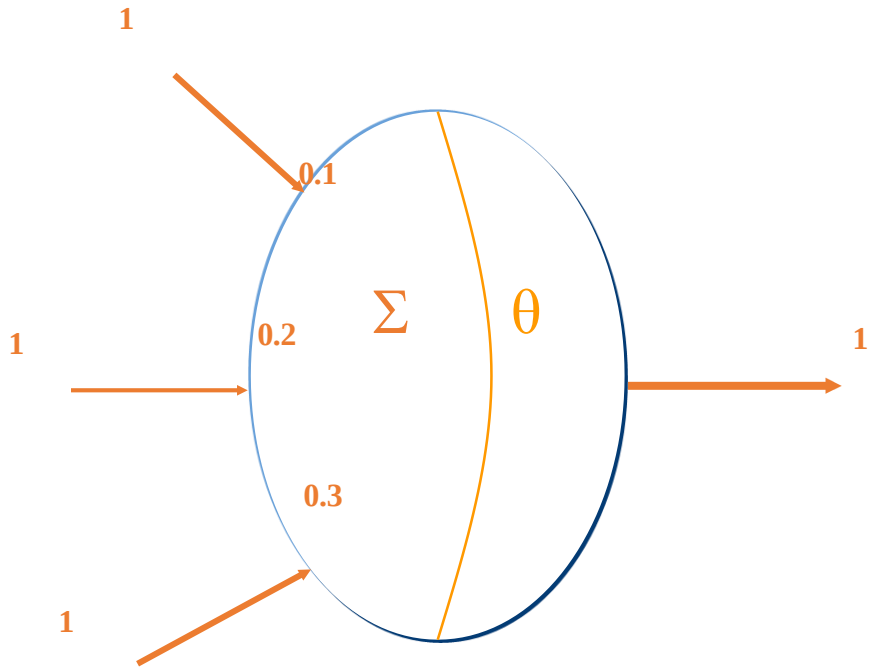
Input (1,1) Target: 1

The net input is: y_in = 0.1 + 0.2*1 + 0.3*1 = 0.6

The new weights are:

b = 0.1 + 0.1(1-0.6) = 0.14
w_1 = 0.2 + 0.1(1-0.6)1 = 0.24
w_2 = 0.3 + 0.1(1-0.6)1 = 0.34

The largest weight change is 0.04



ADALINE (continued)

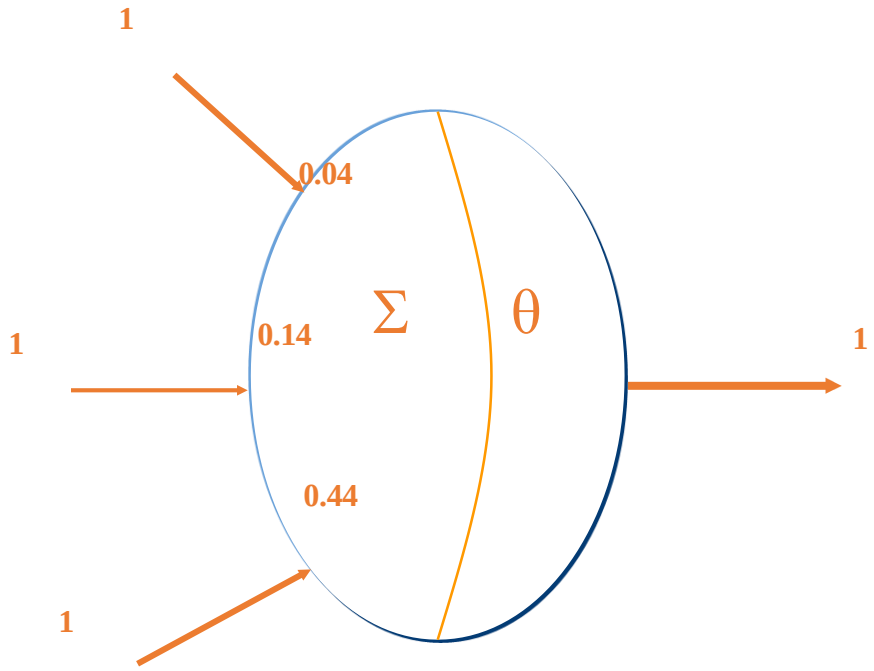
Input (1,-1) Target: -1

The net input is: y_in = 0.14 + 0.24*1 + 0.34*(-1) = 0.04

The new weights are:

b = 0.14 - 0.1(1+0.04) = 0.04
w_1 = 0.24 - 0.1(1+0.04)1 = 0.14
w_2 = 0.34 + 0.1(1+0.04)1 = 0.44

The largest weight change is 0.1



ADALINE (continued)

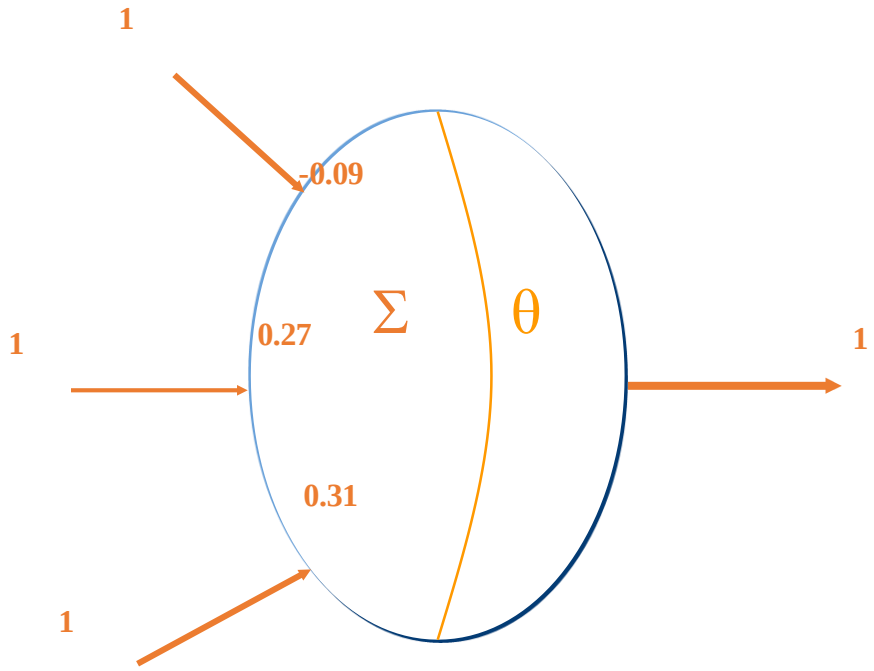
Input (-1,1) Target: -1

The net input is: y_in = 0.04 - 0.14*1 + 0.44*1 = 0.34

The new weights are:

b = 0.04 - 0.1(1+0.34) = -0.09
w_1 = 0.14 + 0.1(1+0.34)1 = 0.27
w_2 = 0.44 - 0.1(1+0.34)1 = 0.31

The largest weight change is 0.13



ADALINE (continued)

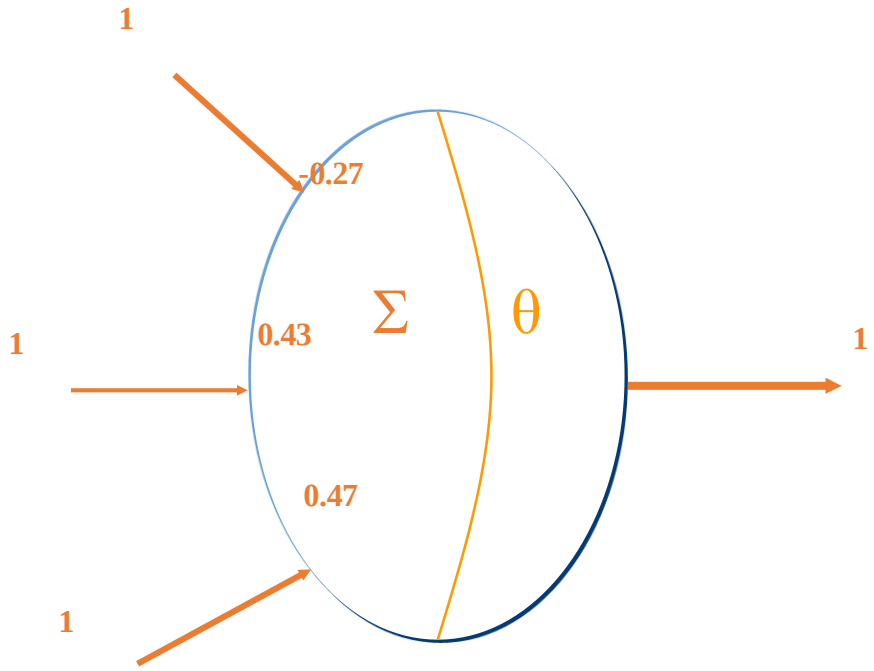
Input (-1,1) Target: -1

The net input is: y_in = -0.09 - 0.27*1 - 0.31*1 = -0.67

The new weights are:

b = -0.09 - 0.1(1+0.67) = -0.27
w_1 = 0.27 + 0.1(1+0.67)1 = 0.43
w_2 = 0.31 + 0.1(1+0.67)1 = 0.47

The largest weight change is 0.16



Carry out the same procedure the largest change in the weights over a complete cycle is less than some small number (say 0.01), the final weights are b = -0.5, w_1 = 0.5, w_2 = 0.5

Limitation of Perceptron, ADALINE

- If the problem is not linearly separable then the behavior of the algorithm is not guaranteed.
- If the problem is linearly separable, there may be a number of possible solutions.
- The algorithm as stated gives no indication of the quality of the solution found.

Multilayer Perceptron (Backpropagation)

- Backpropagation is the most well know and widely used neural network system
- It is a multi-layered, feedfoward, perceptron-like structure
- Uses the backpropagation rule (or generalized delta rule) for training

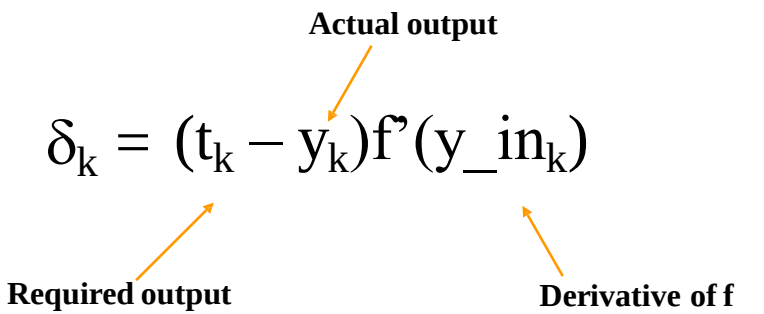
Multilayer Perceptron (Backpropagation) (continued)

A multi-layered perceptron has three distinctive characteristics

- The network contains one or more layers of hidden neurons
- The network exhibits a high degree of connectivity
- Each neuron has a smooth (differentiable everywhere) nonlinear activation function, the most common is the sigmoidal nonlinearity:

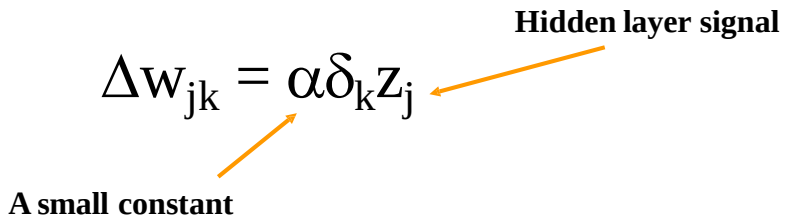
Multilayer Perceptron (Backpropagation) (continued)

- Step 0: Initialize the weights to small random values
- Step 1: Feed the training sample through the network and determine the final output
- Step 2: Compute the error for each output unit, for unit k it is:



Multilayer Perceptron (Backpropagation) (continued)

- Step 3: Calculate the weight correction term for each output unit, for unit k it is:



Step 4: Propagate the delta terms (errors) back through the weights of the hidden units where the delta input for the j^{th} hidden unit is:

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

The delta term for the j^{th} hidden unit is: $\delta_j = \delta\_in_j f'(z\_in_j)$

Multilayer Perceptron (Backpropagation) (continued)

- Step 5: Calculate the weight correction term for the hidden units:

$$\Delta w_{ij} = \alpha \delta_j x_i$$

Step 6: Update the weights:

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

- Step 7: Test for stopping (maximum cylces, small changes, etc)

Multilayer Perceptron (Backpropagation) (continued)

- There are a number of options in the design of a backprop system
  - Initial weights – best to set the initial weights (and all other free parameters) to random numbers inside a small range of values (say –0.5 to 0.5)
  - Number of cycles – tend to be quite large for backprop systems
  - Number of neurons in the hidden layer – as few as possible

Multilayer Perceptron (Backpropagation) (continued)

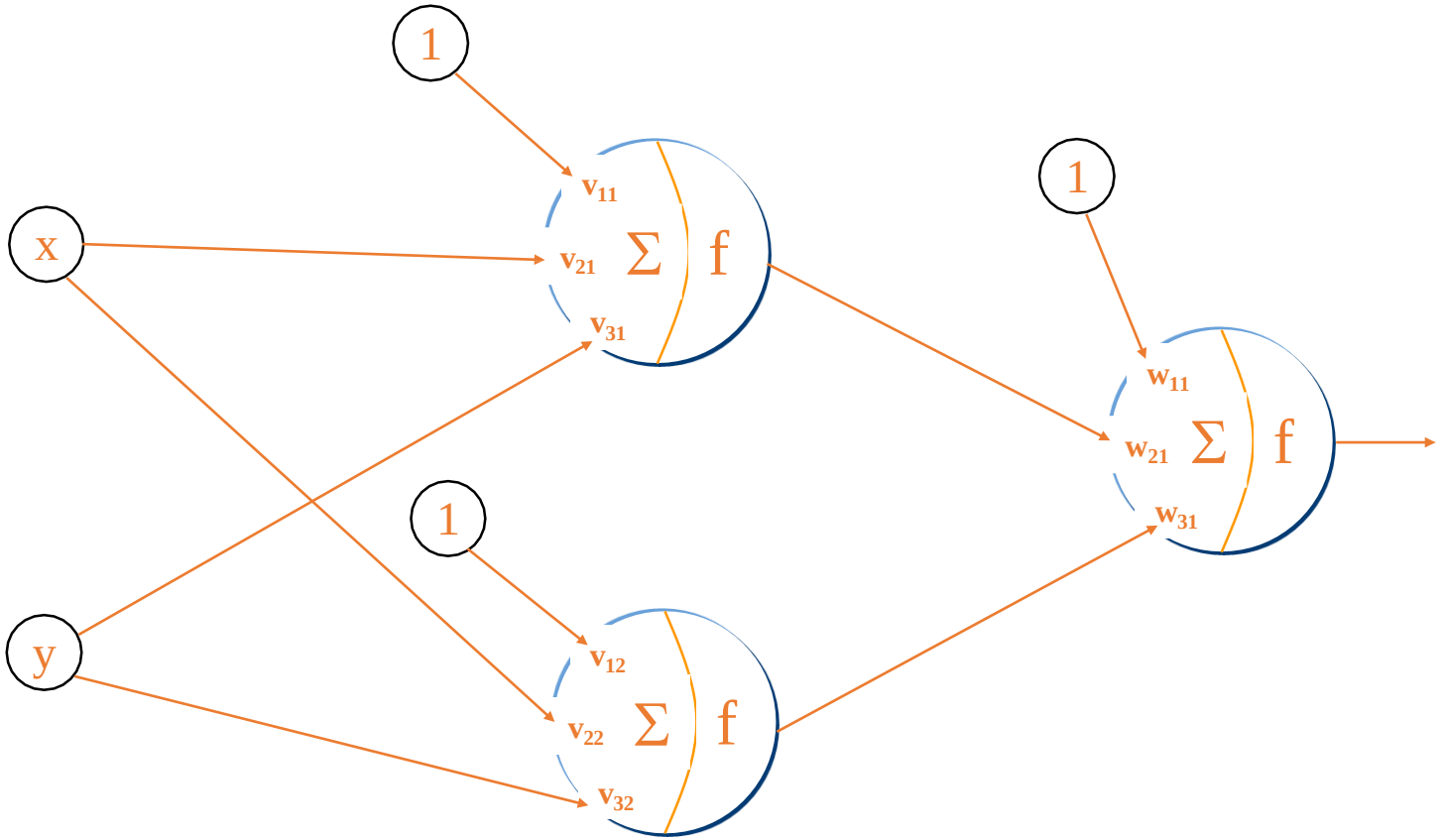
- The XOR function could not be solved by a single layer perceptron network

- The function is:

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

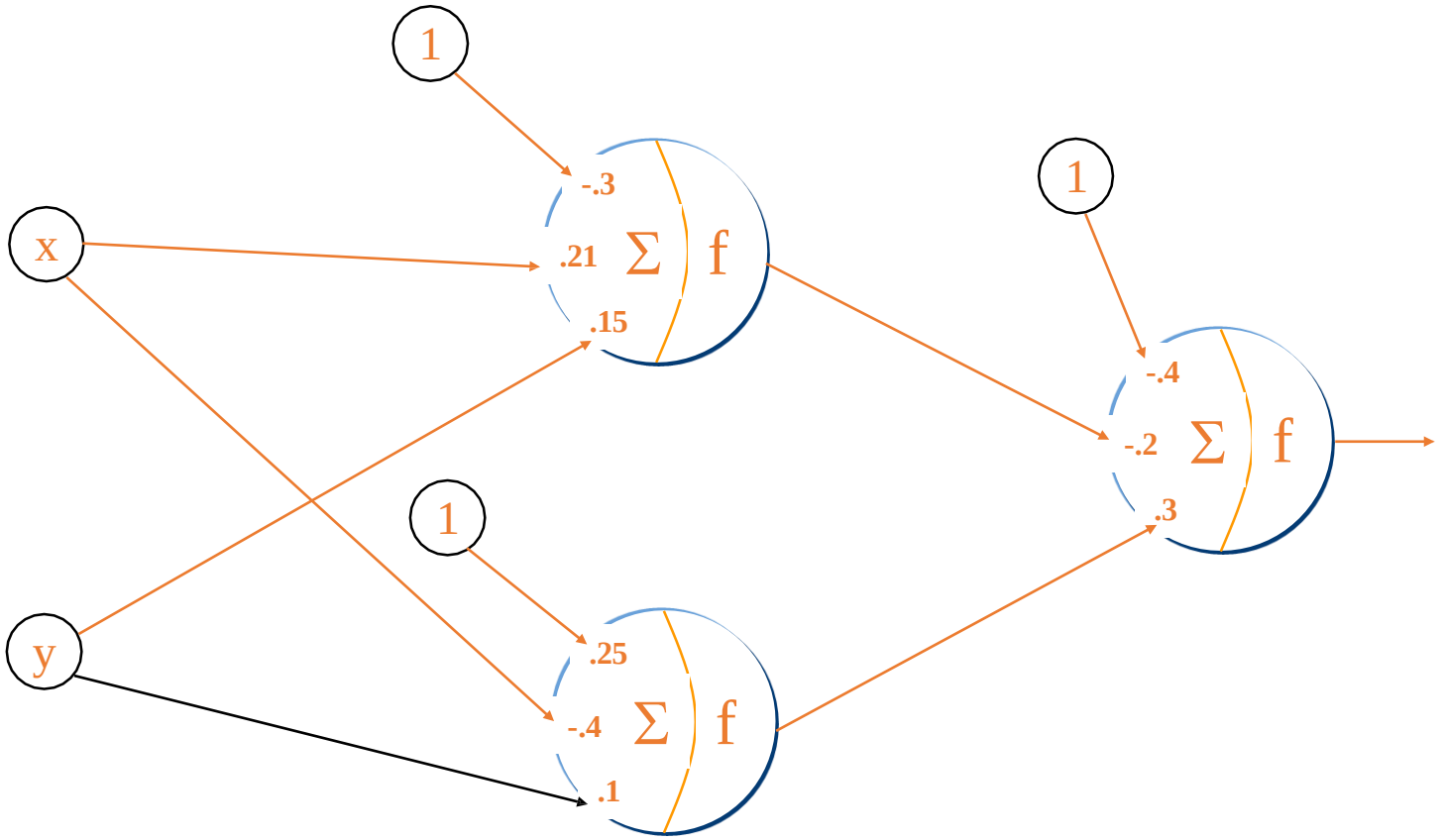
Multilayer Perceptron (Backpropagation) (continued)

XOR Architecture

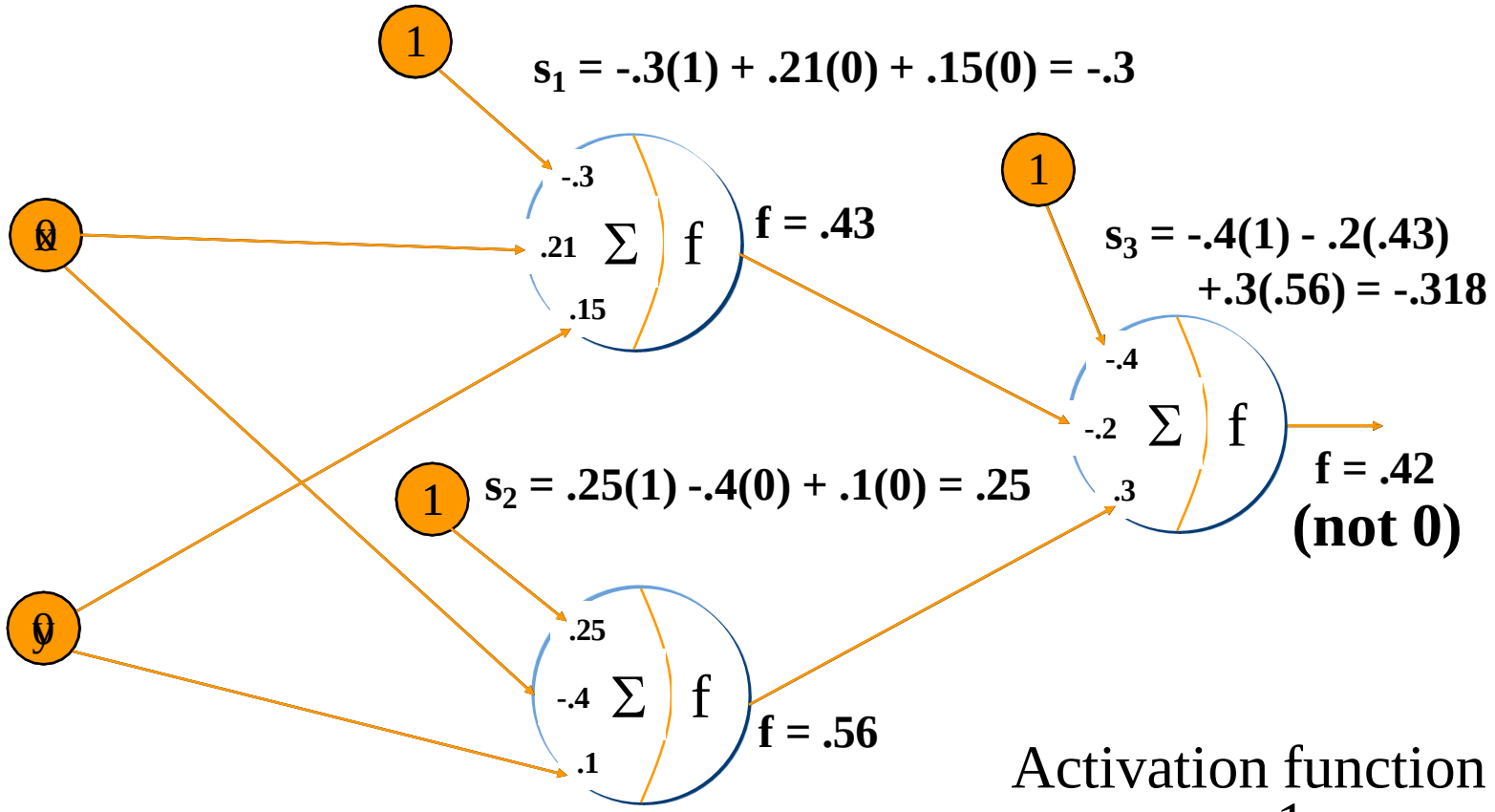


Multilayer Perceptron (Backpropagation) (continued)

- Randomly assign small weight values:



Multilayer Perceptron (Backpropagation) (continued)



Training Case: (0 0 0)

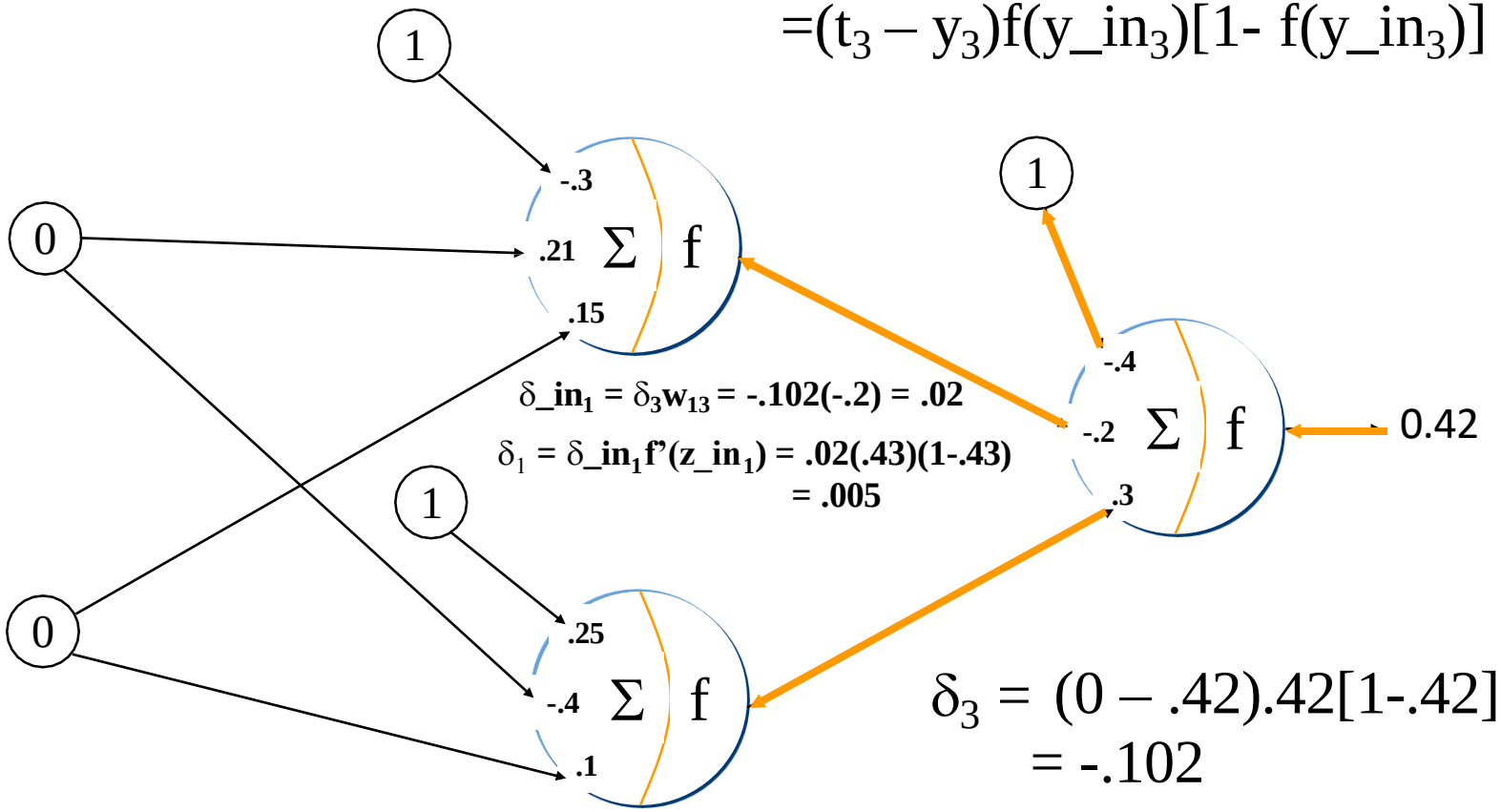
Activation function f:

$$y_j = \frac{1}{1 + e^{-s_j}}$$

Multilayer Perceptron (Backpropagation) (continued)

$$\delta_3 = (t_3 - y_3)f'(y\_in_3)$$

$$=(t_3 - y_3)f(y\_in_3)[1- f(y\_in_3)]$$



$\delta\_in_1 = \delta_3 w_{13} = -.102(-.2) = .02$

$\delta_1 = \delta\_in_1 f'(z\_in_1) = .02(.43)(1-.43)$
$= .005$

$\delta_3 = (0 - .42).42[1-.42]$
$= -.102$

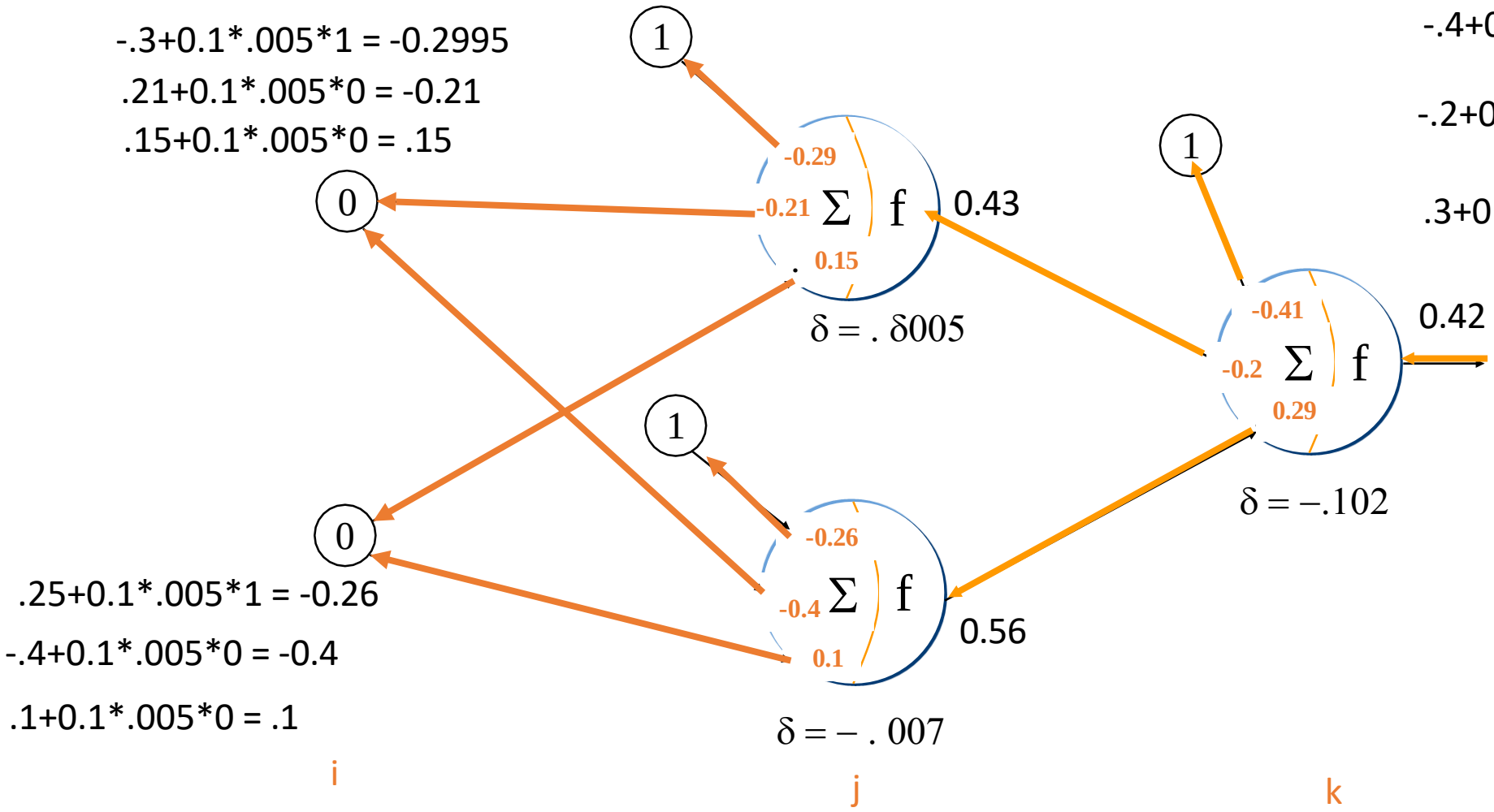
$\delta\_in_2 = \delta_3 w_{12} = -.102(.3) = -.03$

$\delta_2 = \delta\_in_2 f'(z\_in_2) = -.03(.56)(1-.56)$
$= -.007$

Multilayer Perceptron (Backpropagation) (continued)

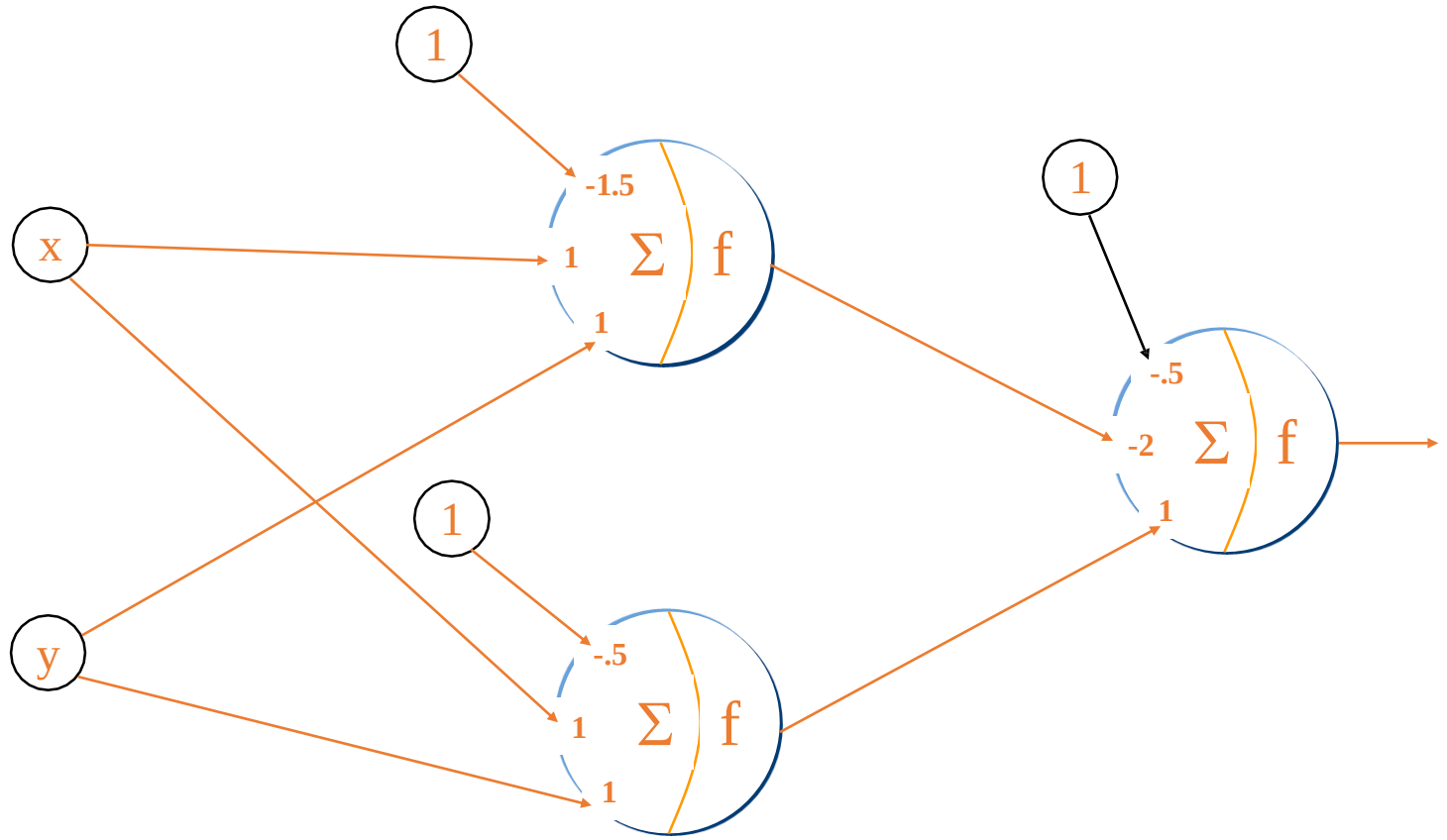
α=0.1, $w_{ij}(new) = w_{ij}(old) + \alpha\delta_j x_i$

α=0.1, $w_{jk}(new) = w_{jk}(old) + \alpha\delta_k y_j$

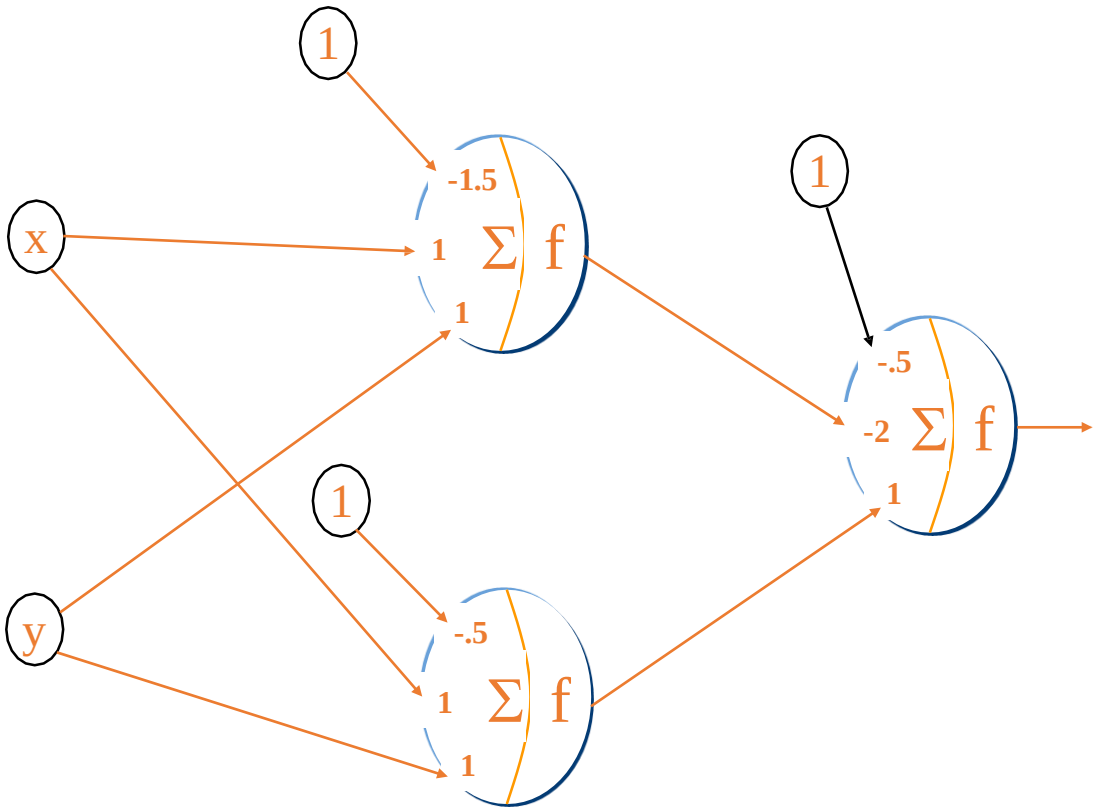


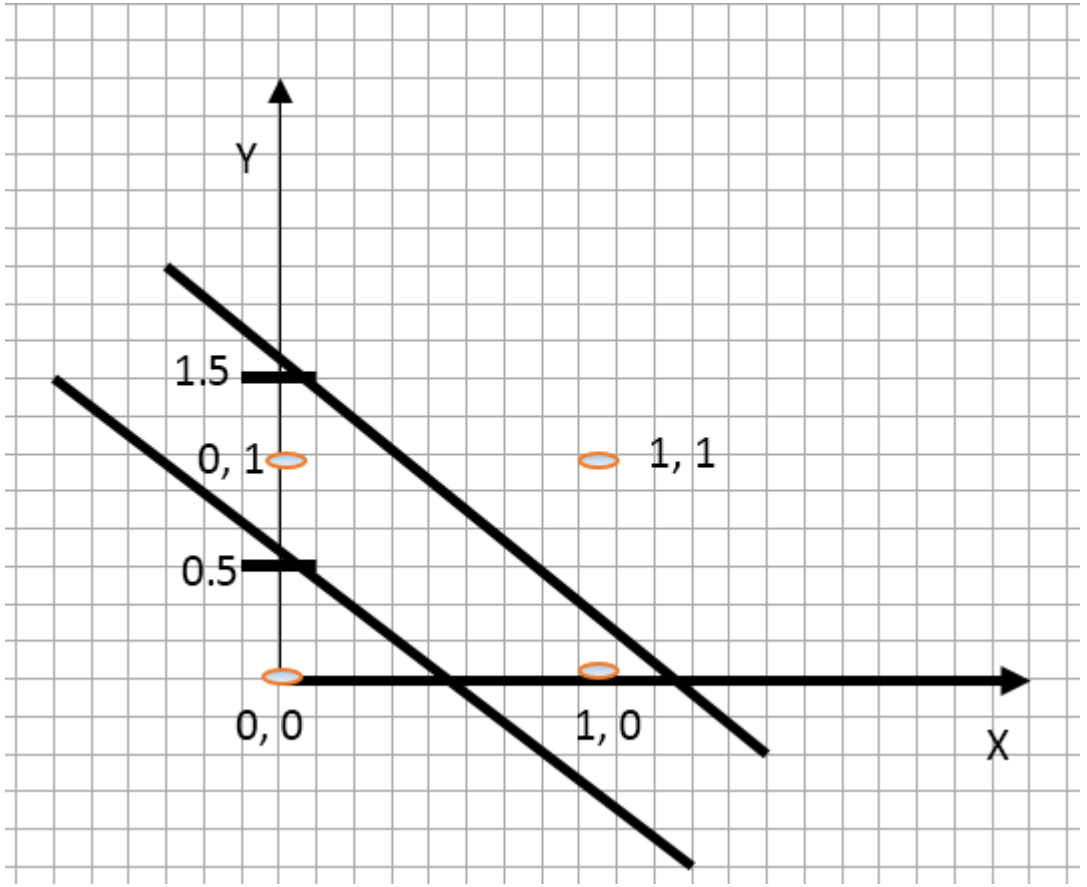
Multilayer Perceptron (Backpropagation) (continued)

- After the error is within tolerable limit i.e. 0.001 in this case, the final output is as follows:

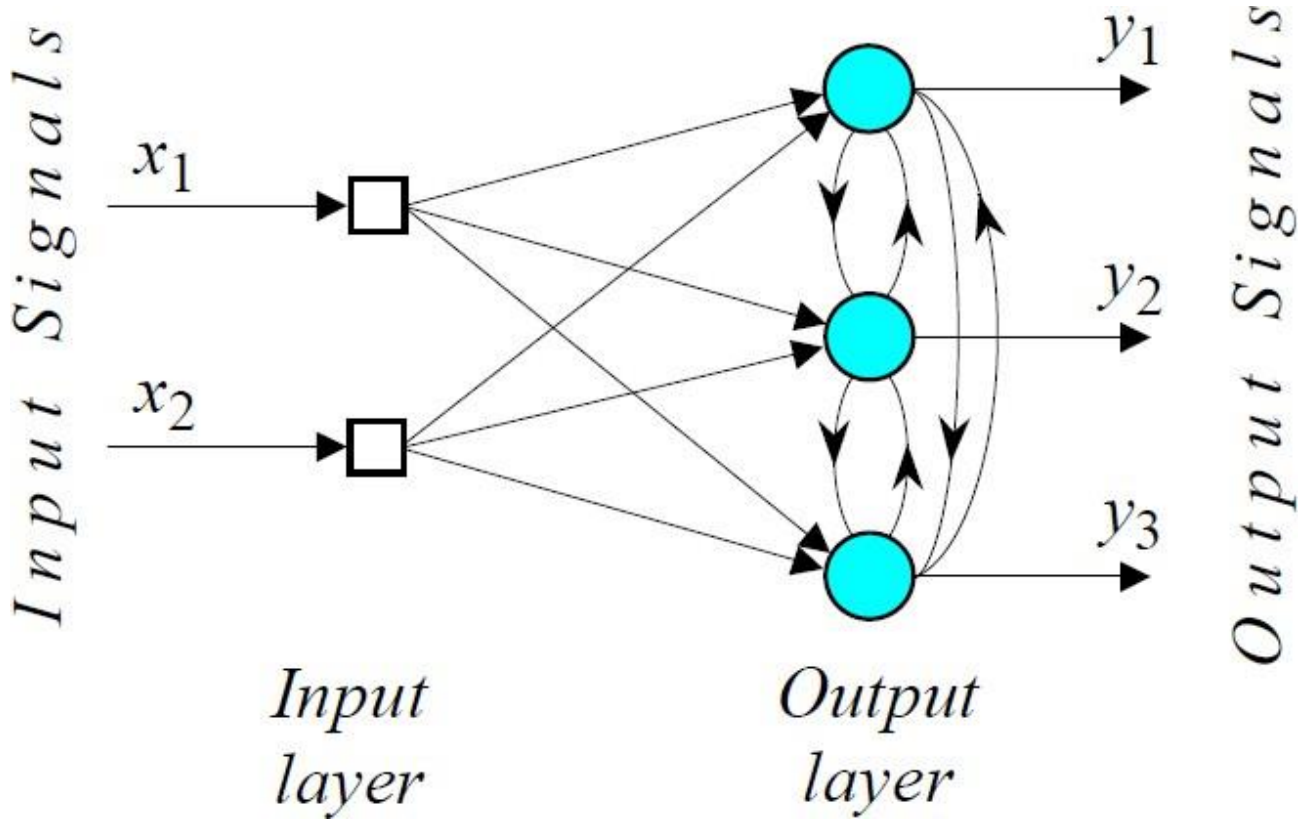


Multilayer Perceptron (Backpropagation) (continued)





Kohonen Network



Kohonen Network (continued)

Step 1: Find Euclidian Distance between input and weight vector

$$d = \left\| \mathbf{X} - \mathbf{W}_j \right\| = \left[\sum_{i=1}^{n} (x_i - w_{ij})^2 \right]^{1/2}$$

2. Determine the Minimum Euclidian Distance

$$j_{\mathbf{X}} = \min_j \left\| \mathbf{X} - \mathbf{W}_j \right\|, \qquad j = 1, 2, \ldots, m$$

3. Implement Competitive learning for the winning neuron i.e. having minimum distance

$$\Delta w_{ij} = \begin{cases} \alpha\,(x_i - w_{ij}), & \text{if neuron } j \text{ wins the competitio n} \\ 0, & \text{if neuron } j \text{ loses the competitio n} \end{cases}$$

Kohonen Network (continued)

$$\mathbf{X} = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix} \qquad \mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} \qquad \mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} \qquad \mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

$$d_1 = \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} = \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73$$

$$d_2 = \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} = \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59$$

$$d_3 = \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} = \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13$$

Kohonen Network (continued)

$$\Delta w_{13} = \alpha\,(x_1 - w_{13}) = 0.1\,(0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha\,(x_2 - w_{23}) = 0.1\,(0.12 - 0.21) = -0.01$$

$$\mathbf{W}_3(p+1) = \mathbf{W}_3(p) + \Delta\mathbf{W}_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

Definition of Artificial Neural Network

Definition

A structure (network) composed of a number of interconnected units (artificial neurons). Each unit has an input/output (I/O) characteristic and implements a local computation of function. The output of any unit is determined by its I/O Characteristic, its interconnection to other units, and (possibly) external inputs. The network usually develops an overall functionality through one or more forms of training.

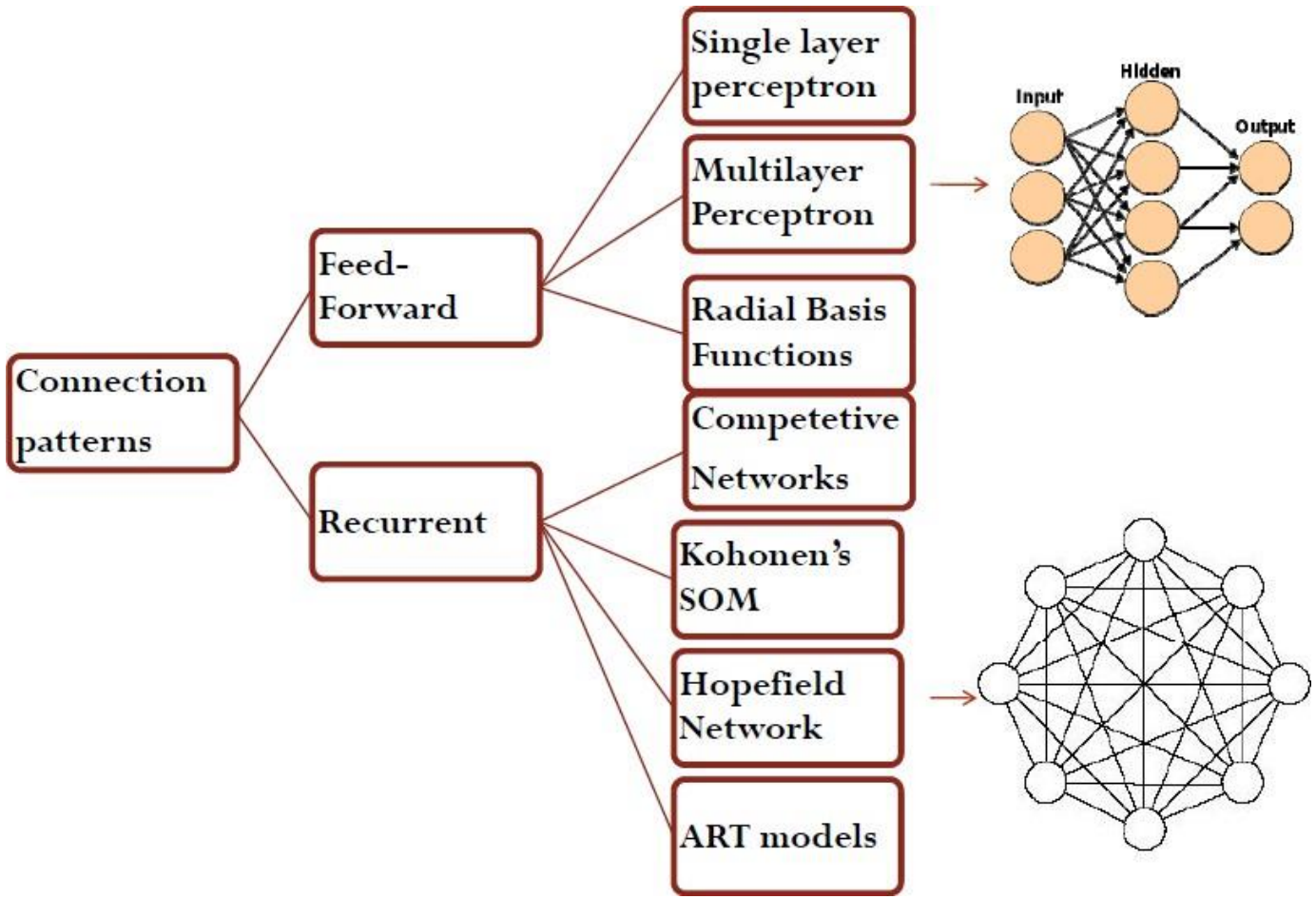
Advantages of ANN

- Advantages
  - Inherently massively parallel
  - May be fault tolerant
  - Adaptive
  - Extensive characterization of problem is not necessary

Disadvantages of ANN

- Disadvantages
  - No clear rules or design guidelines for arbitrary application
  - No general way to assess the internal operation of the network
  - Training may be difficult
  - Difficult to generalize

Architecture



Learning

- Supervised
  - Hebbian
  - Error correction
    - Perceptron
    - ADALINE
    - Muti layer perceptron
- Unsupervised
  - Hebbian
  - Competitive

Application of Neural Network

- Pattern Classification
- Clustering
- Function approximation
- Prediction
- optimization
- Content-addressable memory
- Control

Application of Neural Network

- Aerospace
- Digital signal processing
- Electronics
- Robotics
- Machine vision
- Speech
- Transportation
- Image processing and recognition
- Process identification

Procedures for ANN System Engineering

Step 1: study the domain and develop possible quantitative characterizations

Step 2: determine features

Step 3: consider constraints on desired system performance and computational resources

Step 4: consider the availability and quality of training and test data.

Procedures for ANN System Engineering

Step 5: consider the availability of suitable and known ANN system structures.

Step 6: develop an ANN simulation

Step 7: train the ANN system

Step 8: simulate ANN system performance using test set(s).

Step 9: Iterate among the preceding steps until the desired performance

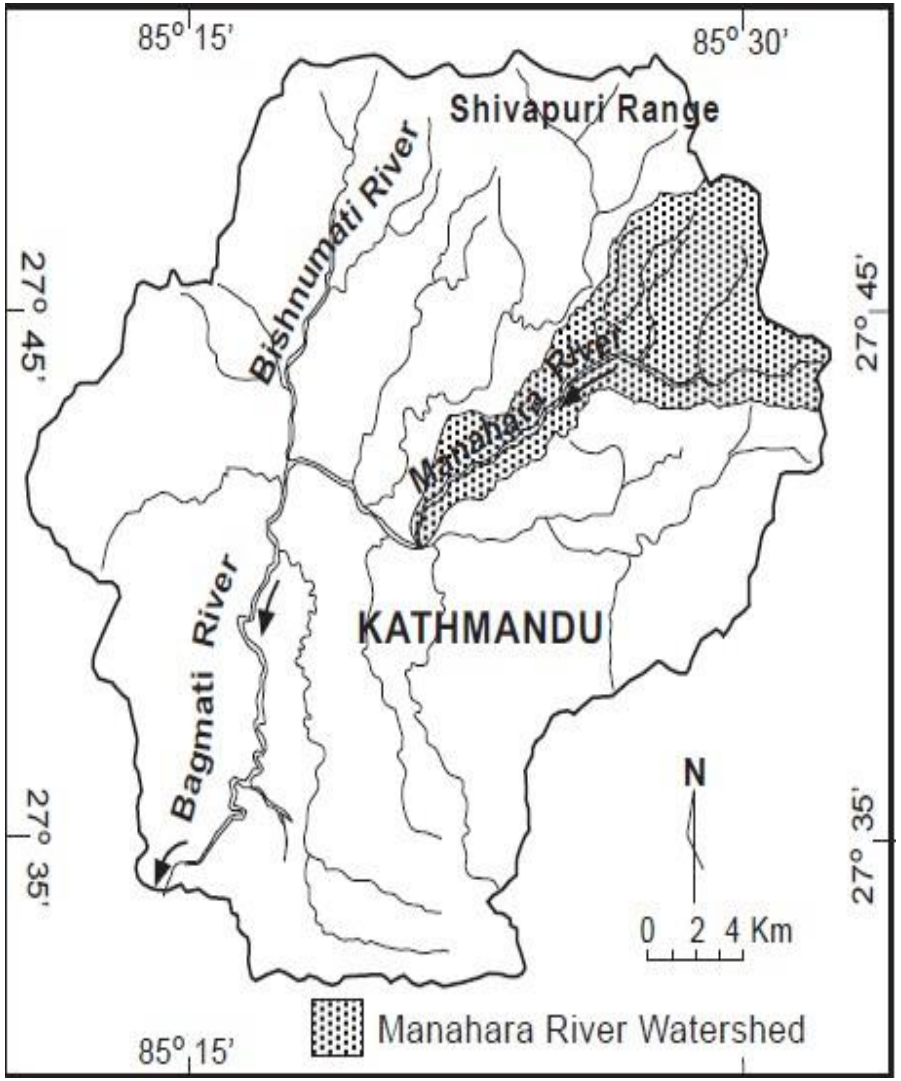
Example of Neural Network Implemetation

Forecasting Model of Manahara River flood levels: Assessing Agricultural impacts and Adaptive Strategies

Objectives

- To forecast the flood level in the Manahara river
- To assess the livelihood loss and agricultural impacts in regard to flood levels.
- To formulate adaptive strategies that can be used by local levels.

Study Area



Source: Bajracharya 2007

Characteristics

Catchment Area: 249.61 Sq.km

Average Slope: 3.3 m per Km

Catchment Length: 24.6 Km

Climate

Characteristics

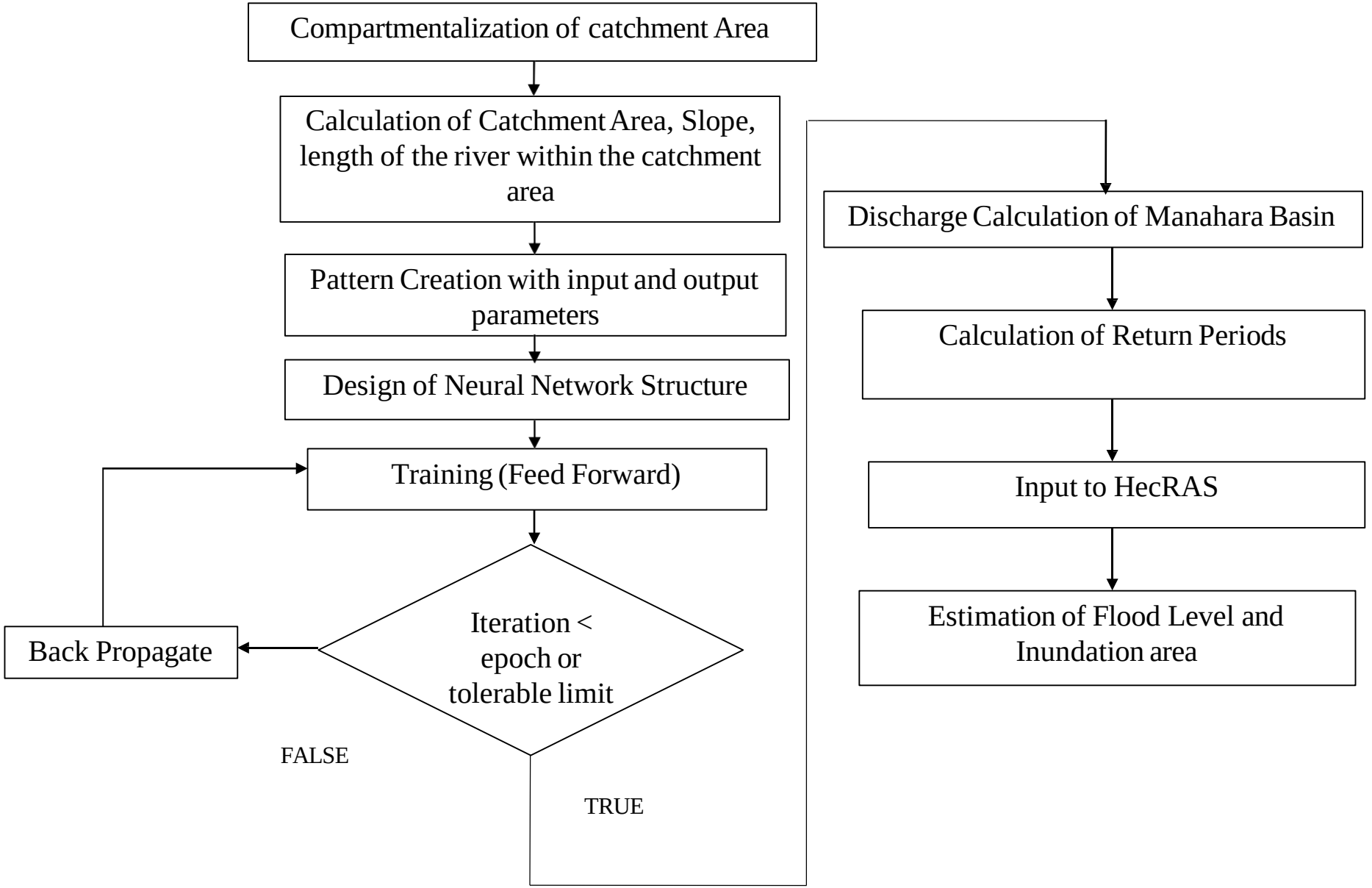
- Straight and narrow in upstream stretch and is meandered and wide in downstream stretch. (Bajracharya 2006)

- Cultivation and settlement areas have been increasing against forest area. (Shrestha 2007).

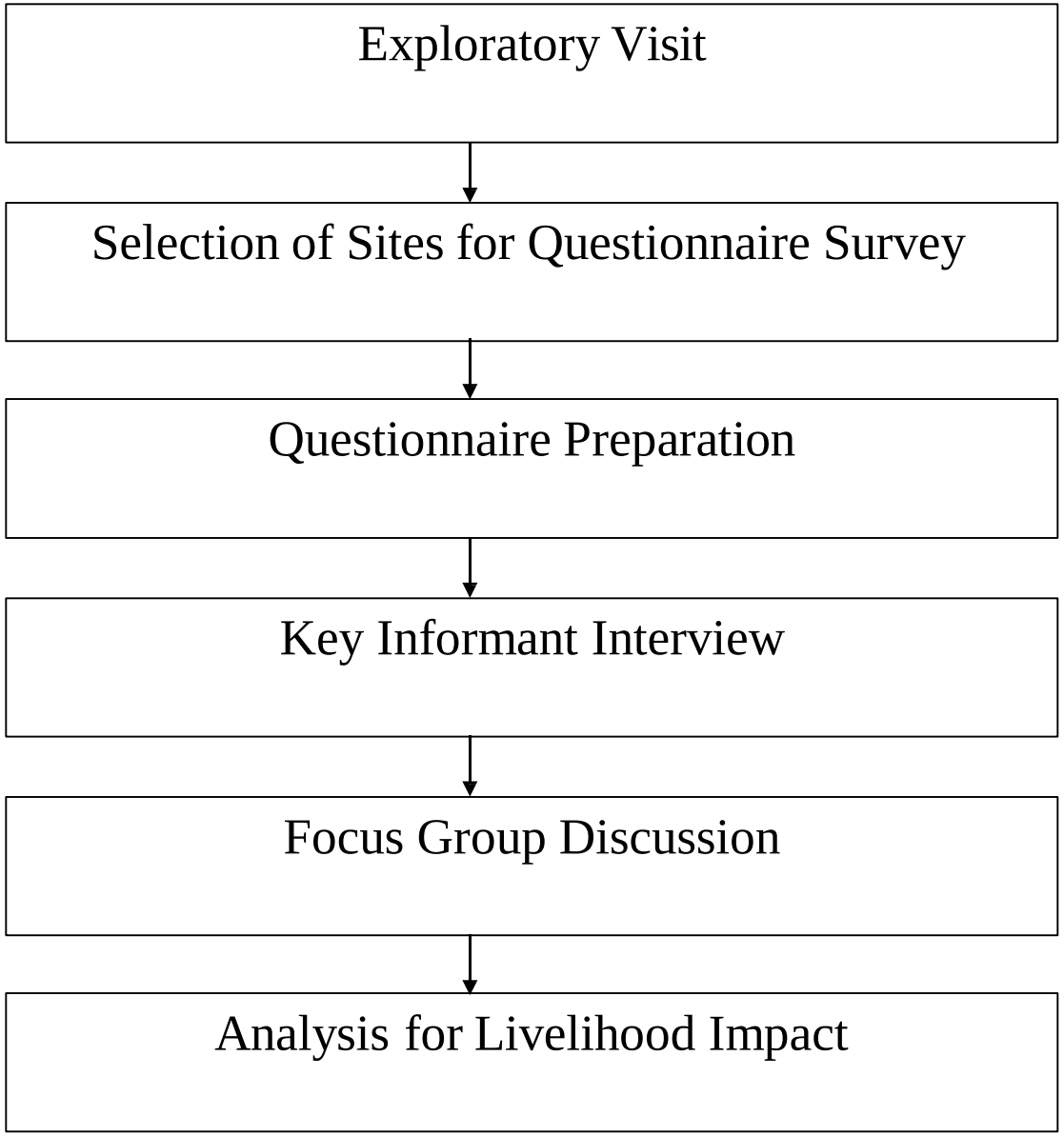
Characteristics

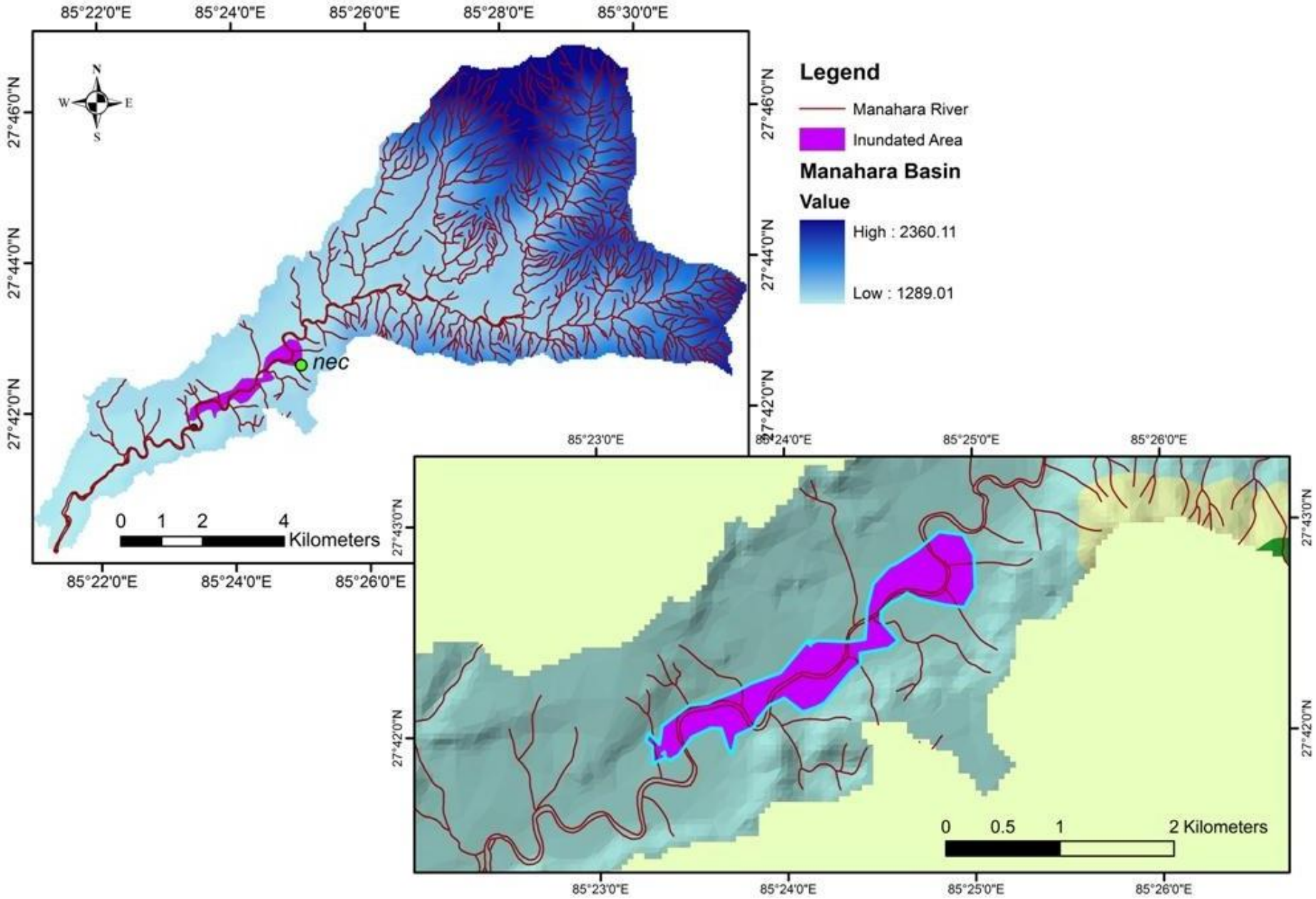
S.N	Stretch	Slope	Channel Material	Flood Intensity
1.	Upper (Kurthali)	0.011 m/m	Cobble, Sand, Pebble	Negligible
2.	Middle (Mulpani)	0.0056 m/m	Sandy Pebble	Moderate
3.	Lowest (Sano Thimi)	0.0235 m/m	Pebbles, Sand with Silt	High

Source: Bajracharya, 2007


Compartmentalization of catchment Area
Calculation of Catchment Area, Slope, length of the river within the catchment area
Pattern Creation with input and output parameters
Design of Neural Network Structure
Training (Feed Forward)
Iteration < epoch or tolerable limit
Back Propagate
FALSE
TRUE
Discharge Calculation of Manahara Basin
Calculation of Return Periods
Input to HecRAS
Estimation of Flood Level and Inundation area

Methodology Continued




85°22'0"E
85°24'0"E
85°26'0"E
85°28'0"E
85°30'0"E
27°46'0"N
27°44'0"N
27°42'0"N
N
W
E
S
nec
0 1 2 4
Kilometers
Legend
Manahara River
Inundated Area
Manahara Basin
Value
High : 2360.11
Low : 1289.01
85°23'0"E
85°25'0"E
27°43'0"N
0 0.5 1
2 Kilometers

Challenges

- Ramification of hidden layers and hidden nodes
- Learning rate – alpha, momentum ….
- Activation function - threshold, sigmoid, Relu, softmax …
- Best algorithm – very hard to ascertain as it depends on lots of constraints and the data availability

Thank you