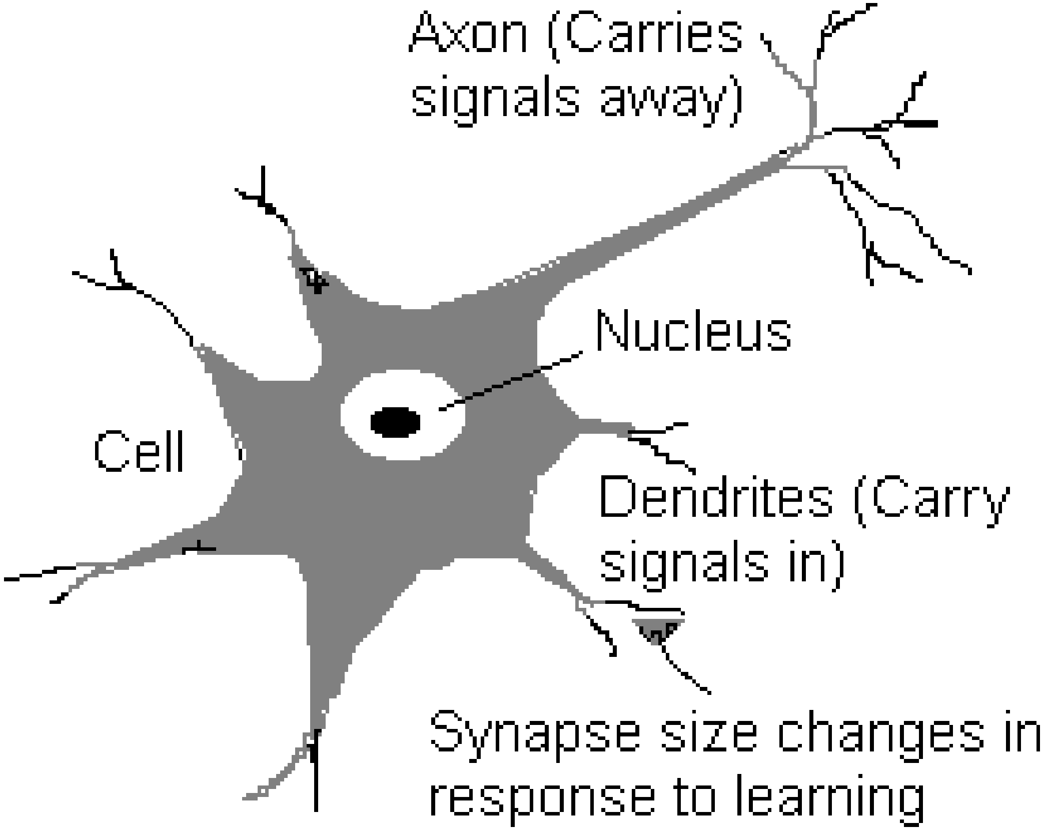
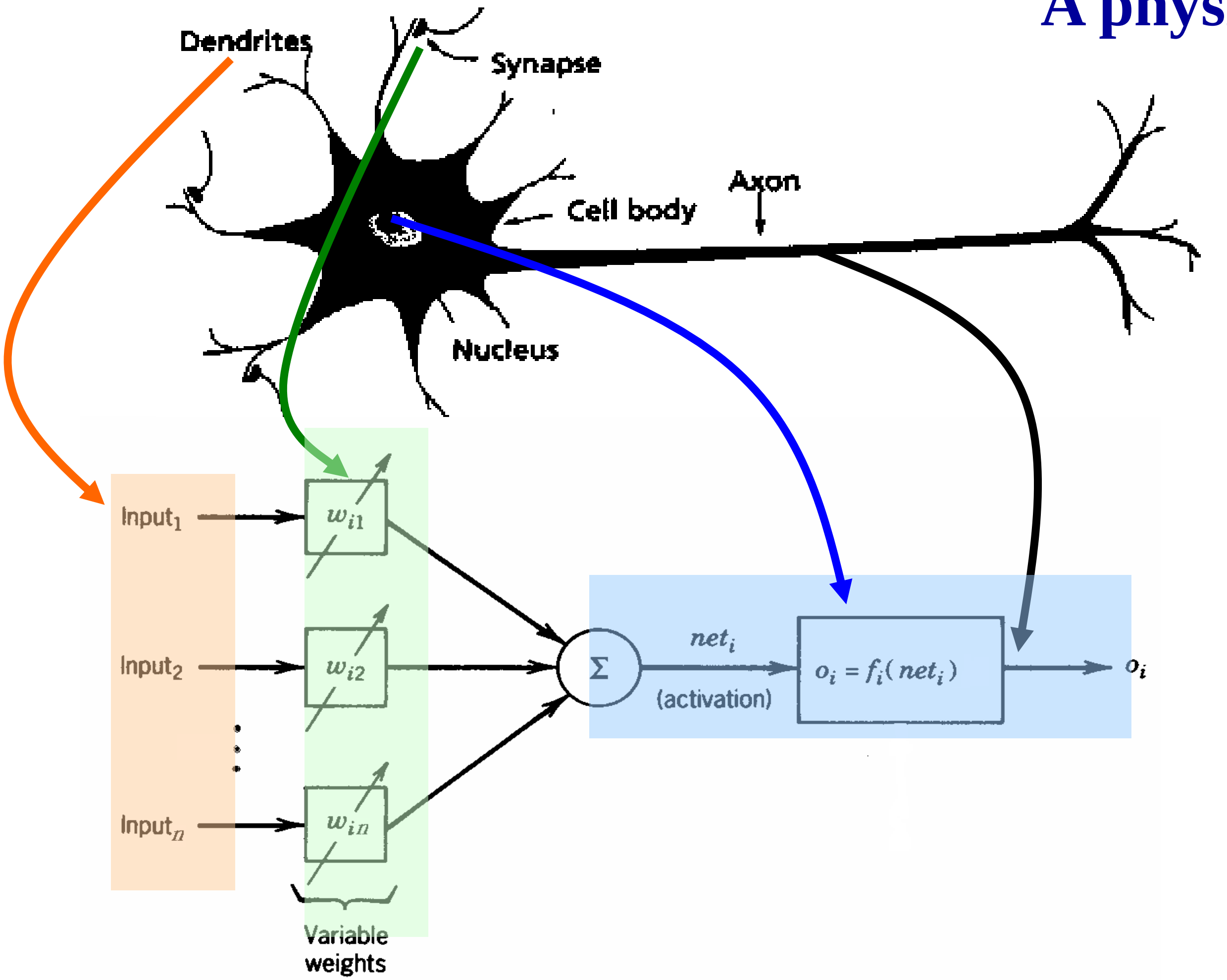
Chapter 3
Neurons, Neural Networks, and Linear Discriminants

Biological Neurons

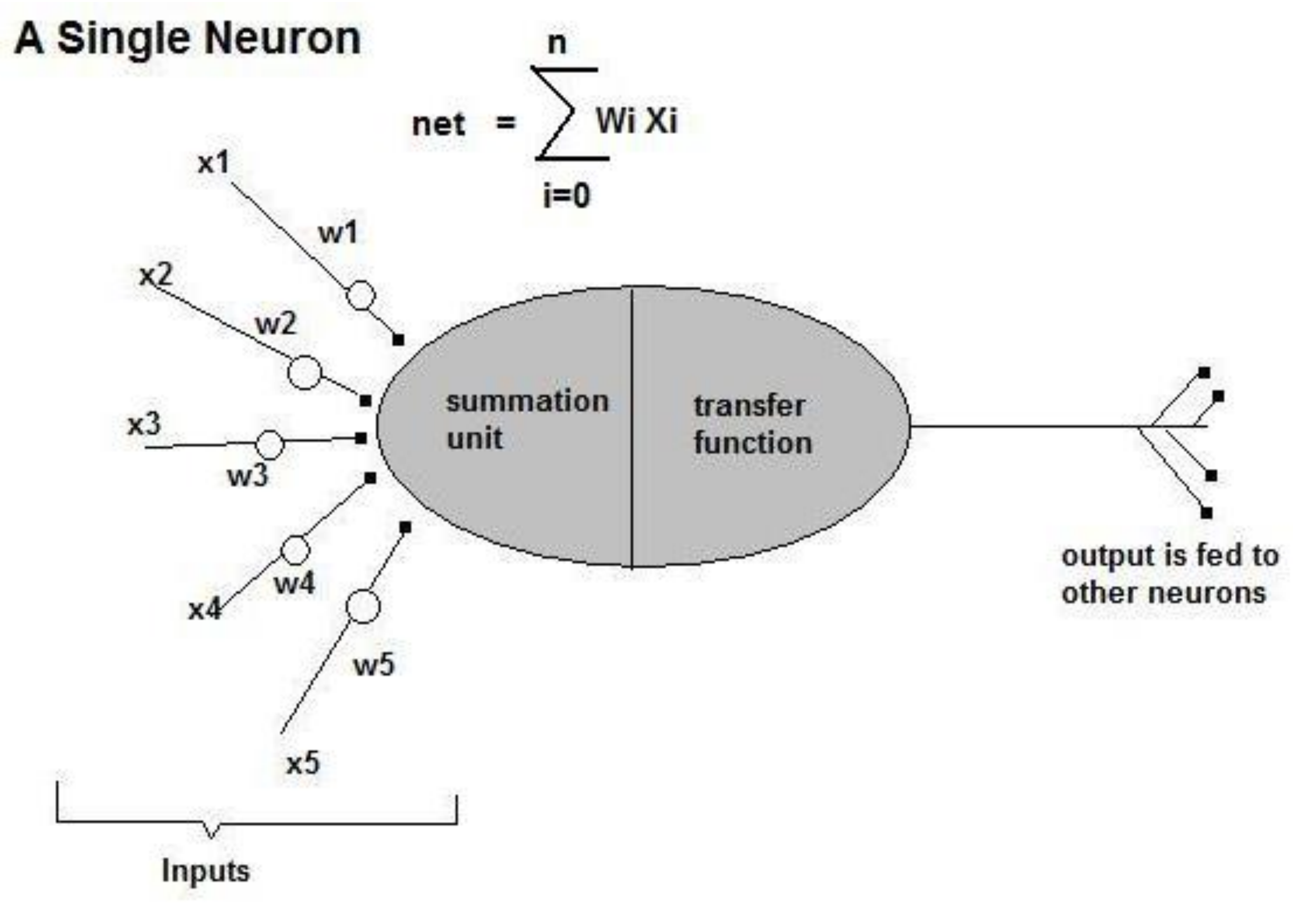

Axon (Carries signals away)
Nucleus
Cell
Dendrites (Carry signals in)
Synapse size changes in response to learning

Artificial Neurons

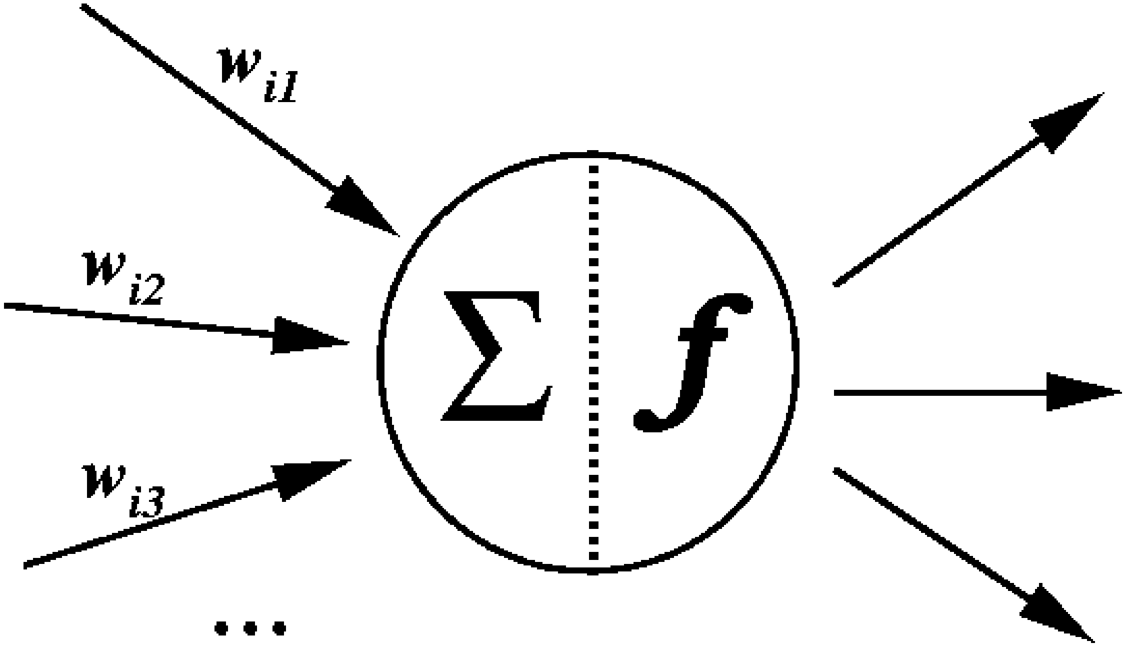
A physical neuron



ARTIFICIAL NEURON MODEL



ARTIFICIAL NEURON MODEL



$$y_i = f(net_i)$$

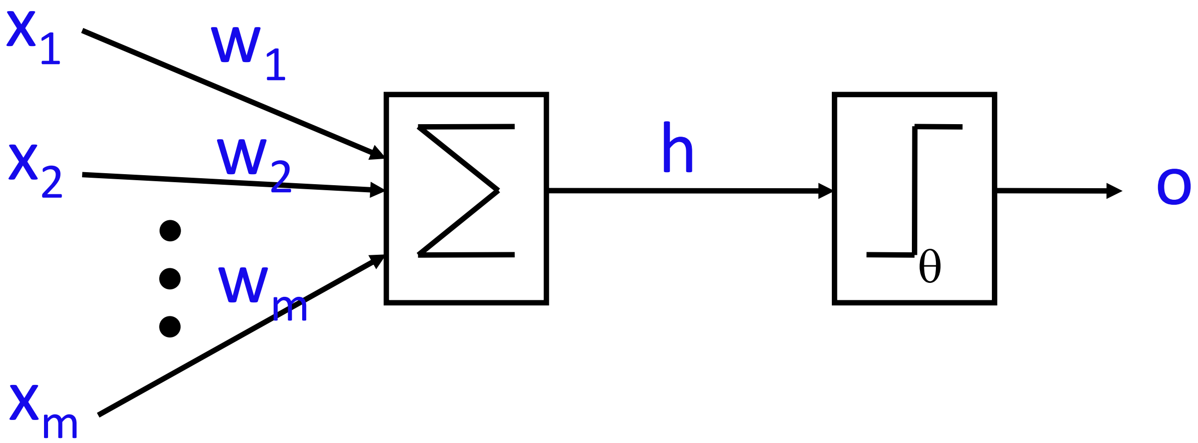
Basic elements of neural networks are:

- Input nodes
- Weights
- Activation function
- Total signal reaching at output are given by:

$$\text{Output (y)} = \sum_{i=1}^{n} Wi\, Xi$$

- Output may be more than one it depends on number of neuron i.e. multiple neuron will cause multiple output.

McCulloch and Pitts Neurons



- Greatly simplified biological neurons
- Sum the inputs
  - If total is less than some threshold, neuron fires
  - Otherwise does not

McCulloch and Pitts Neurons

$$h = \sum_{i=1}^{m} x_i w_i \qquad o = \begin{cases} 1 & h \geq \theta \\ 0 & h < \theta \end{cases}$$

for some threshold θ

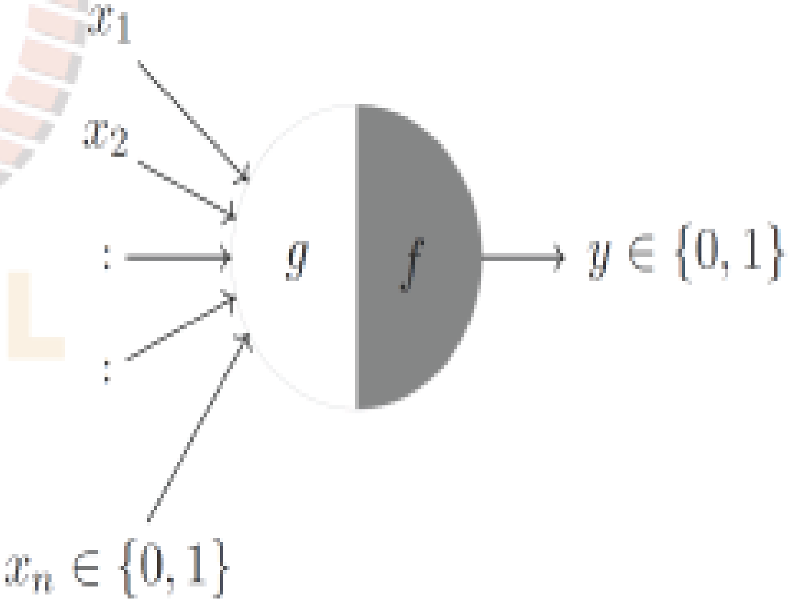
- The weight w_j can be positive or negative
  - Inhibitory or exitatory
- Use only a linear sum of inputs
- Use a simple output instead of a pulse (spike train)

McCulloch-Pitts Neuron

- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)
- g aggregates the inputs and the function f takes a decision based on this aggregation
- The inputs can be excitatory or inhibitory
- $y = 0$ if any x_i is inhibitory, else

$$g(x_1, x_2, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

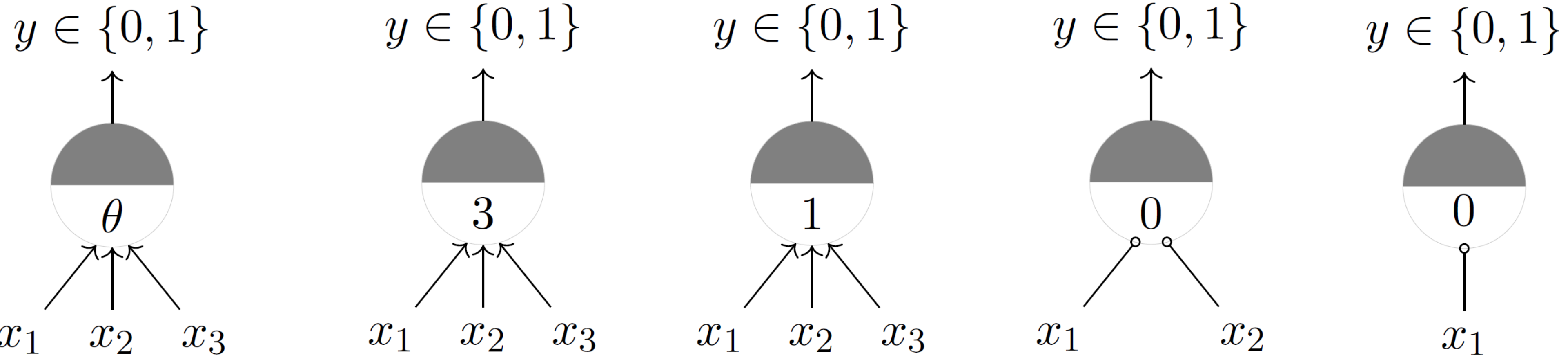
$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \ \textit{if} \ g(\mathbf{x}) \geq \theta \\ &= 0 \ \textit{if} \ g(\mathbf{x}) < \theta \end{aligned}$$



- θ is a thresholding parameter

NPTEL

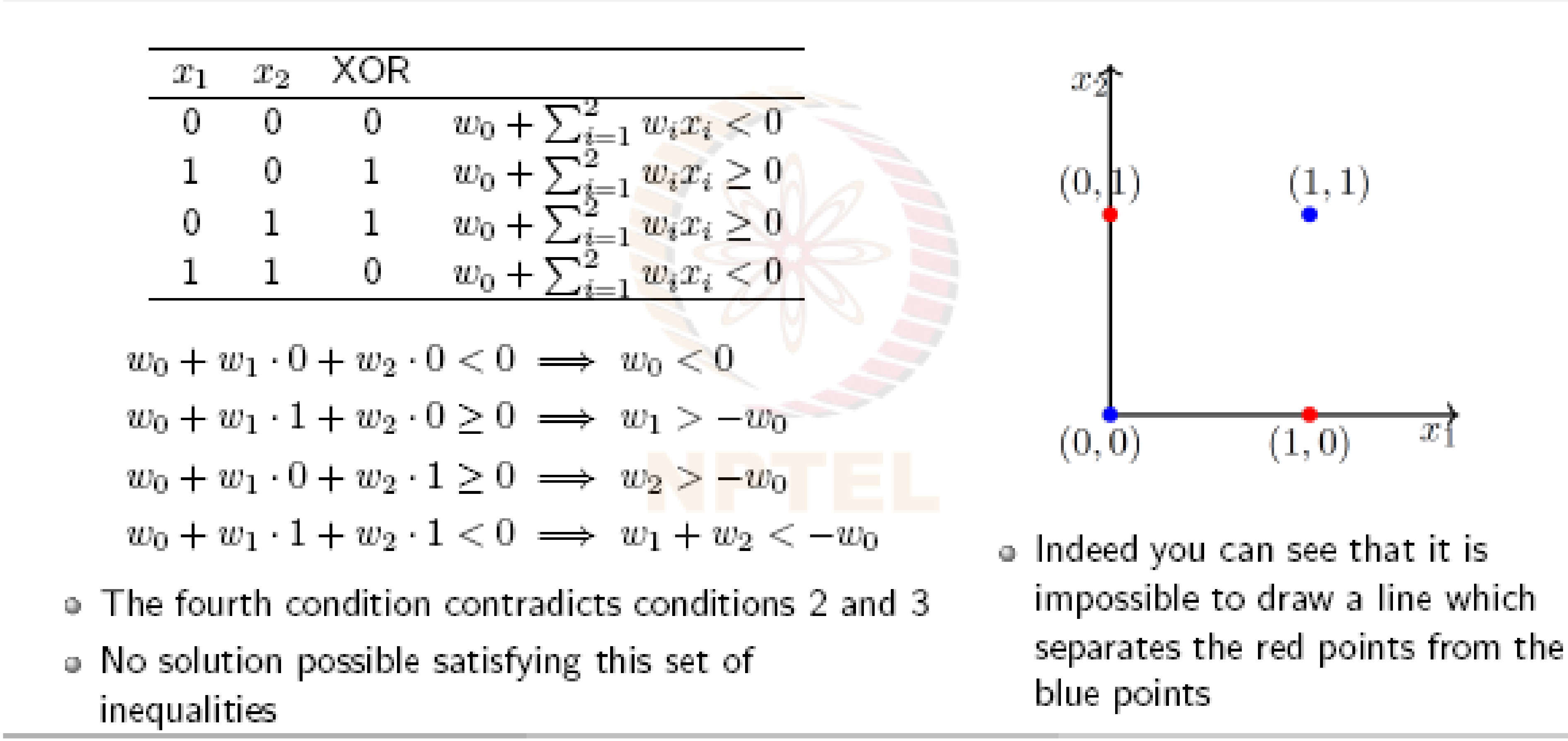
McCulloch-Pitts Neuron



A McCulloch-Pitts Unit AND function OR function NOR function NOT function

- Feedforward MP networks can compute any Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$
- Recursive MP networks can simulate any Deterministic Finite Automaton (DFA) (S

The XOR Problem



x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \implies w_1 + w_2 < -w_0$$

- The fourth condition contradicts conditions 2 and 3
- No solution possible satisfying this set of inequalities

- Indeed you can see that it is impossible to draw a line which separates the red points from the blue points

Neural **Networks**

- Can put lots of McCulloch & Pitts neurons together
- Connect them up in any way we like
- In fact, assemblies of the neurons are capable of *universal computation*
    - Can perform any computation that a normal computer can
    - Just have to solve for all the weights w_{ij}

Training Neurons

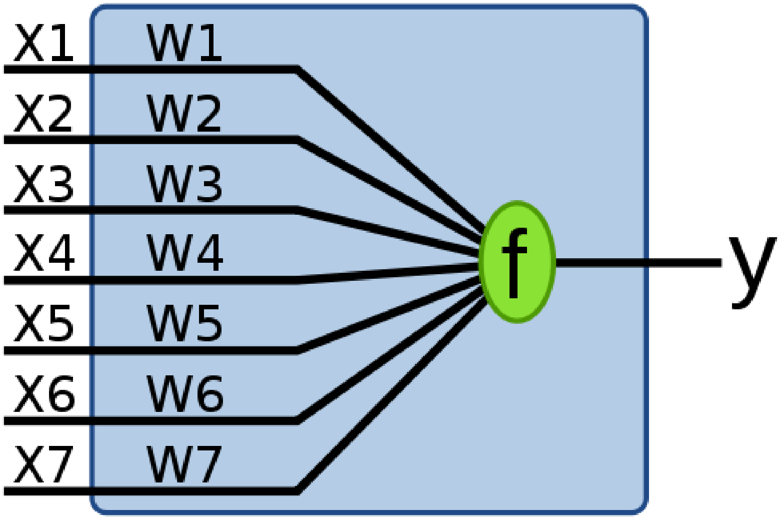
- Adapting the weights is learning
  - How does the network know it is right?
  - How do we adapt the weights to make the network right more often?
- Training set with target outputs
- Learning rule

2.2 The Perceptron

The perceptron is considered the simplest kind of feed-forward neural network.

Definition from Wikipedia:
The **perceptron** is a binary classifier which maps its input *x* (a real-valued vector) to an output value *f*(*x*) (a single binary value) across the matrix:

$$f(x) = \begin{cases} 1 & wx - b > 0 \\ 0 & else \end{cases}$$



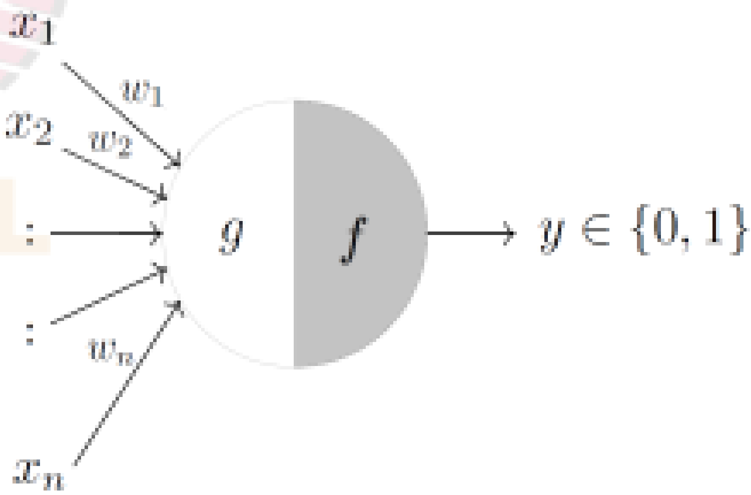
In order to not explicitly write b, we extend the input vector x by one more dimension that is always set to -1, e.g., **x=(-1,x_1, …, x_7)** with **x_0=-1**, and extend the weight vector to **w=(w_0,w_1, …, w_7)**. Then adjusting w_0 corresponds to adjusting b.

Perceptrons

- Frank Rosenblatt, an American psychologist, proposed the perceptron model (1958)
- Later refined and carefully analyzed by Minsky and Papert (1969)
- A more general computational model than McCulloch-Pitts neurons
- Inputs are no longer limited to boolean values

$$g(x_1, x_2, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} w_i * x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \; if \; g(\mathbf{x}) \geq \theta \\ &= 0 \; if \; g(\mathbf{x}) < \theta \end{aligned}$$



NPTEL

Perceptrons

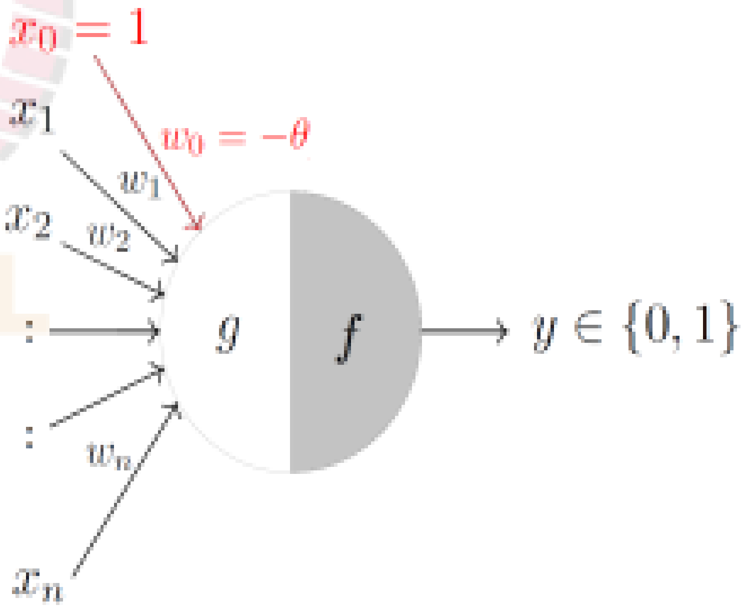
- Frank Rosenblatt, an American psychologist, proposed the perceptron model (1958)
- Later refined and carefully analyzed by Minsky and Papert (1969)
- A more general computational model than McCulloch-Pitts neurons
- Inputs are no longer limited to boolean values

- A more accepted convention

$$g(x_1, x_2, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=0}^{n} w_i * x_i$$

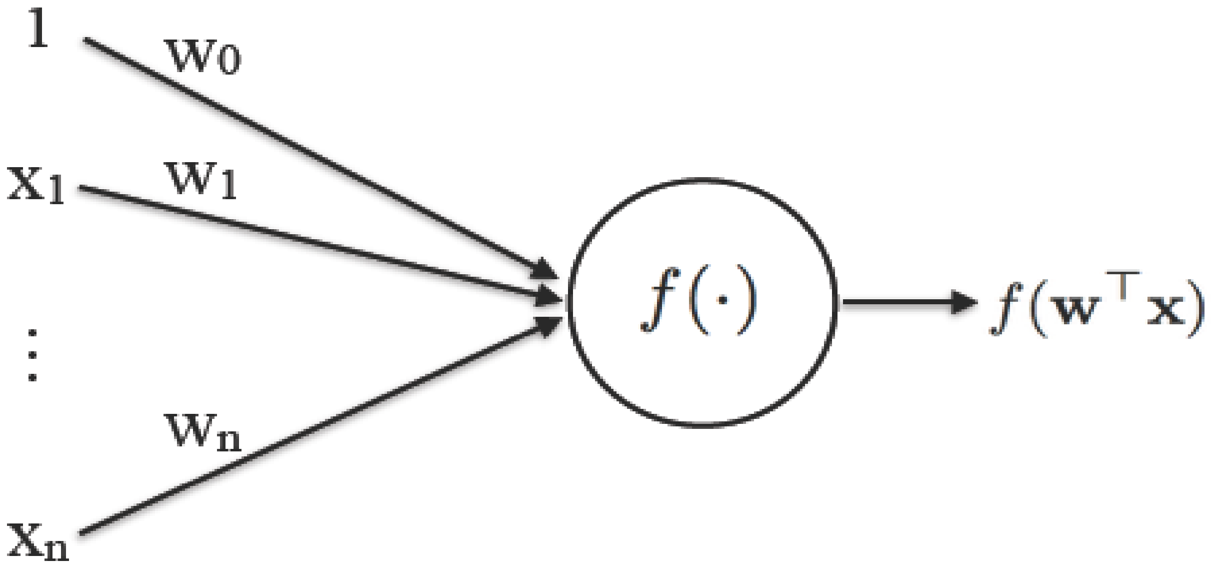
$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \; if \; g(\mathbf{x}) \geq 0 \\ &= 0 \; if \; g(\mathbf{x}) < 0 \end{aligned}$$

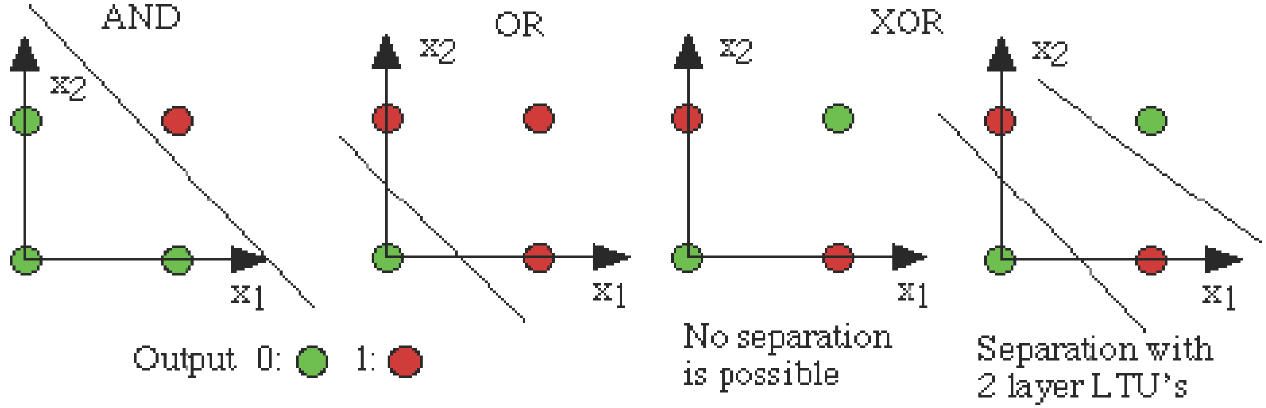
$$where \; x_0 = 1 \; and \; w_0 = -\theta$$



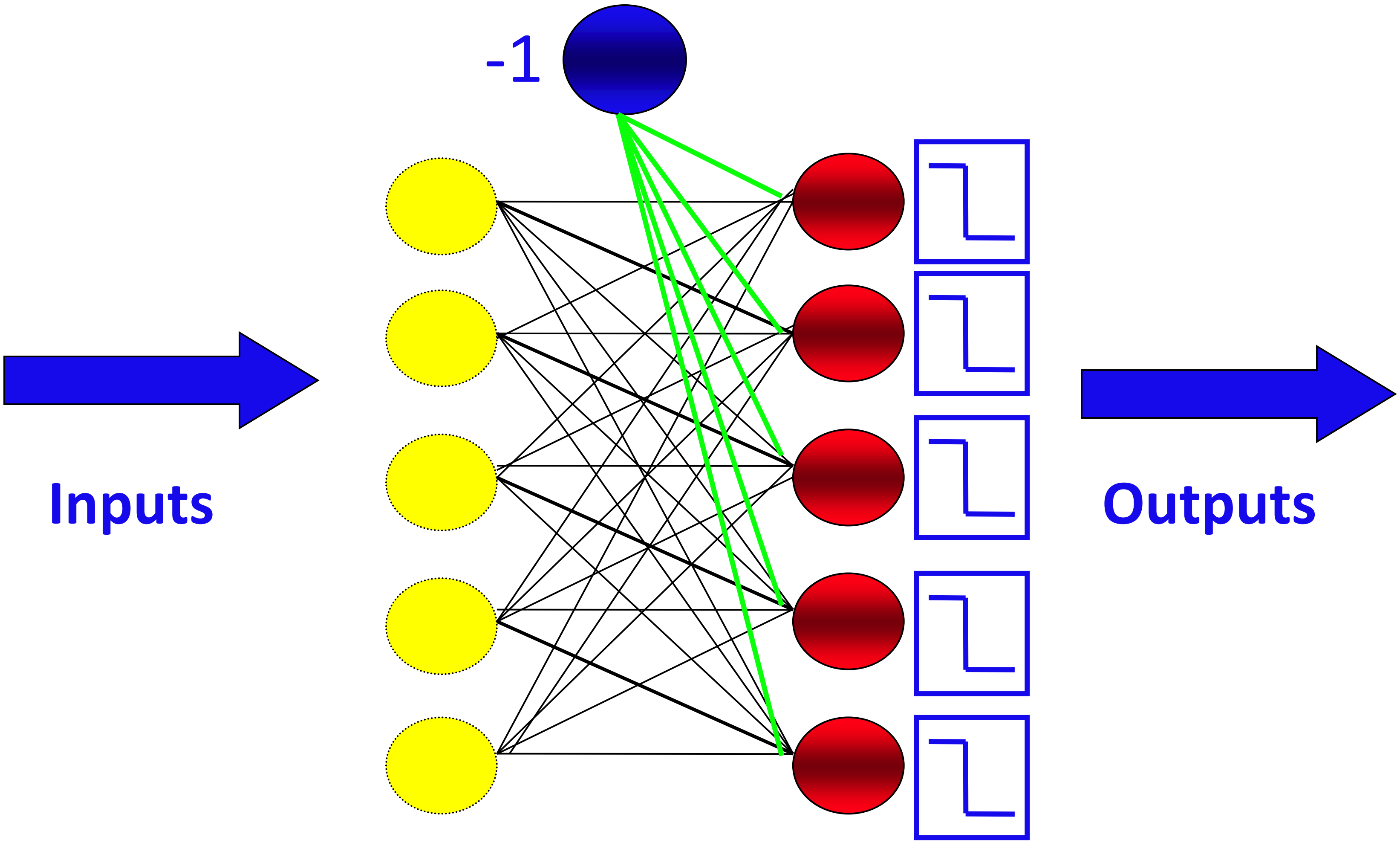
NPTEL

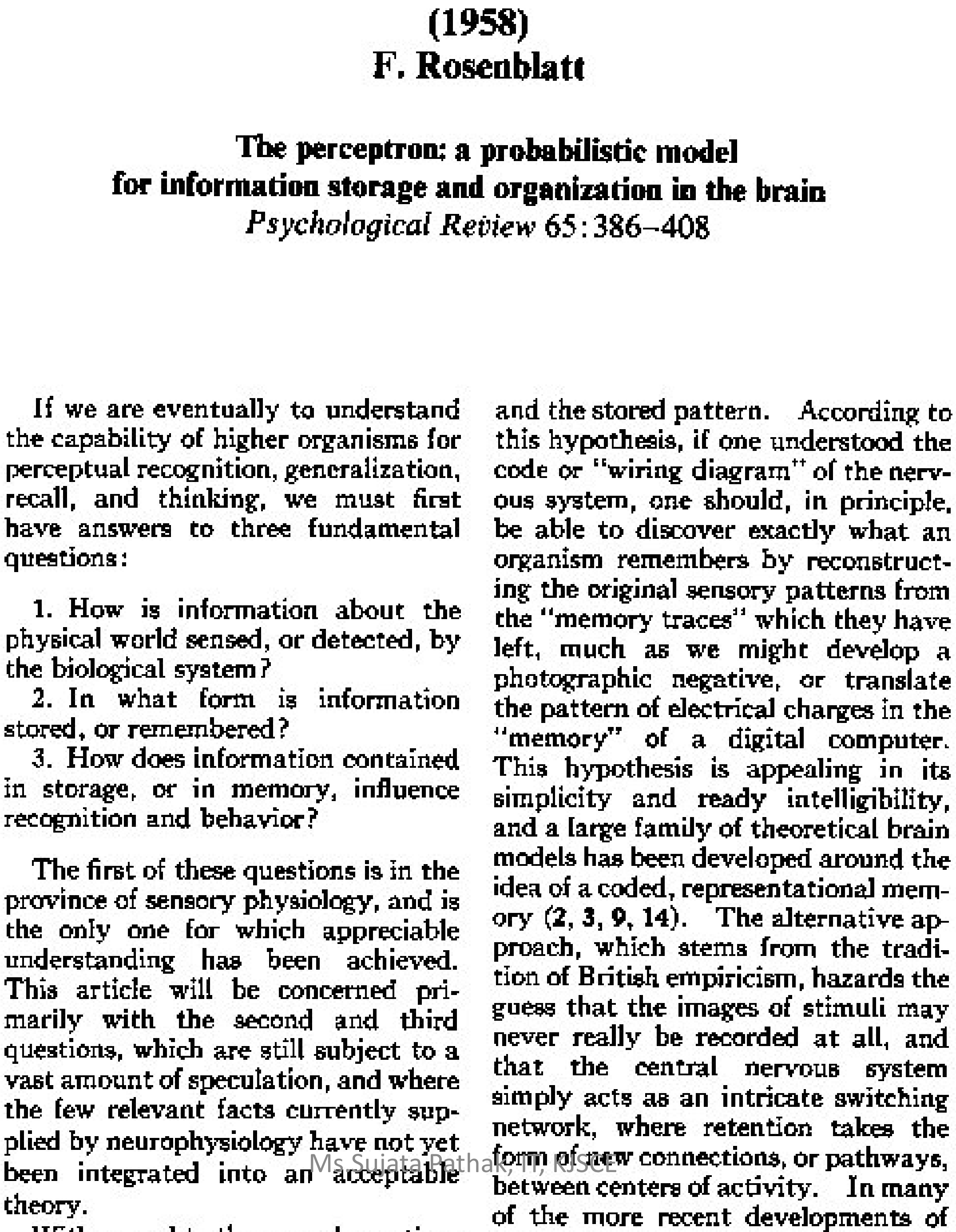
Perceptrons





Bias Replaces Threshold





(1958)
F. Rosenblatt

The perceptron: a probabilistic model for information storage and organization in the brain

Psychological Review 65:386–408

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

The first of these questions is in the province of sensory physiology, and is the only one for which appreciable understanding has been achieved. This article will be concerned primarily with the second and third questions, which are still subject to a vast amount of speculation, and where the few relevant facts currently supplied by neurophysiology have not yet been integrated into an acceptable theory.

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain models has been developed around the idea of a coded, representational memory (2, 3, 9, 14). The alternative approach, which stems from the tradition of British empiricism, hazards the guess that the images of stimuli may never really be recorded at all, and that the central nervous system simply acts as an intricate switching network, where retention takes the form of new connections, or pathways, between centers of activity. In many of the more recent developments of

Ms Sujata Pathak, IT, KJSCE

Perceptron Decision = Recall

- Outputs are:

$$y_j = \text{sign}\left(\sum_{i=1}^{n} w_{ij} x_i\right)$$

$$\Rightarrow \mathbf{w} \cdot \mathbf{x} > 0$$

For example, y=(y_1, …, y_5)=(1, 0, 0, 1, 1) is a possible output.
We may have a different function **g** in the place of sign, as in (2.4) in the book.

Perceptron Learning = Updating the Weights

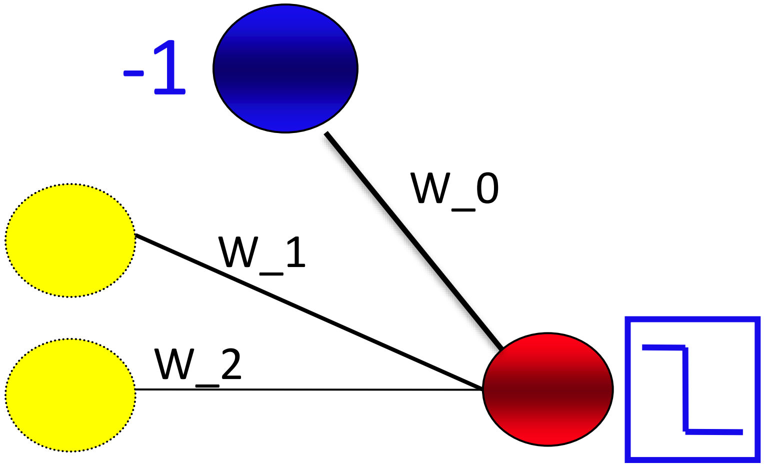
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

- We want to change the values of the weights
- Aim: minimise the *error* at the output
- If *E = t-y*, want *E* to be 0
- Use:

$$\Delta w_{ij} = \overset{\text{Learning rate}}{\eta} \cdot \underset{\text{Error}}{(t_j - y_j)} \cdot \overset{\text{Input}}{x_i}$$

Example 1: The Logical OR

X_1	X_2	t
0	0	0
0	1	1
1	0	1
1	1	1



Initial values: w_0(0)=-0.05, w_1(0) =-0.02, w_2(0)=0.02, and η=0.25

Take first row of our training table:

y_1= sign(-0.05×-1 + -0.02×0 + 0.02×0) = 1

w_0(1) = -0.05 + 0.25×(0-1)×-1=0.2

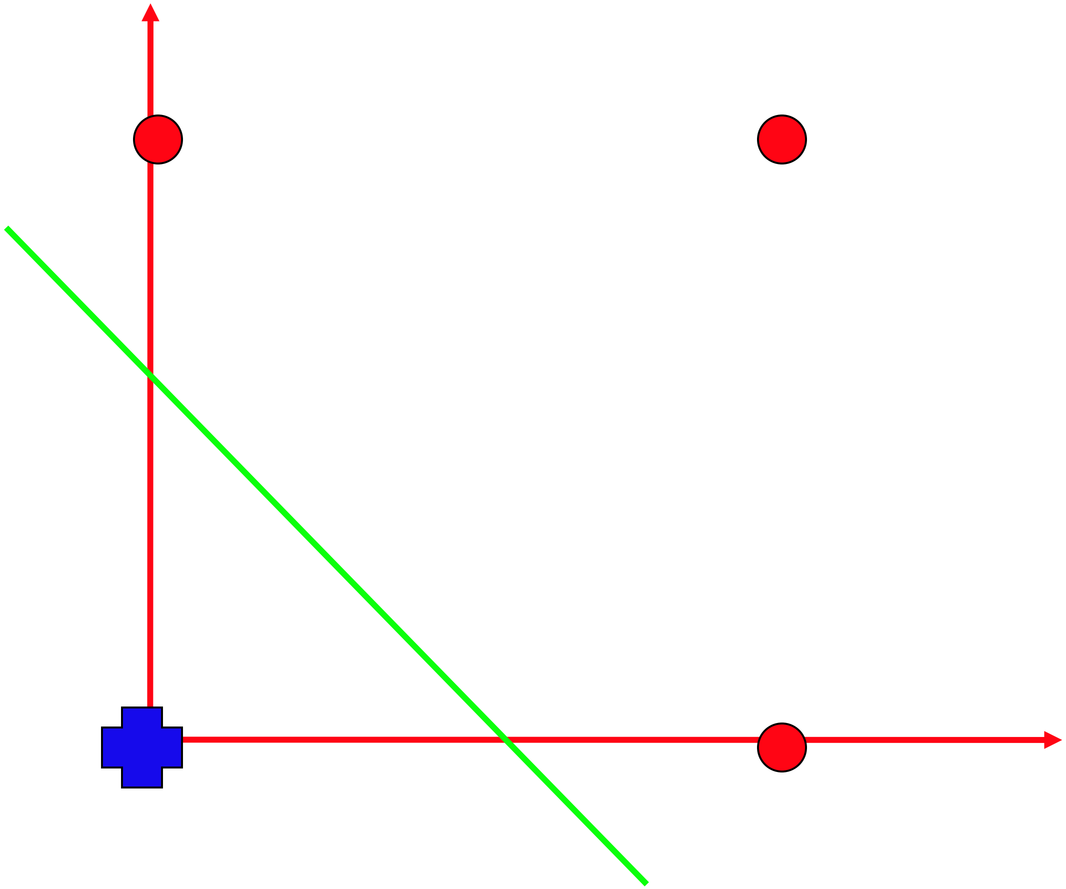
w_1(1) = -0.02 + 0.25×(0-1)×0=-0.02

w_2(1) = 0.02 + 0.25×(0-1)×0=0.02

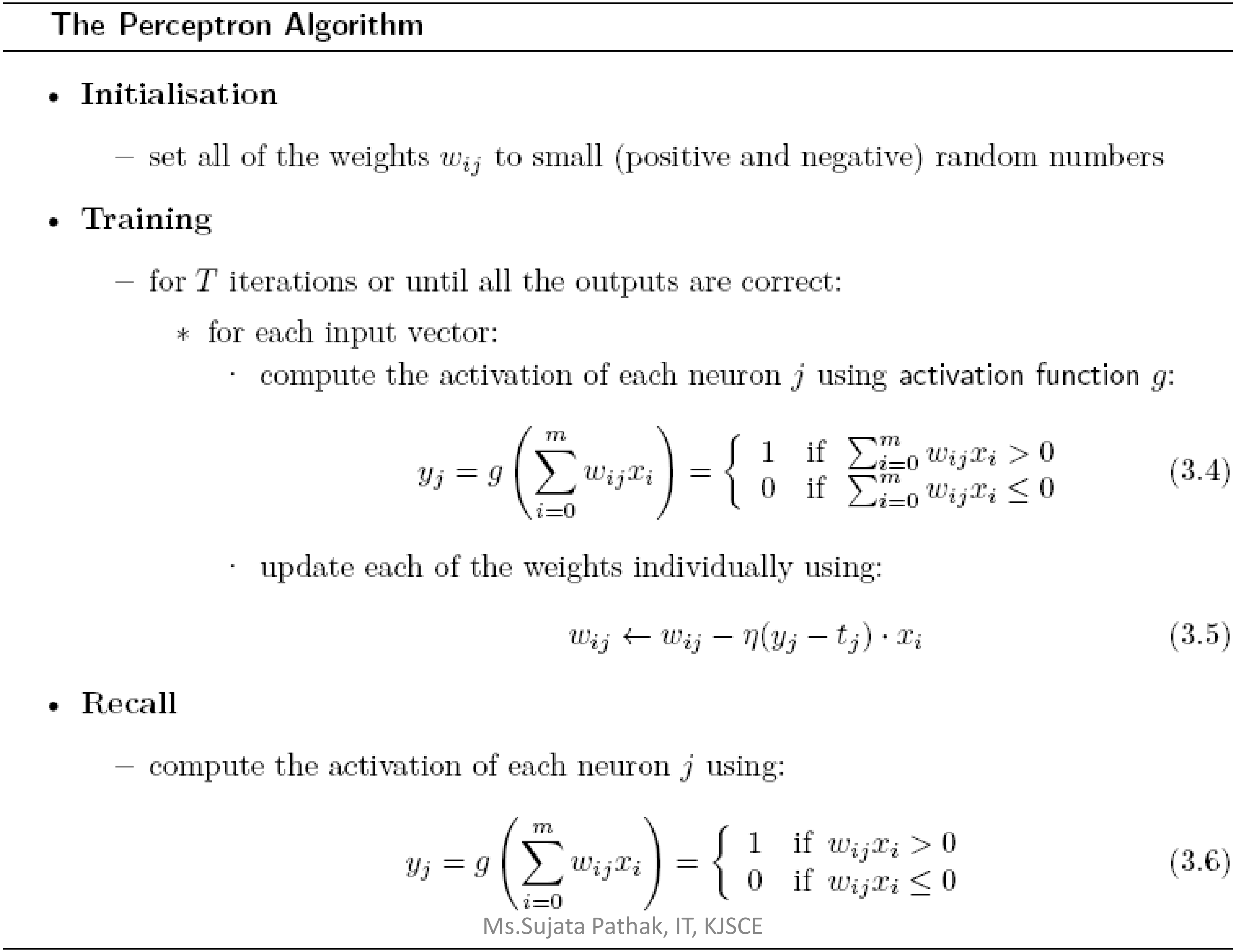
We continue with the new weights and the second row, and so on

We make several passes over the training data.

Decision boundary for OR perceptron



The Perceptron Learning Algorithm



The Perceptron Algorithm

- **Initialisation**
  - set all of the weights w_{ij} to small (positive and negative) random numbers
- **Training**
  - for T iterations or until all the outputs are correct:
    - for each input vector:
      - compute the activation of each neuron j using activation function g:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij}x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij}x_i \leq 0 \end{cases} \tag{3.4}$$

      - update each of the weights individually using:

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i \tag{3.5}$$

- **Recall**
  - compute the activation of each neuron j using:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } w_{ij}x_i > 0 \\ 0 & \text{if } w_{ij}x_i \leq 0 \end{cases} \tag{3.6}$$

MLP

```
# Import MLPClassifer
from sklearn.neural_network import MLPClassifier

# Create model object
clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=True,
                    learning_rate_init=0.01)

# Fit data onto the model
clf.fit(X_train,y_train)
```

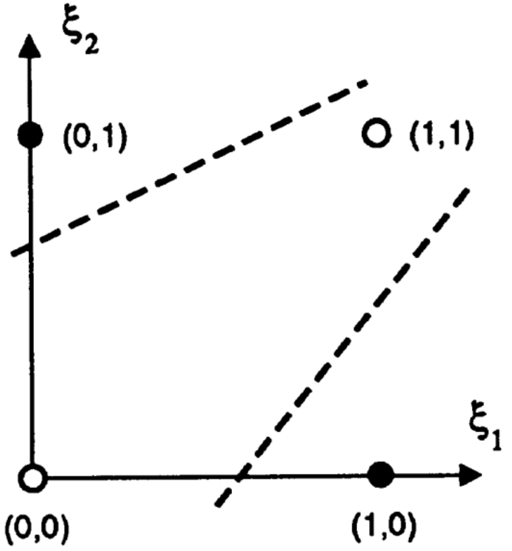
MLP

```
# Make prediction on test dataset
ypred=clf.predict(X_test)

# Import accuracy score
from sklearn.metrics import accuracy_score

# Calcuate accuracy
accuracy_score(y_test,ypred)
```

Solution in 1980s: Multilayer perceptrons

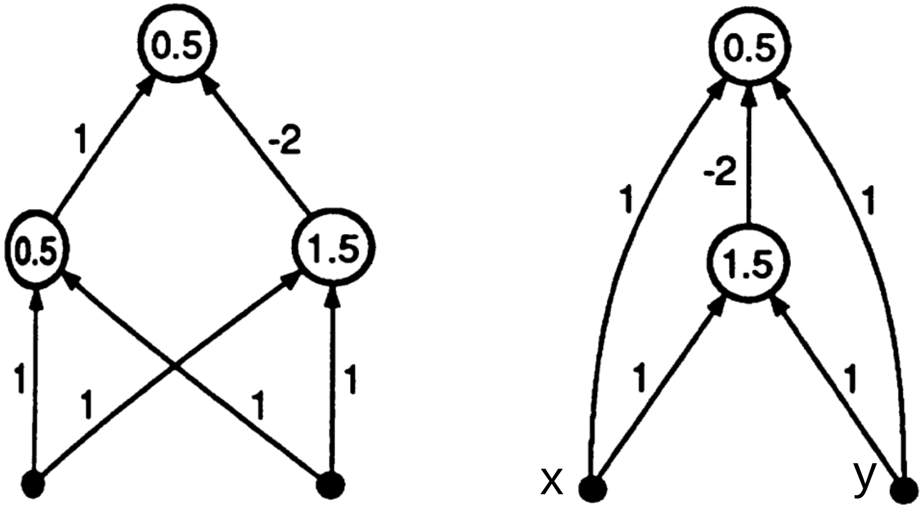
- Removes many limitations of single-layer networks
  - Can solve XOR

- Exercise: Draw a two-layer perceptron that computes the XOR function
  - 2 binary inputs ξ_1 and ξ_2
  - 1 binary output
  - One "hidden" layer
  - Find the appropriate weights and threshold



Solution in 1980s: Multilayer perceptrons

- Examples of two-layer perceptrons that compute XOR

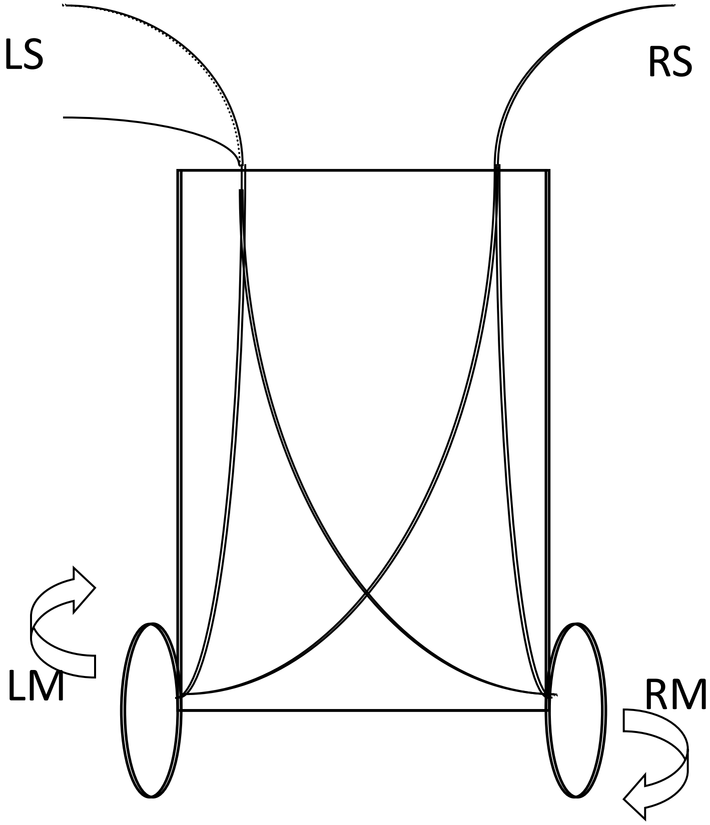


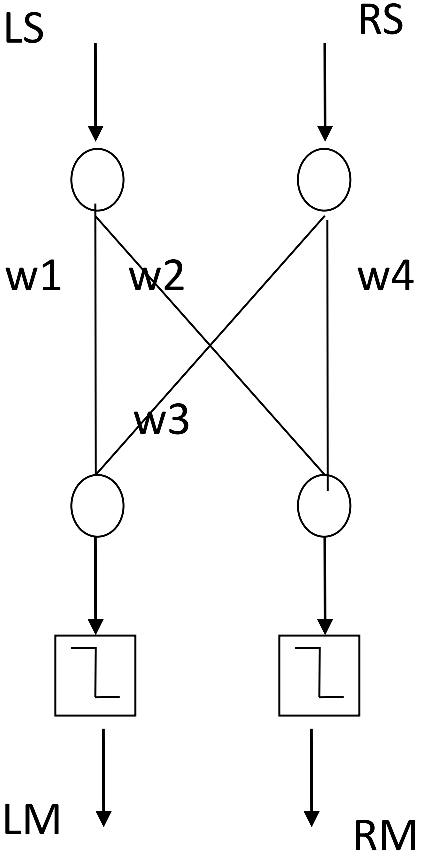
- E.g. Right side network
  - Output is 1 if and only if $x + y - 2(x + y - 1.5 > 0) - 0.5 > 0$

Perceptron Learning Applet

- http://lcn.epfl.ch/tutorial/english/perceptron/html/index.html

Example 2: Obstacle Avoidance with the Perceptron





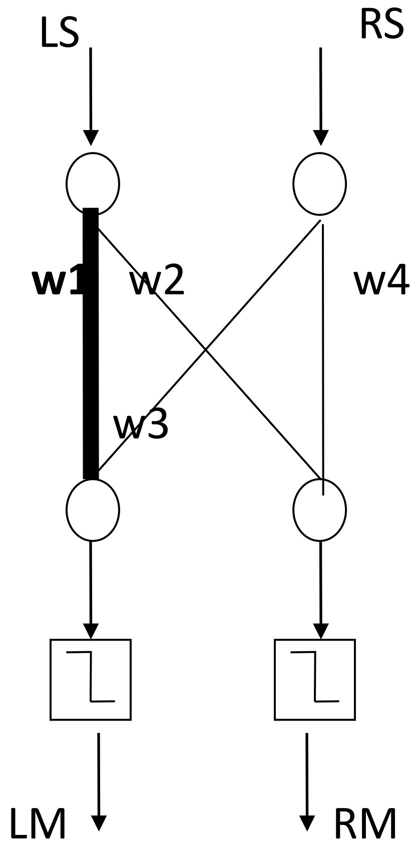
$\eta = 0.3$

$\theta = -0.01$

Obstacle Avoidance with the Perceptron

LS	**RS**	**LM**	**RM**
0	**0**	**1**	**1**
0	1	-1	1
1	0	1	-1
1	1	X	X

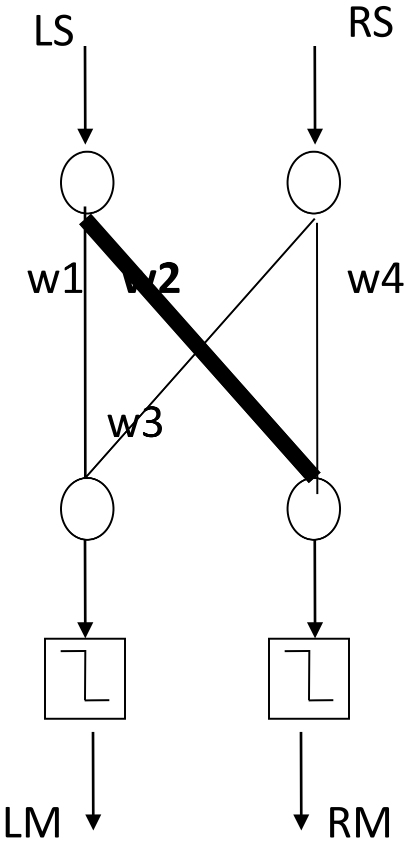
Obstacle Avoidance with the Perceptron



$$\Delta w_{ij} = \eta \cdot (t_j - y_j) \cdot x_i$$

w1=0+0.3 * (1-1) * 0 = 0

Obstacle Avoidance with the Perceptron



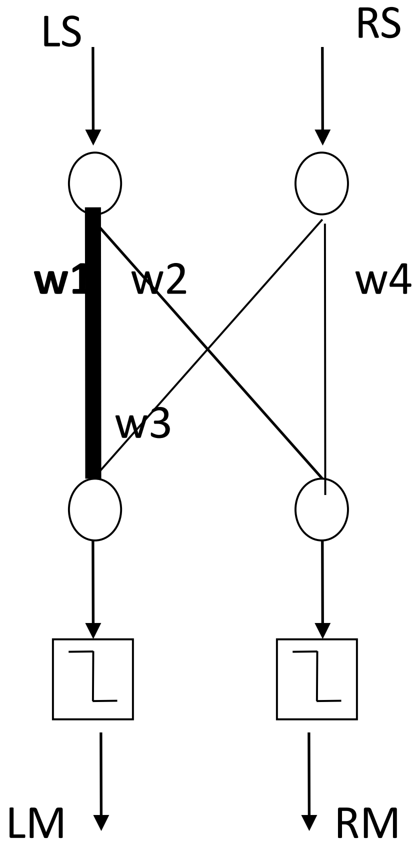
w2=0+0.3 * (1-1) * 0 = 0

And the same for w3, w4

Obstacle Avoidance with the Perceptron

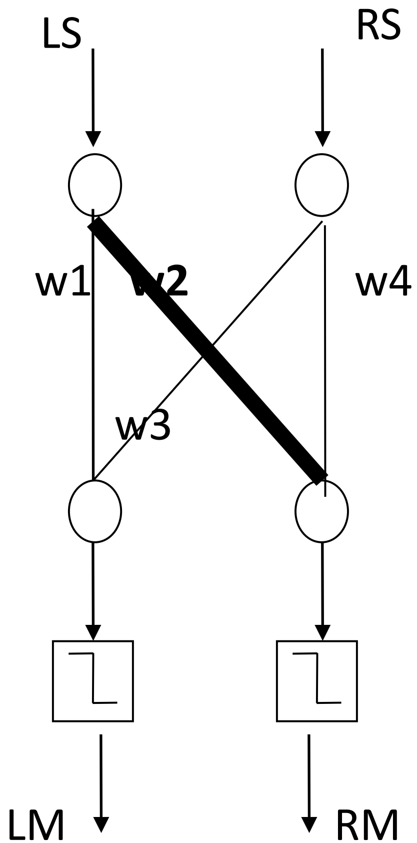
LS	**RS**	**LM**	**RM**
0	0	1	1
0	**1**	**-1**	**1**
1	0	1	-1
1	1	X	X

Example 1: Obstacle Avoidance with the Perceptron



$w1=0+0.3 * (-1-1) * 0 = 0$

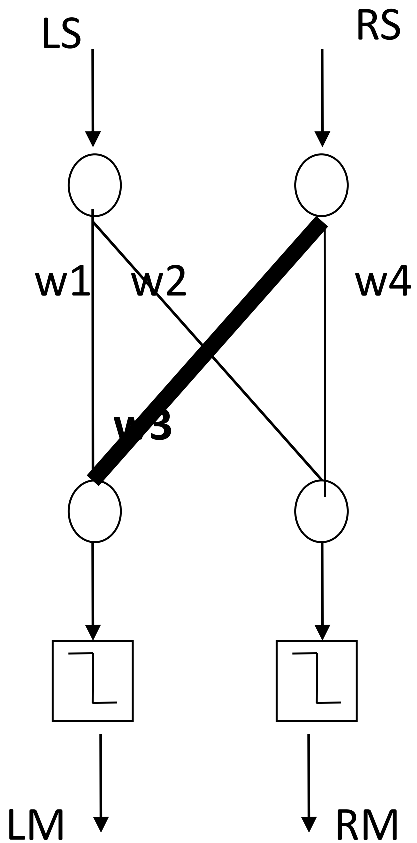
Obstacle Avoidance with the Perceptron



w1=0+0.3 * (-1-1) * 0 = 0

w2=0+0.3 * (1-1) * 0 = 0

Obstacle Avoidance with the Perceptron

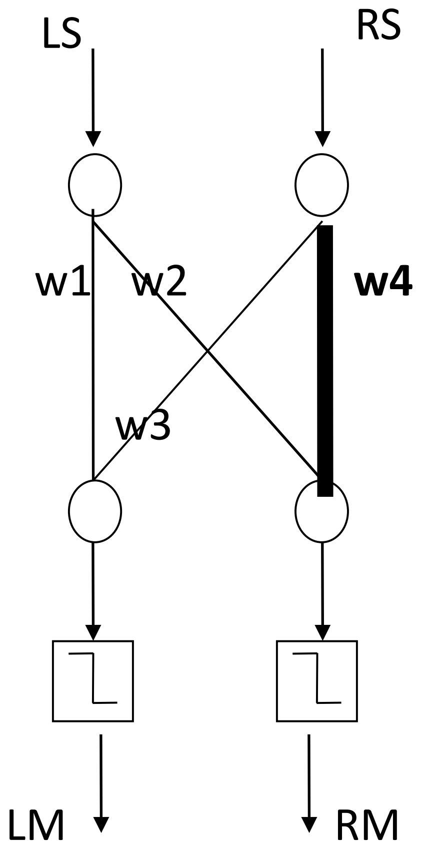


w1=0+0.3 * (-1-1) * 0 = 0

w2=0+0.3 * (1-1) * 0 = 0

w3=0+0.3 * (-1-1) * 1 = -0.6

Obstacle Avoidance with the Perceptron



w1=0+0.3 * (-1-1) * 0 = 0

w2=0+0.3 * (1-1) * 0 = 0

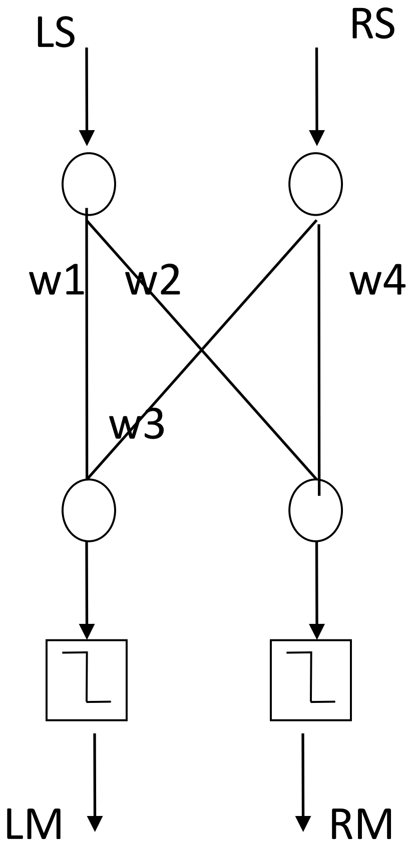
w3=0+0.3 * (-1-1) * 1 = -0.6

w4=0+0.3 * (1-1) * 1 = 0

Obstacle Avoidance with the Perceptron

LS	**RS**	**LM**	**RM**
0	0	1	1
0	1	-1	1
1	**0**	**1**	**-1**
1	1	X	X

Obstacle Avoidance with the Perceptron



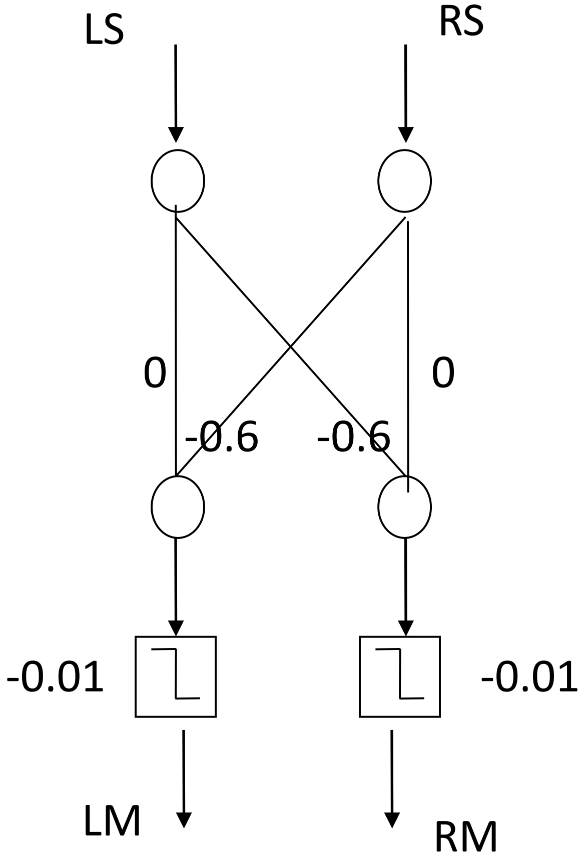
w1=0+0.3 * (1-1) * 1 = 0

w2=0+0.3 * (-1-1) * 1 = -0.6

w3=-0.6+0.3 * (1-1) * 0 = -0.6

w4=0+0.3 * (-1-1) * 0 = 0

Obstacle Avoidance with the Perceptron



2.3 Linear Separability

- Outputs are:

$$y_j = \operatorname{sign}\left(\sum_{i=1}^{n} w_{ij} x_i\right)$$

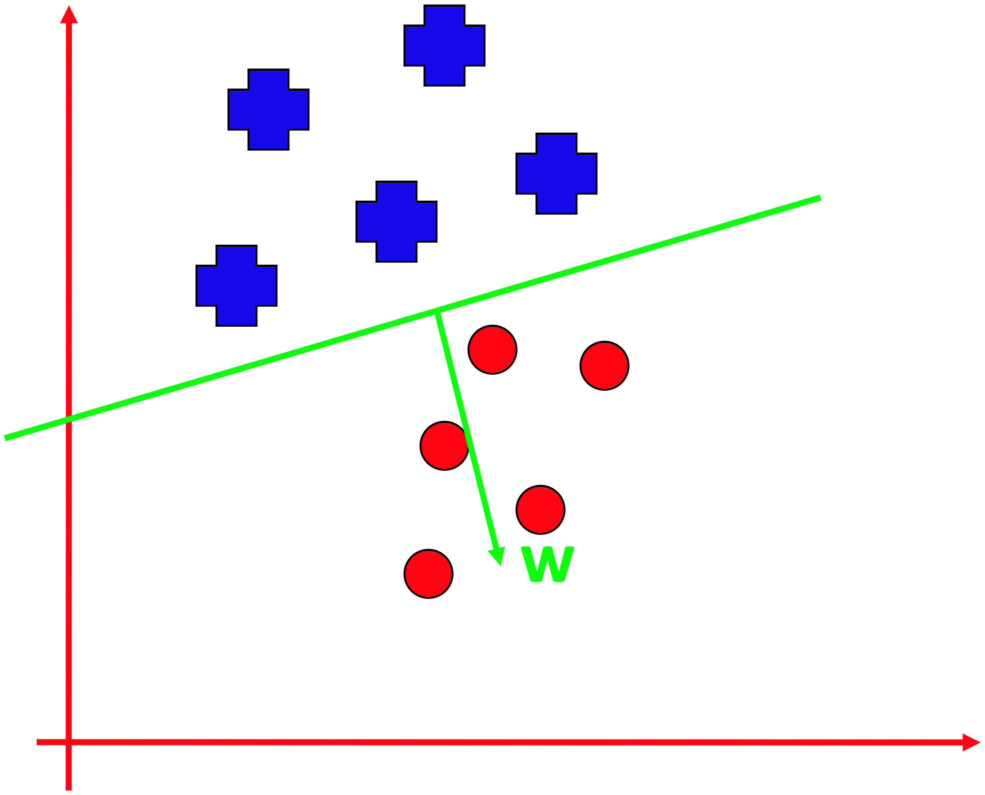
$$\Rightarrow \mathbf{w} \cdot \mathbf{x} > 0$$

where

$$\mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| \times \|\mathbf{x}\| \cos \alpha$$

and α is the angle between vectors **x** and **w**.

Geometry of linear Separability



The equation of a line is
w_0 + w_1*x + w_2*y=0
It also means that point (x,y) is on the line

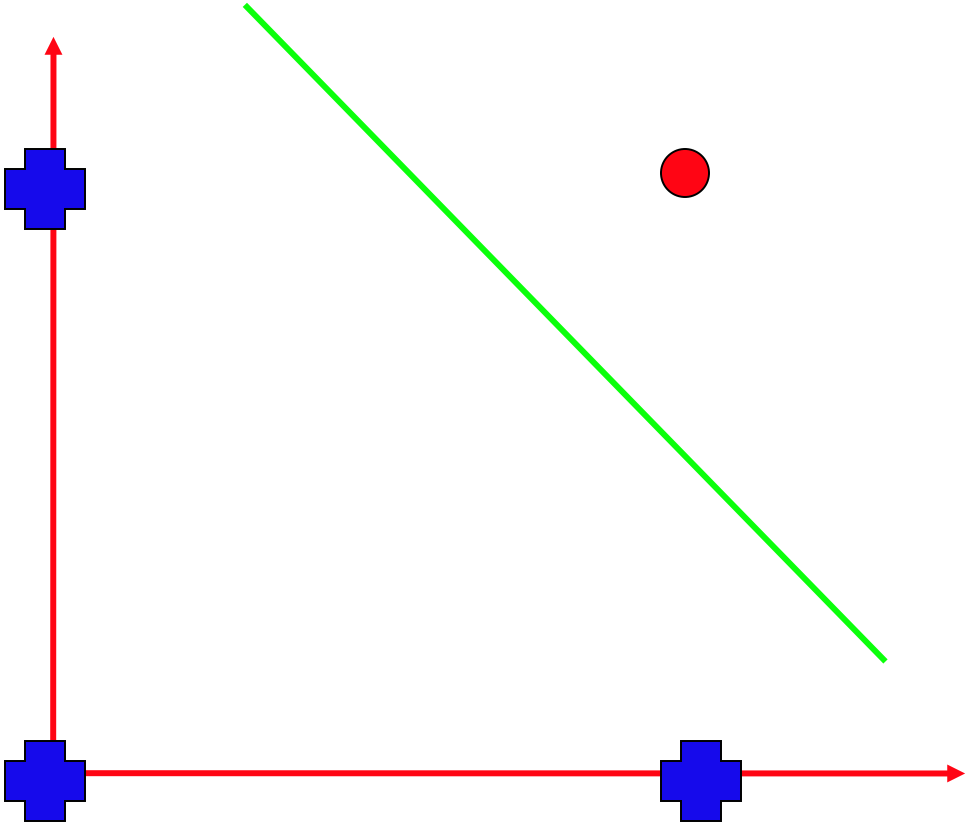
This equation is equivalent to

$\mathbf{w} \cdot \mathbf{x}$ = (w_0, w_1,w_2) ·(1,x,y) = 0

If $\mathbf{w} \cdot \mathbf{x} > 0$, then the angle
between **w** and **x**
is less than 90 degree, which means that
w and **x** lie on the same side of the line.

Each output node of perceptron tries to separate the training data
Into two classes (fire or no-fire) with a linear decision boundary,
i.e., straight line in 2D, plane in 3D, and hyperplane in higher dim.

Linear Separability

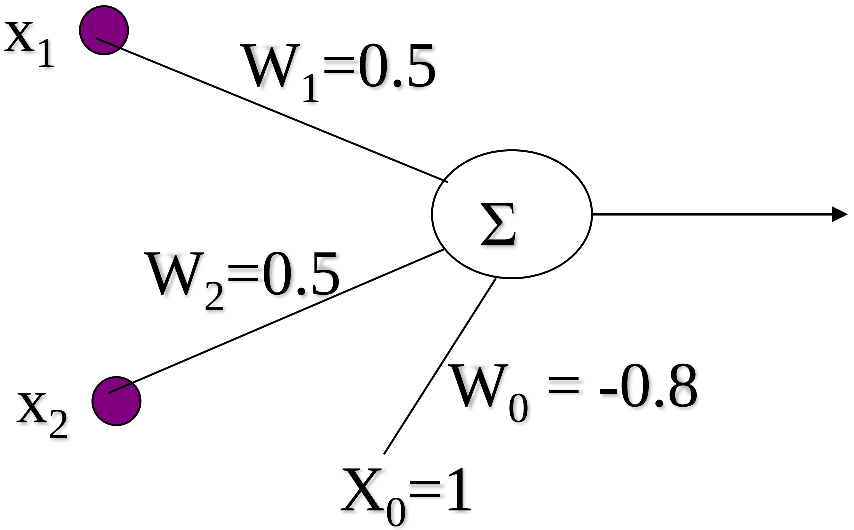


The Binary AND Function

Perceptrons

Perceptrons can learn many boolean functions:
AND, OR, NAND, NOR, but not XOR

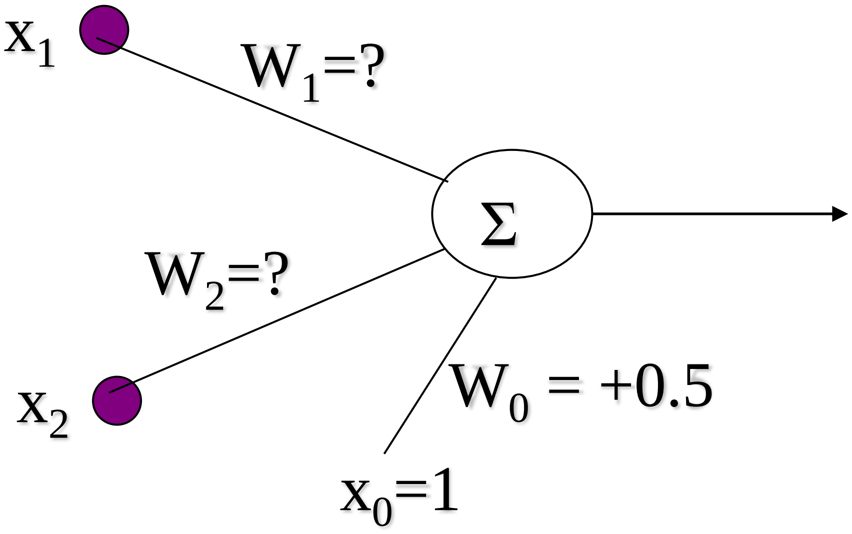
AND:



Every boolean function can be represented with a perceptron network that has two levels of depth or more.

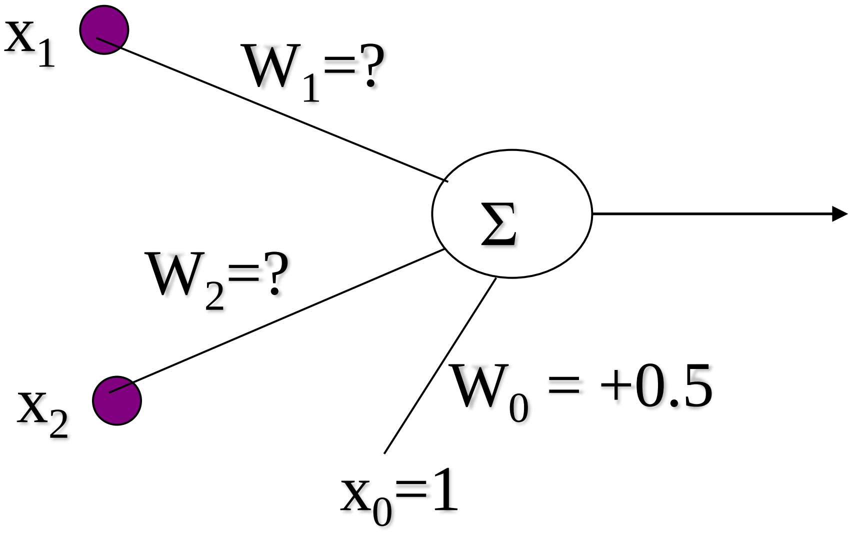
Design of Primitive Units

Design a two-input perceptron that implements the Boolean function X_1 OR ~X_2 (X_1 OR not X_2). Assume the independent weight is always +0.5 (assume W_0 = +0.5 and X_0 = 1). You simply have to provide adequate values for W_1 and W_2.



Design of Primitive Units

X_1 OR ~X_2 (X_1 OR not X_2).

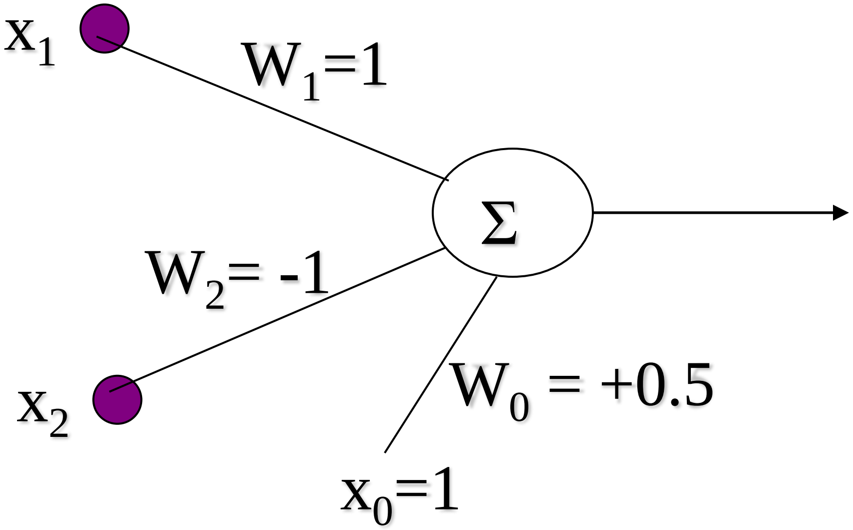


Design of Primitive Units

There are different solutions, but a general solution is that:

$W_2 < -0.5$
$W_1 > |W_2| - 0.5$
where |x| is the absolute value of x.

(X$_1$ OR not X$_2$).



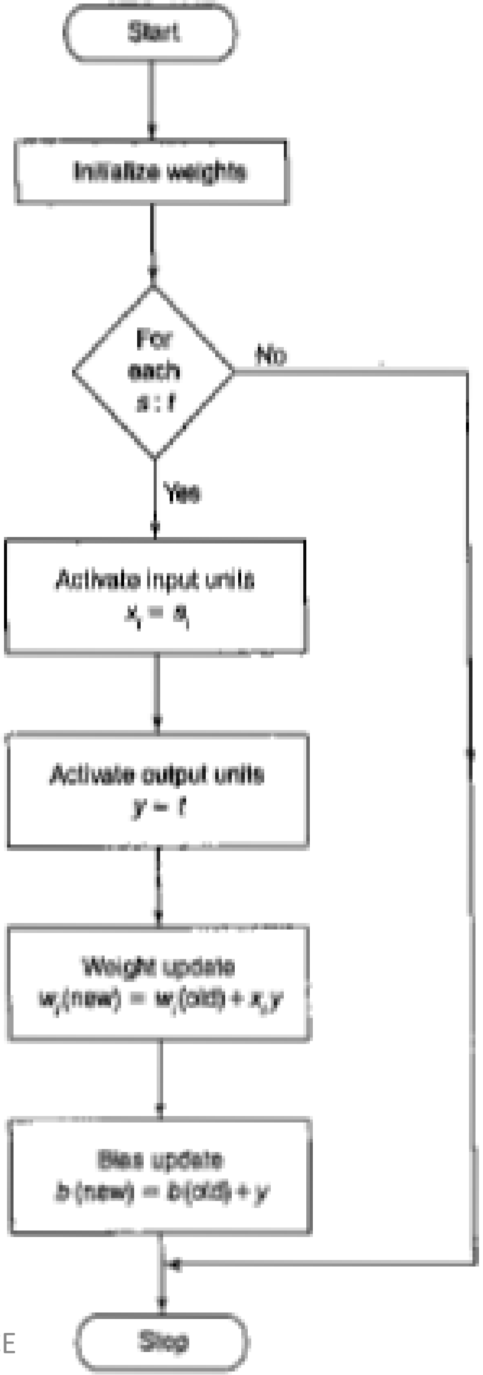
Hebb’s Rule

- Hebb’s rule says that-
  - “Changes in the strength of synaptic connections are proportional to the correlation in the firing of the two connecting neurons.
  - So if two neurons consistently fire simultaneously, then any connection between them will change in strength, becoming stronger.
  - However, if the two neurons never fire simultaneously, the connection between them will die away.

Hebb’s Rule

– The idea is that if two neurons both respond to something, then they should be connected.

Hebb’s Rule



Start

Initialize weights

For each s : t

No

Yes

Activate input units
$x_i = s_i$

Activate output units
$y = t$

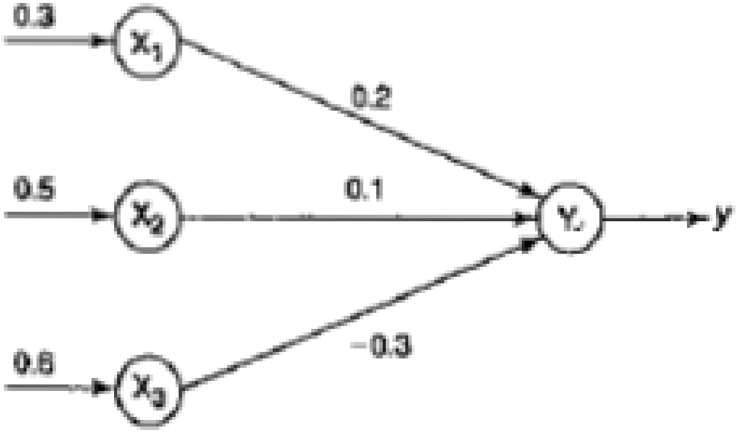
Weight update
$w_i(\text{new}) = w_i(\text{old}) + x_i y$

Bias update
$b(\text{new}) = b(\text{old}) + y$

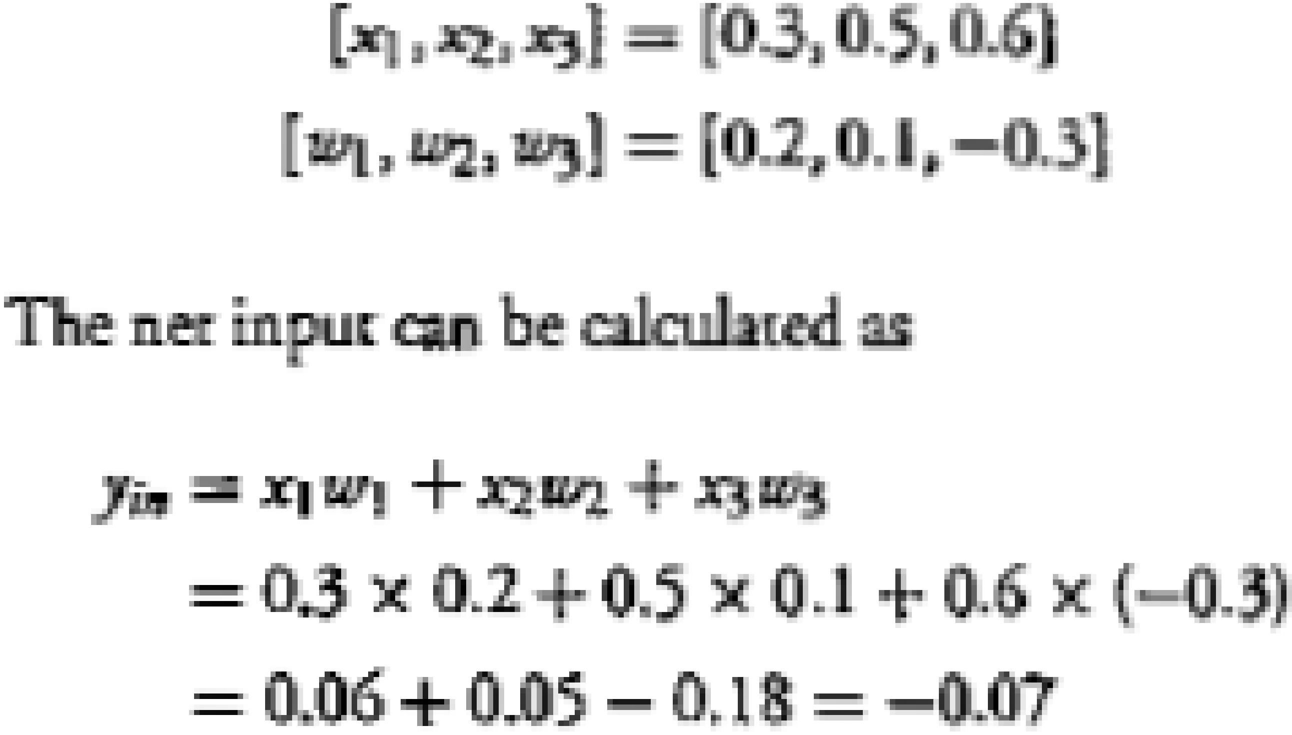
Stop

Problem-1

- For the n/w shown calculate the net input to o/p neuron



Problem-1-soln



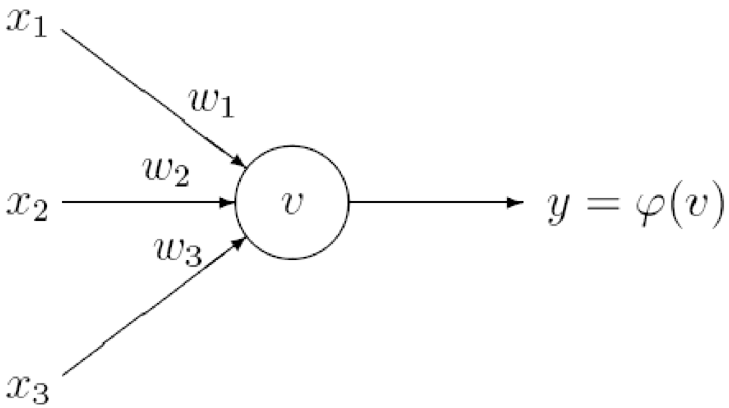
$$[x_1, x_2, x_3] = [0.3, 0.5, 0.6]$$

$$[w_1, w_2, w_3] = [0.2, 0.1, -0.3]$$

The net input can be calculated as

$$\begin{aligned} y_{in} &= x_1 w_1 + x_2 w_2 + x_3 w_3 \\ &= 0.3 \times 0.2 + 0.5 \times 0.1 + 0.6 \times (-0.3) \\ &= 0.06 + 0.05 - 0.18 = -0.07 \end{aligned}$$

Problem -2



- Suppose that the weights corresponding to the three inputs have the following values:

$$\boxed{\begin{aligned} w_1 &= 2 \\ w_2 &= -4 \\ w_3 &= 1 \end{aligned}}$$

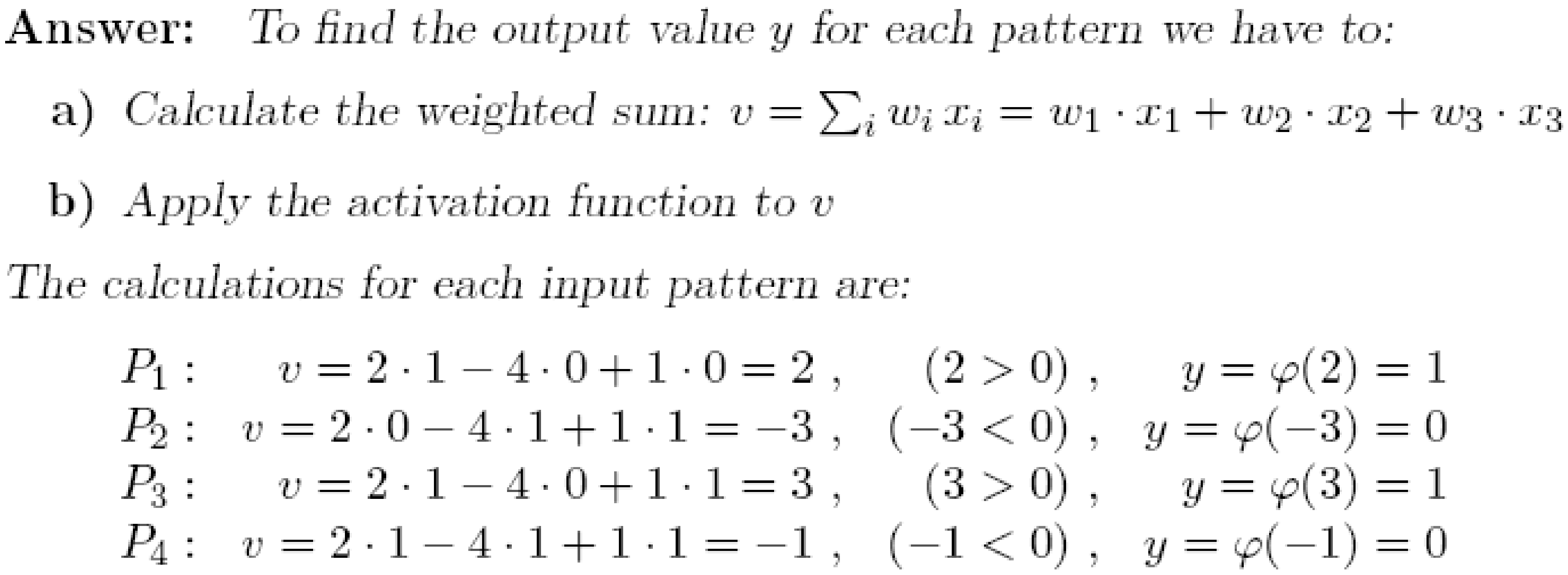
and the activation of the unit is given by the step-function:

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

Calculate what will be the output value y of the unit for each of the following input patterns:

Pattern	P_1	P_2	P_3	P_4
x_1	1	0	1	1
x_2	0	1	0	1
x_3	0	1	1	1

Problem-2-solution



Answer: *To find the output value y for each pattern we have to:*

a) *Calculate the weighted sum:* $v = \sum_i w_i\, x_i = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3$

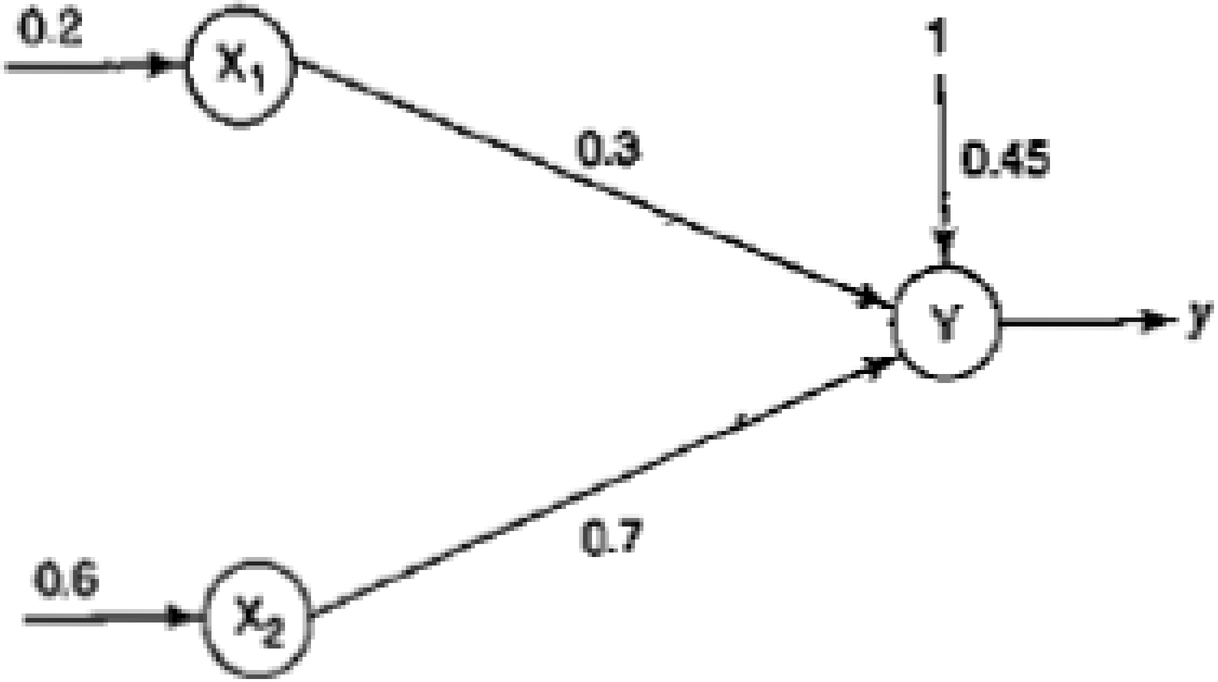
b) *Apply the activation function to* v

The calculations for each input pattern are:

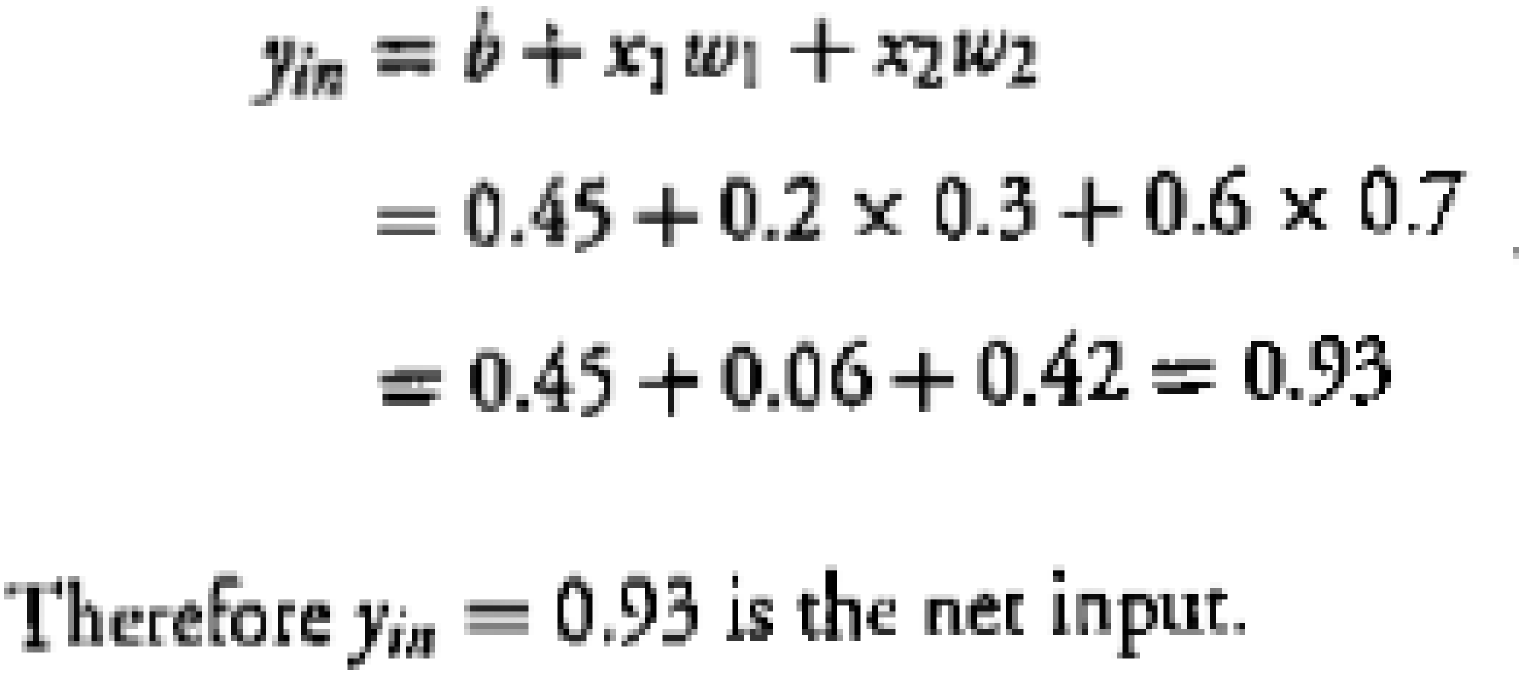
$$\begin{aligned}
P_1: &\quad v = 2 \cdot 1 - 4 \cdot 0 + 1 \cdot 0 = 2\ , &\quad (2 > 0)\ , &\quad y = \varphi(2) = 1\\
P_2: &\quad v = 2 \cdot 0 - 4 \cdot 1 + 1 \cdot 1 = -3\ , &\quad (-3 < 0)\ , &\quad y = \varphi(-3) = 0\\
P_3: &\quad v = 2 \cdot 1 - 4 \cdot 0 + 1 \cdot 1 = 3\ , &\quad (3 > 0)\ , &\quad y = \varphi(3) = 1\\
P_4: &\quad v = 2 \cdot 1 - 4 \cdot 1 + 1 \cdot 1 = -1\ , &\quad (-1 < 0)\ , &\quad y = \varphi(-1) = 0
\end{aligned}$$

Problem-3

- Calculate the ner input for the network shown in below Figure with bias included in the network.



Problem-3-soln



$$y_{in} = b + x_1 w_1 + x_2 w_2$$

$$= 0.45 + 0.2 \times 0.3 + 0.6 \times 0.7$$

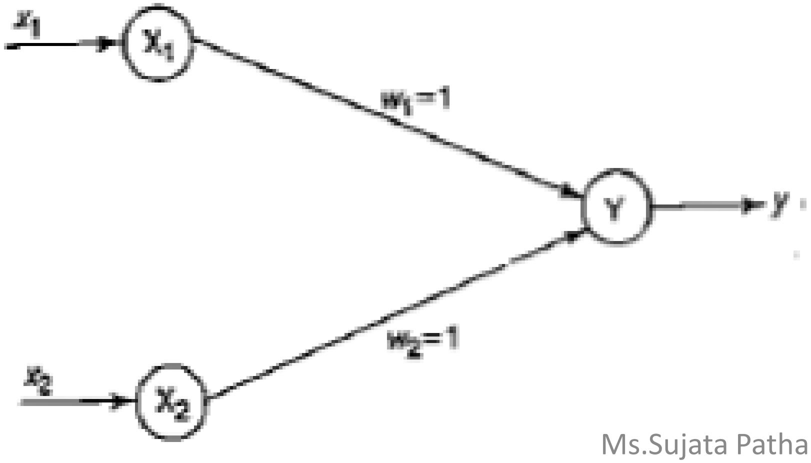
$$= 0.45 + 0.06 + 0.42 = 0.93$$

Therefore $y_{in} = 0.93$ is the net input.

Problem-4

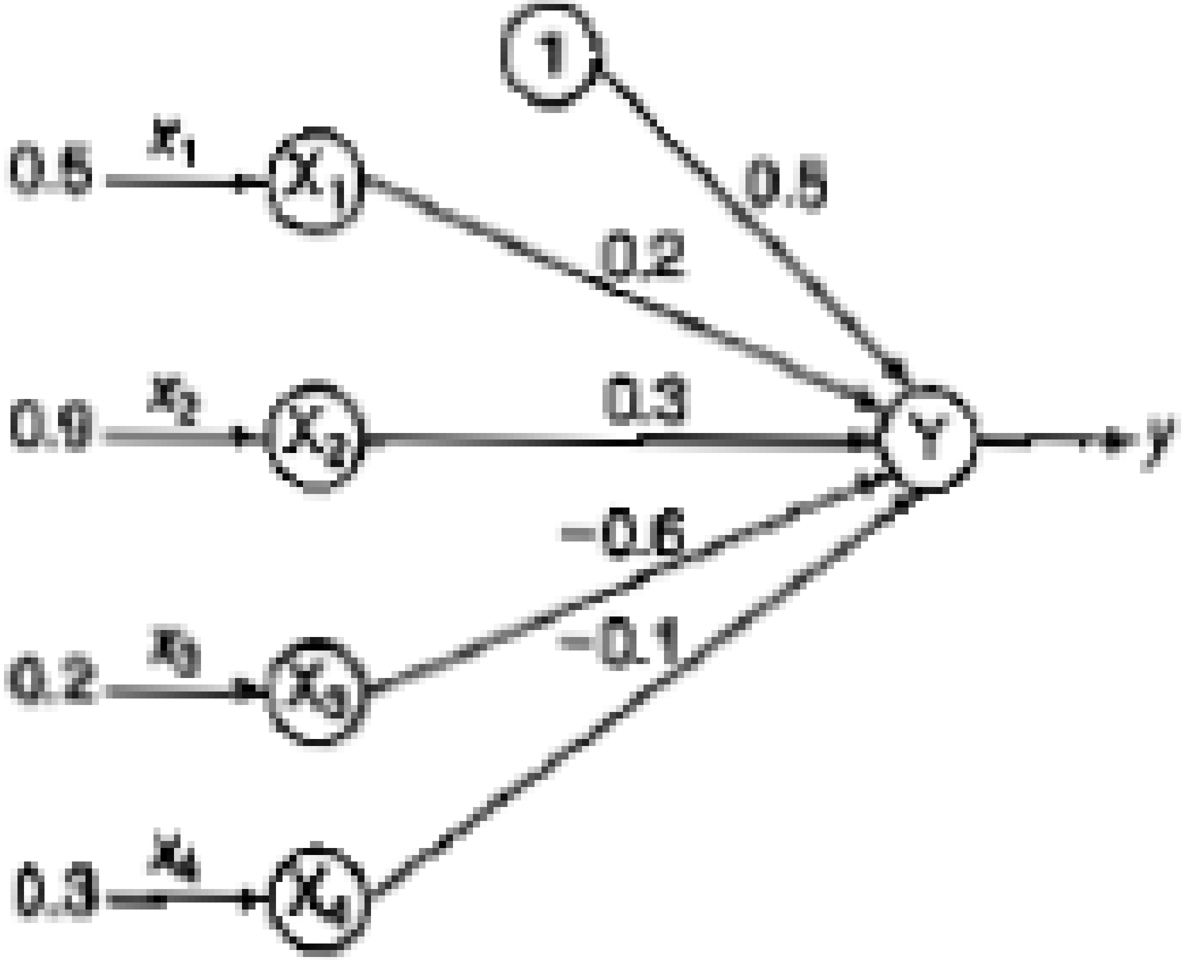
- Implement AND function using McCulloch-Pitts neuron.

$$(1,1),\quad y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$$
$$(1,0),\quad y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1$$
$$(0,1),\quad y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 1 \times 1 = 1$$
$$(0,0),\quad y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0$$



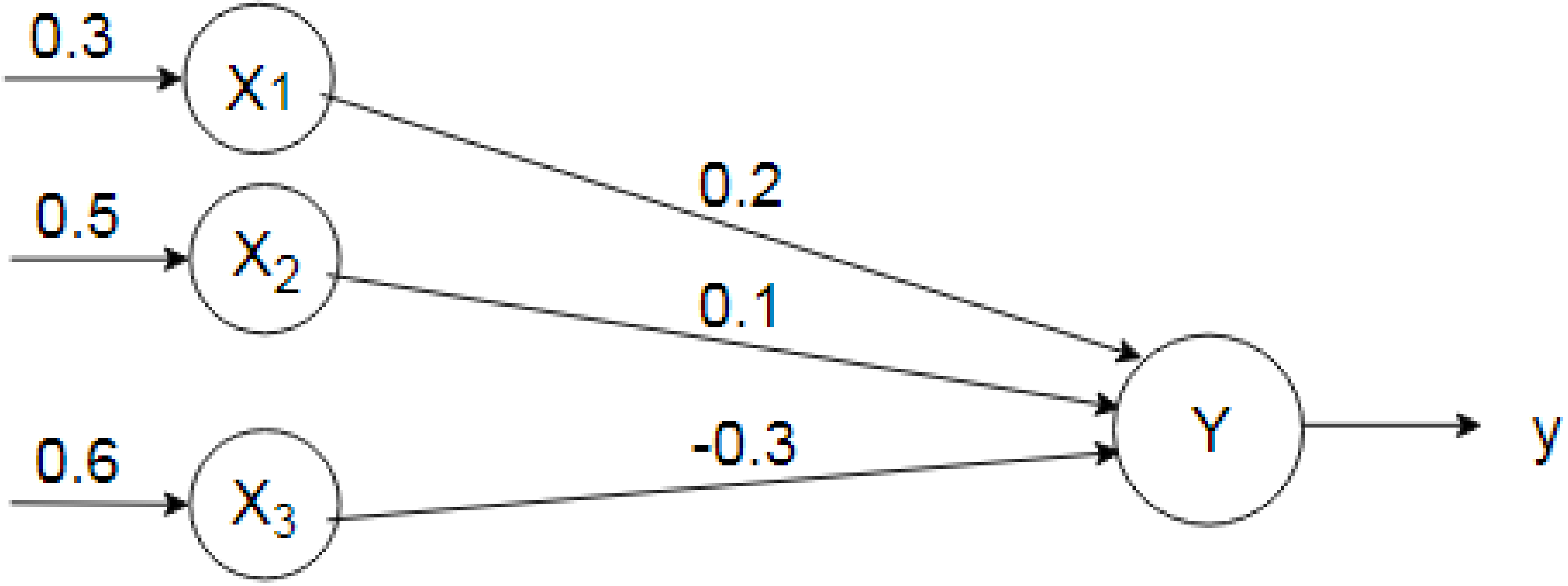
$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geqslant 2 \\ 0 & \text{if } y_{in} < 2 \end{cases}$$

Problem-4



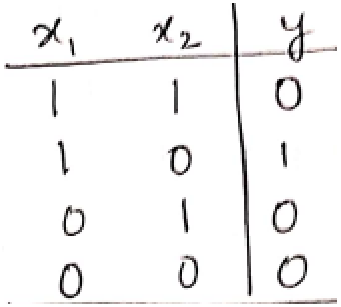
Problem-5

- For the network shown in figure, calculate the net input to the neuron?

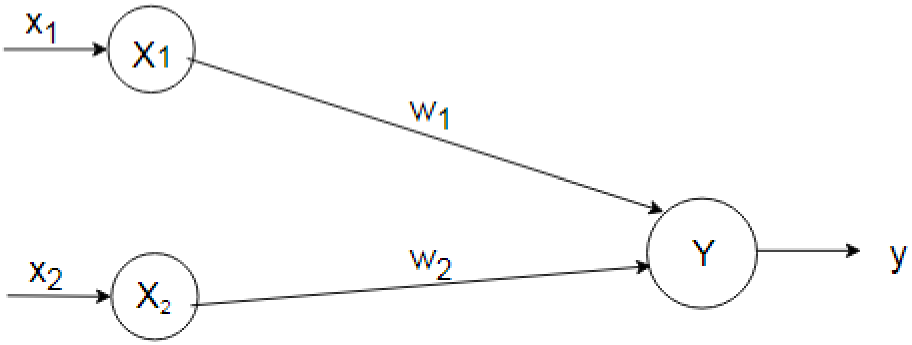


Problem-6

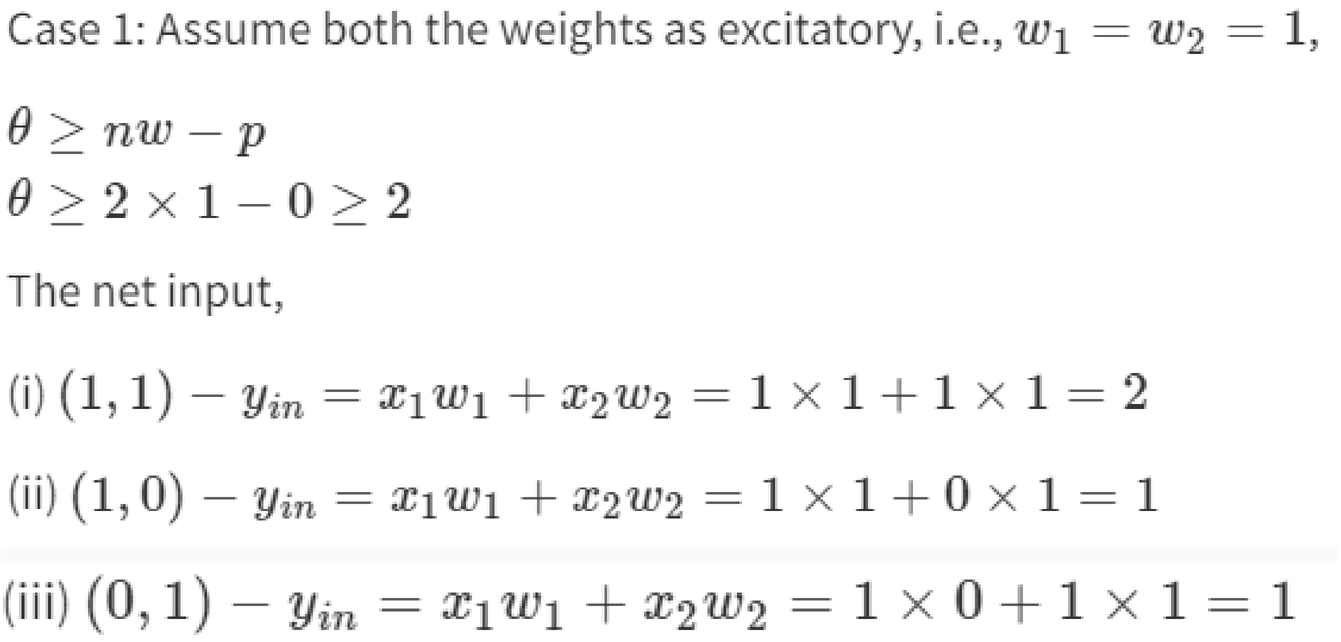
- Use McCulloch-Pitts Neuron to implement AND NOT function (take binary data representation).



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



Problem-6



Case 1: Assume both the weights as excitatory, i.e., $w_1 = w_2 = 1$,

$$\theta \geq nw - p$$
$$\theta \geq 2 \times 1 - 0 \geq 2$$

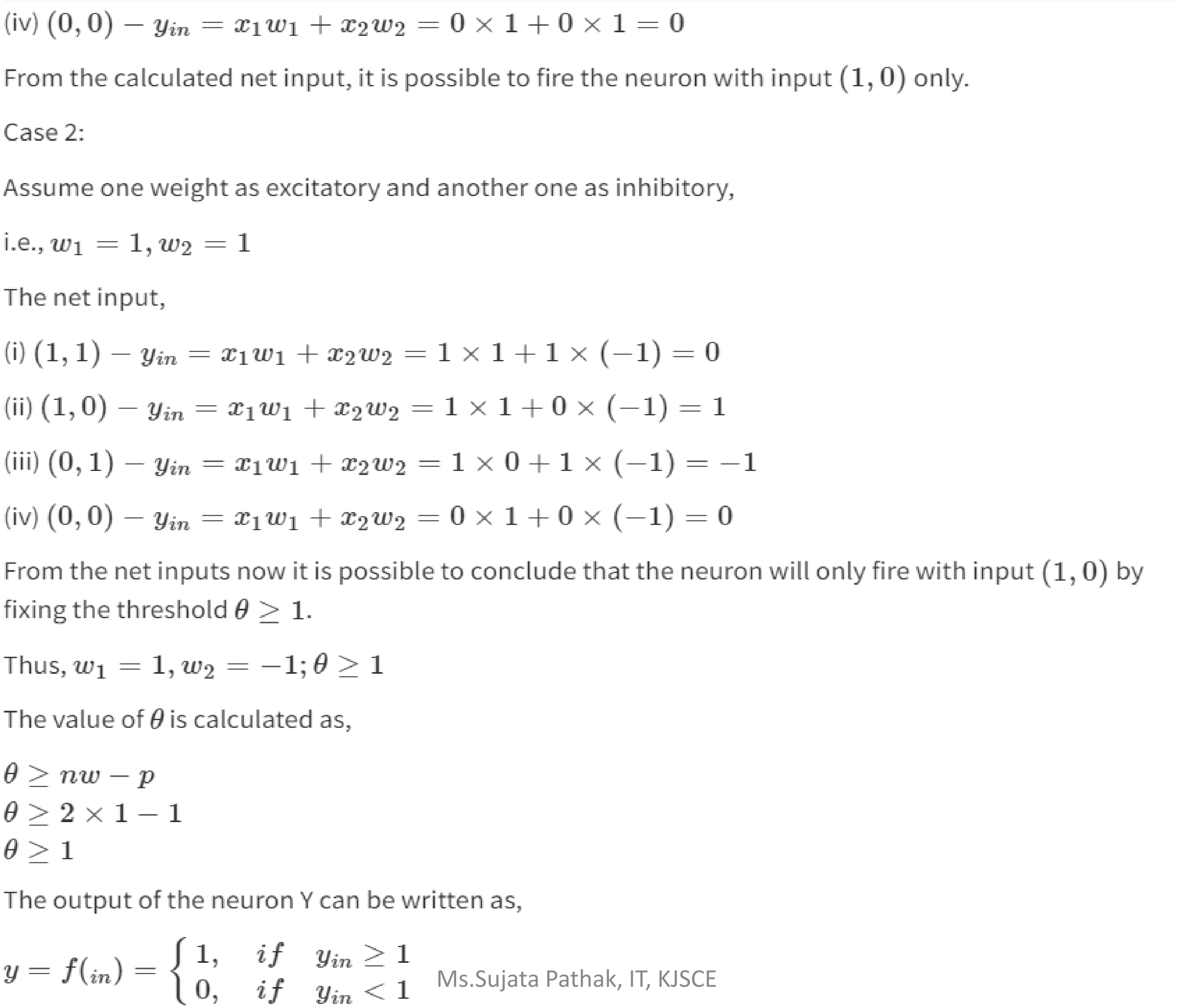
The net input,

(i) $(1, 1) - y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$

(ii) $(1, 0) - y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1$

(iii) $(0, 1) - y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 0 + 1 \times 1 = 1$

Problem-6



(iv) $(0, 0) - y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0$

From the calculated net input, it is possible to fire the neuron with input $(1, 0)$ only.

Case 2:

Assume one weight as excitatory and another one as inhibitory,

i.e., $w_1 = 1, w_2 = 1$

The net input,

(i) $(1, 1) - y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times (-1) = 0$

(ii) $(1, 0) - y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times (-1) = 1$

(iii) $(0, 1) - y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 0 + 1 \times (-1) = -1$

(iv) $(0, 0) - y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times (-1) = 0$

From the net inputs now it is possible to conclude that the neuron will only fire with input $(1, 0)$ by fixing the threshold $\theta \geq 1$.

Thus, $w_1 = 1, w_2 = -1; \theta \geq 1$

The value of θ is calculated as,

$$\theta \geq nw - p$$
$$\theta \geq 2 \times 1 - 1$$
$$\theta \geq 1$$

The output of the neuron Y can be written as,

$$y = f(_{in}) = \begin{cases} 1, & if \quad y_{in} \geq 1 \\ 0, & if \quad y_{in} < 1 \end{cases}$$