

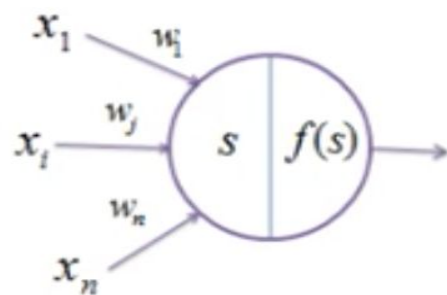
# The Perceptron

# 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

# Perceptron (an artificial neuron)

- A perceptron **models** a neuron
- It receives **n inputs** (corresponding to **features**)
- It sums those inputs, checks the result and produces an output
- It is used to classify **linearly separable** classes
- Often for **binary classification**



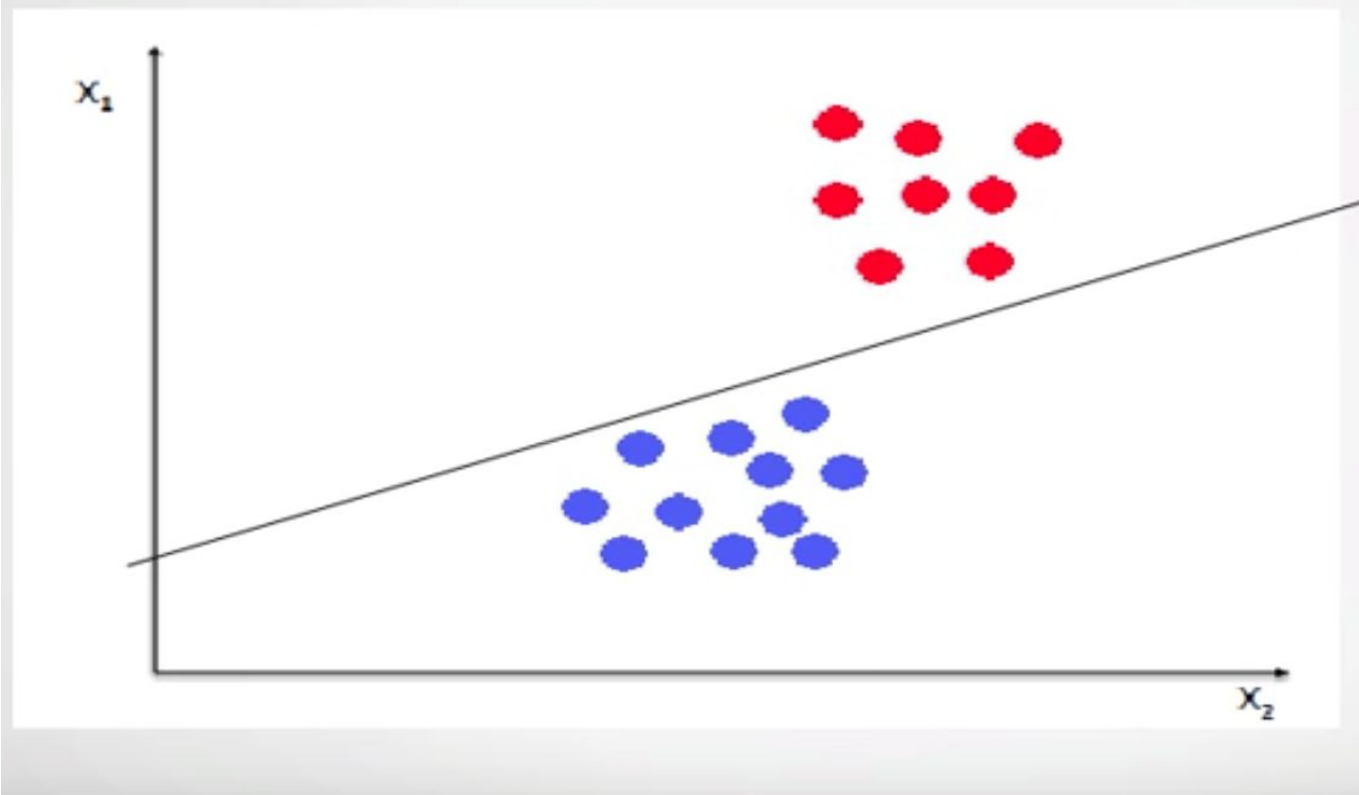
Summation

$$s = \sum_{i=1}^n w_i \cdot x_i$$

Transformation



# Linear/Non-linear Separability



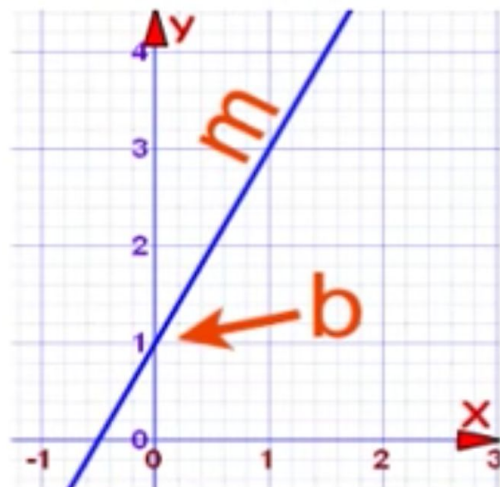
# An Artificial Neuron

- The perceptron consists of weights, the summation processor, and an activation function
- A perceptron takes a weighted sum of inputs and outputs:
  - 1 if the sum is  $>$  some adjustable threshold value ( $\theta$ )
  - 0 otherwise

	Output
$w_1x_1 + w_2x_2 + \dots + w_nx_n > \theta$	$\Rightarrow 1$
$w_1x_1 + w_2x_2 + \dots + w_nx_n \leq \theta$	$\Rightarrow 0$

- The inputs and connection weights are typically real values

What does it stand for?



$$y = mx + b$$

Slope (or Gradient)

Y Intercept

**y** = how far up

**x** = how far along

**m** = Slope or Gradient (how steep the line is)

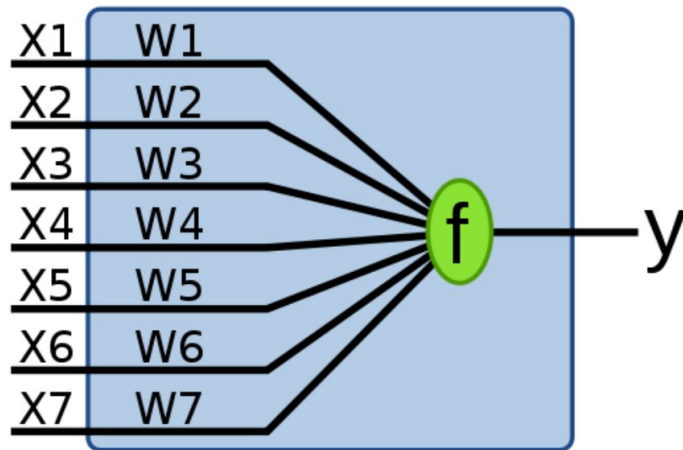
**b** = the Y Intercept (where the line crosses the Y axis)

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:

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

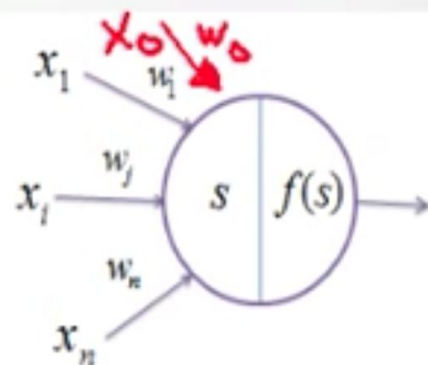


In order to not explicitly write  $b$ , we extend the input vector  $x$  by one more dimension that is always set to  $-1$ , e.g.,  $x=(-1, x_1, \dots, x_7)$  with  $x_0=-1$ , and extend the weight vector to  $w=(w_0, w_1, \dots, w_7)$ . Then adjusting  $w_0$  corresponds to adjusting  $b$ .



# The Role of Weights and Bias

- The perceptron can have another input *known as the bias*
- It is normal practice is to treat the bias as just another input
- The bias allows us to shift the transfer function curve horizontally (left/right) along the input axis while leaving the shape/curvature unaltered
- The weights determine the slope





## So ... In Plain English

- If we have  $n$  variables, then we need to find  $n+1$  weight values ( $n$  variables + the bias)
- These will be the coefficients in the equation of the separation line|plane|hyperplane
- For example, if we have 4 inputs, the equation becomes:

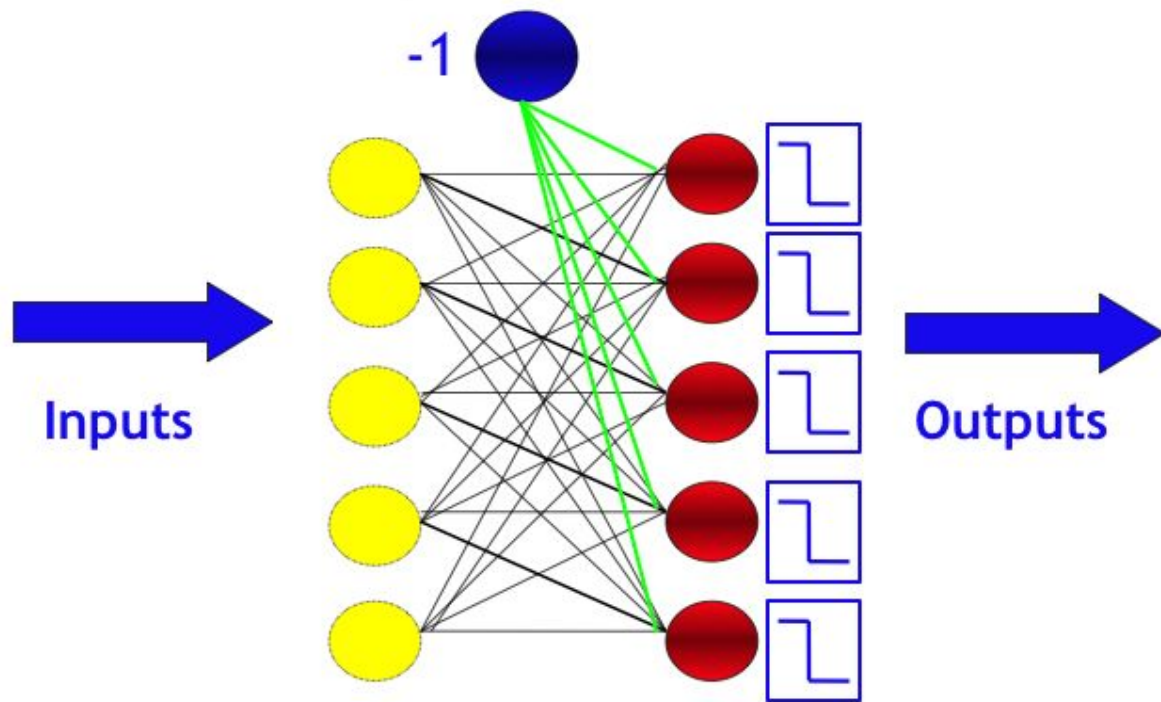
$$w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 = 0$$

which is equivalent to:

$$w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b = 0$$

*where  $w_i$  is the weight of input  $i$  and  $b$  is the bias ( $w_0$  with input value ( $x_0$ ) of 1)*

# Bias Replaces Threshold



# 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,  $y=(y_1, \dots, y_5)=(1, 0, 0, 1, 1)$  is a possible output.

We may have a different function  $\mathbf{g}$  in the place of  $\text{sign}$ , as in (2.4) in the book.

## Perceptron Learning = Updating the Weights

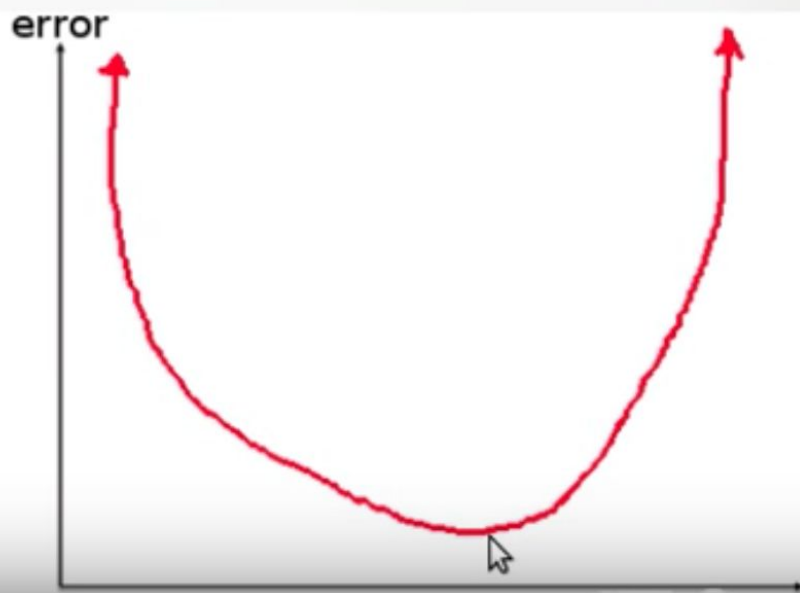
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

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

Learning rate      Input  
Error

# The learning rate

- We would like to update the weights and bias in order to get a smaller error
- The learning rate helps us control how much we change the weight and bias



# The Perceptron 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  $g$ :

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

- update each of the weights individually using:

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

- **Recall**

- compute the activation of each neuron  $j$  using:

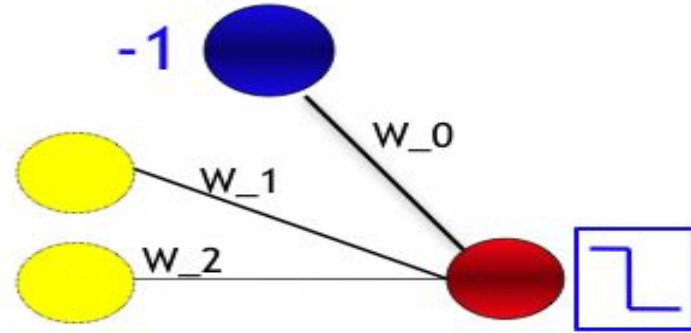
$$y_j = g\left(\sum_{i=0}^m w_{ij}x_i\right) = \begin{cases} 1 & \text{if } w_{ij}x_i > 0 \\ 0 & \text{if } w_{ij}x_i \leq 0 \end{cases} \quad (3.6)$$

---



# 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  $\eta=0.25$

Take first row of our training table:

$$y_1 = \text{sign}(-0.05 \times -1 + -0.02 \times 0 + 0.02 \times 0) = 1$$

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

$$w_0 : -0.05 - 0.25 \times (1 - 0) \times -1 = 0.2,$$

$$w_1 : -0.02 - 0.25 \times (1 - 0) \times 0 = -0.02,$$

$$w_2 : 0.02 - 0.25 \times (1 - 0) \times 0 = 0.02.$$

$$w_0 : 0.2 - 0.25 \times (0 - 1) \times -1 = -0.05,$$

$$w_1 : -0.02 - 0.25 \times (0 - 1) \times 0 = -0.02,$$

$$w_2 : 0.02 - 0.25 \times (0 - 1) \times 1 = 0.27.$$

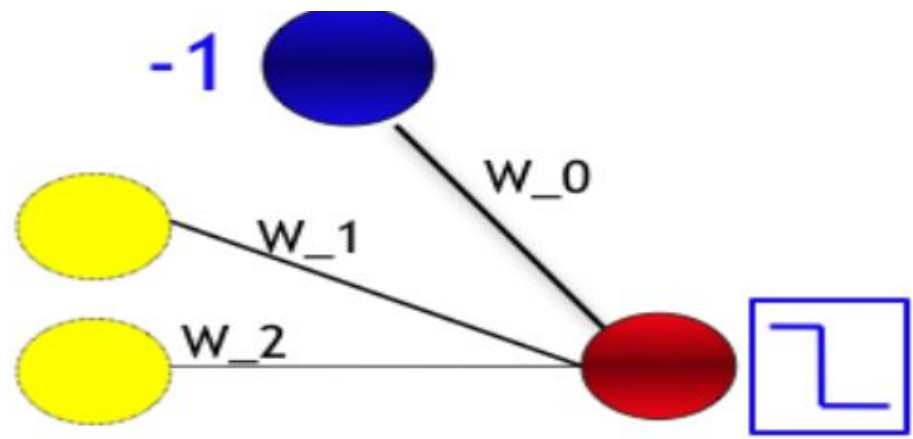
Iteration: 0  
[[-0.03755646]  
[ 0.01484562]  
[ 0.21173977]]

Final outputs are:  
[[0]  
[0]  
[0]  
[0]]

Iteration: 1  
[[ 0.46244354]  
[ 0.51484562]  
[-0.53826023]]

Final outputs are:  
[[1]  
[1]  
[1]  
[1]]

Iteration: 2  
[[ 0.46244354]  
[ 0.51484562]  
[-0.28826023]]



$x_1$	$x_2$	$t$
0	0	0
0	1	1
1	0	1
1	1	1

Final outputs are:  
[[1]  
[1]  
[1]  
[1]]

Iteration: 3  
[[ 0.46244354]  
[ 0.51484562]  
[-0.03826023]]

Final outputs are:  
[[1]  
[1]  
[1]  
[1]]

Iteration: 4  
[[ 0.46244354]  
[ 0.51484562]  
[ 0.21173977]]

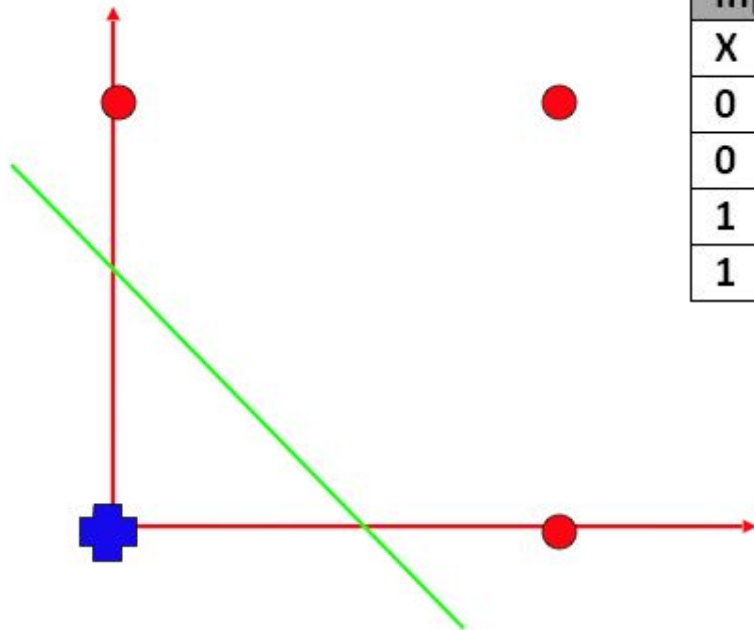
Final outputs are:  
[[0]  
[1]  
[1]  
[1]]

Iteration: 5  
[[ 0.46244354]  
[ 0.51484562]  
[ 0.21173977]]

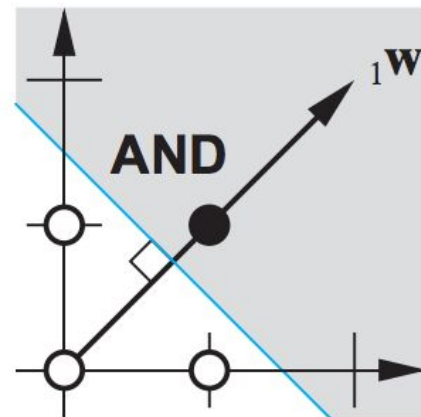
Final outputs are:  
[[0]  
[1]  
[1]  
[1]]

# Linear Separability

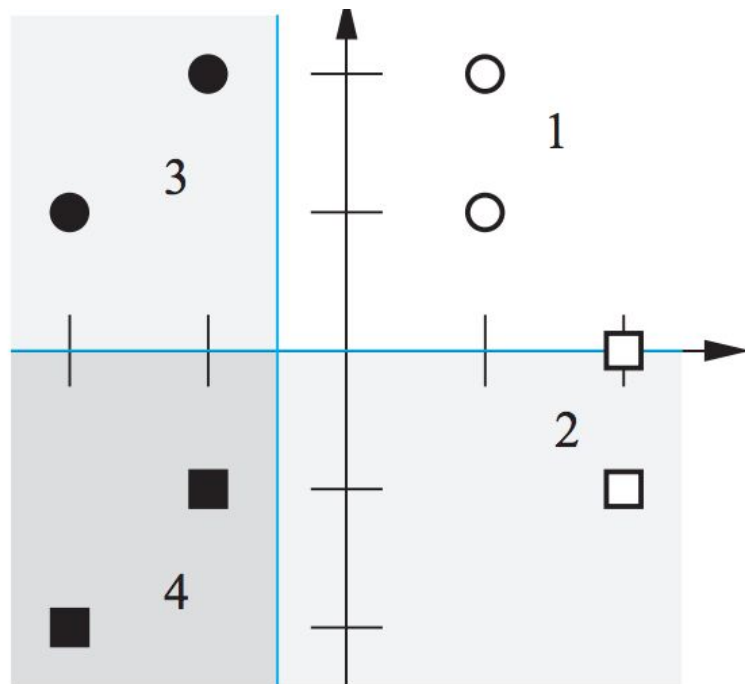
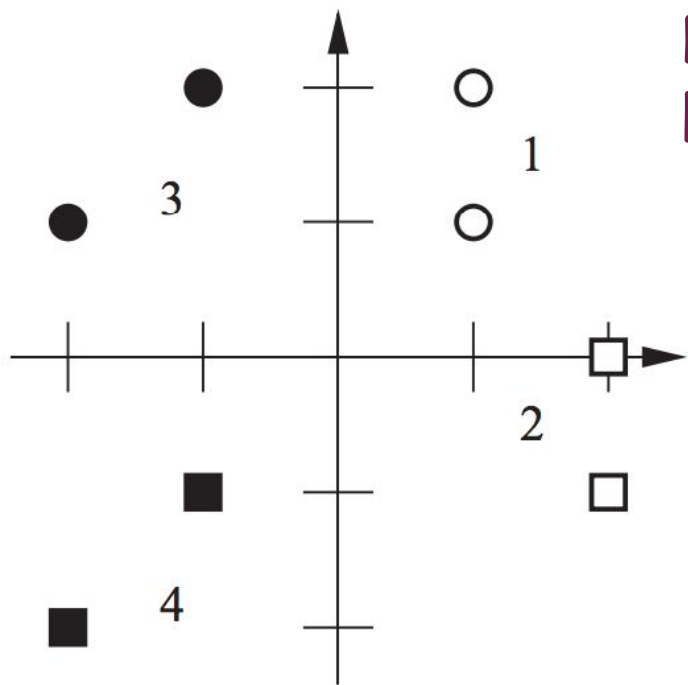
# Decision boundary for OR perceptron



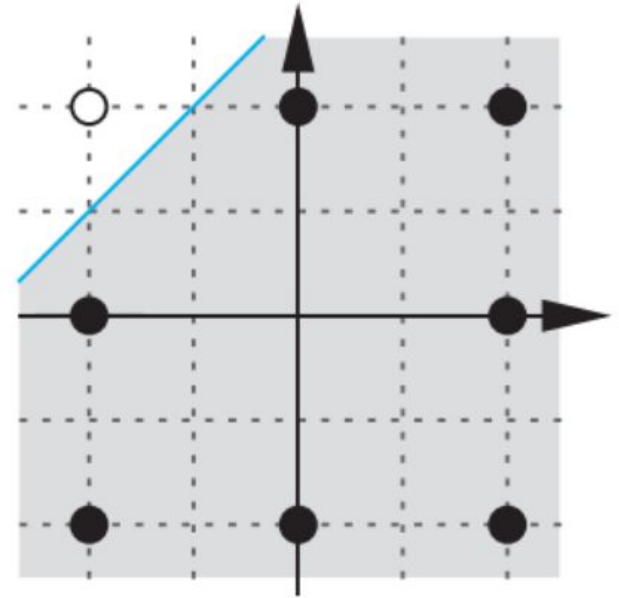
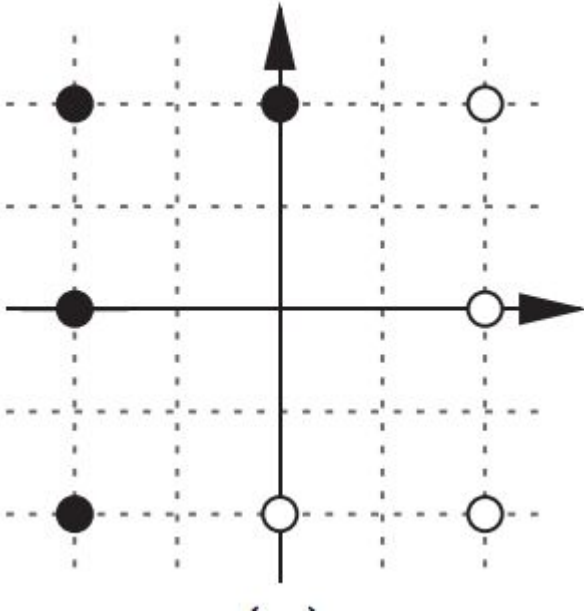
Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

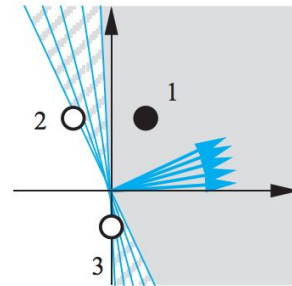
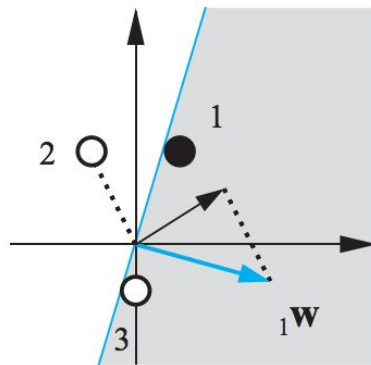
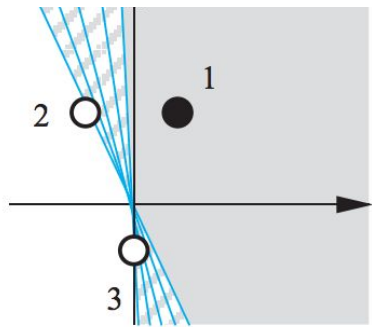
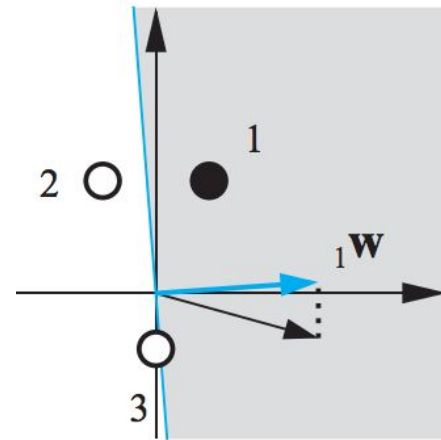
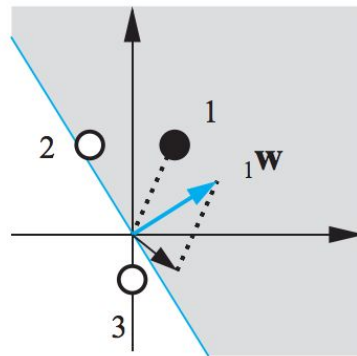
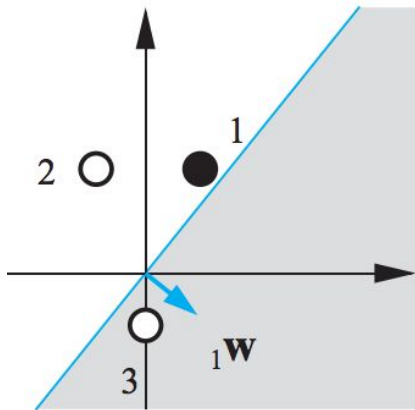
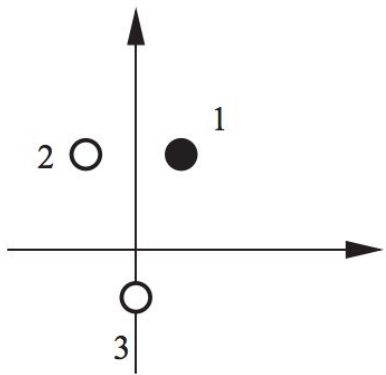


## Decision Boundary / Discriminant Function



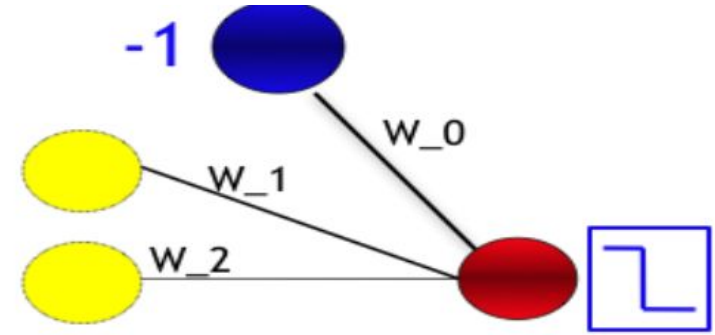
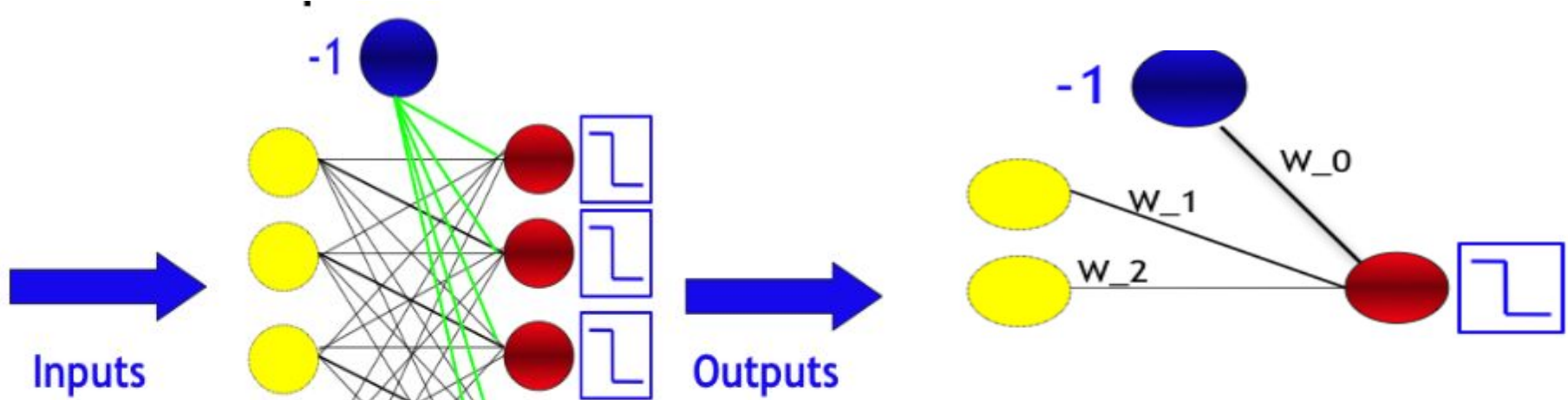
## Decision Boundary / Discriminant Function





Perceptron Learning Linear Separability(Binary)





One Row

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}.$$

$w_{32}$  is the weight that connects input node 3 to neuron 2.

input vector  $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ ,

One Row

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}.$$

weight vector  $\mathbf{w}$

$$\begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_{12288} \end{bmatrix}_{(1, 12288)} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{12288} \end{bmatrix}_{(12288, 1)}$$

weight vector  $\mathbf{w}$

input vector  $\mathbf{x}$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{12288} \end{bmatrix} \odot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{12288} \end{bmatrix} = \sum_{i=1}^{12288} w_i x_i$$

input vector  $\mathbf{x}$  : weight vector  $\mathbf{w}^T$

V  
E  
C  
T  
O  
R  
  
D  
O  
T  
  
P  
R  
O  
D  
U  
C  
T

**A · B**



# Theorem of Dot Product

$$\mathbf{A} \cdot \mathbf{B} = |\mathbf{A}| |\mathbf{B}| \cos \theta \dots\dots(1)$$

$$\cos \theta = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| |\mathbf{B}|}$$

$$\mathbf{A \cdot B}$$

$$\mathbf{A} = [2, -2, 1]$$

$$\mathbf{B} = [12, 4, -3]$$

$$\mathbf{A \cdot B} = (2)(12) + (-2)(4) + (1)(-3)$$

$$\mathbf{A \cdot B} = 24 - 8 - 3$$

$$\mathbf{A \cdot B} = 13$$

$$|\mathbf{B}| = \sqrt{(12)^2 + (4)^2 + (-3)^2}$$

$$= \sqrt{169}$$

$$= 13$$

$$|\mathbf{A}| = \sqrt{(2)^2 + (-2)^2 + (1)^2}$$

$$= \sqrt{9}$$

$$= 3$$

$$\mathbf{A} \cdot \mathbf{B}$$

$$\mathbf{A} = [2, -2, 1]$$

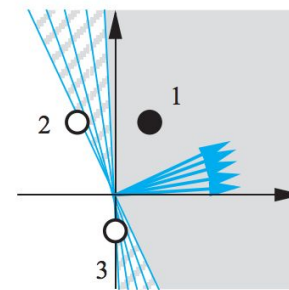
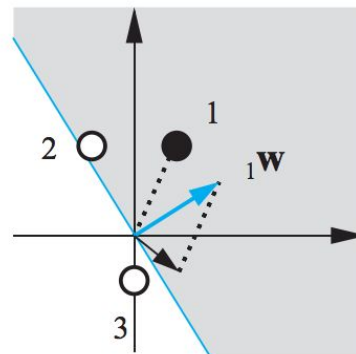
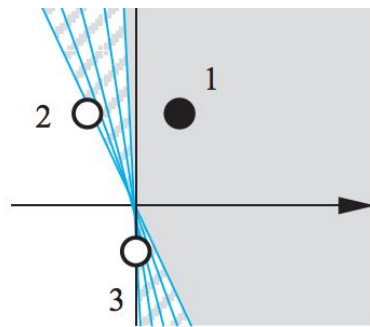
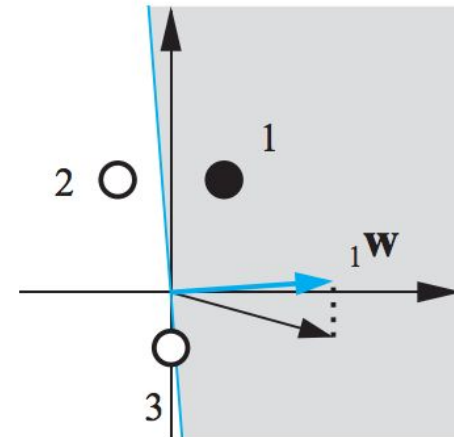
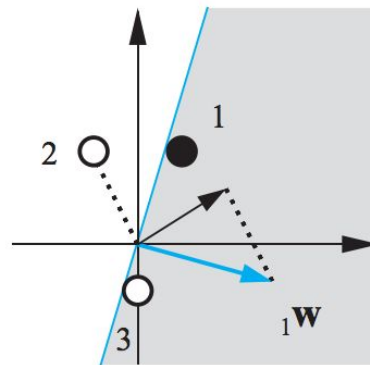
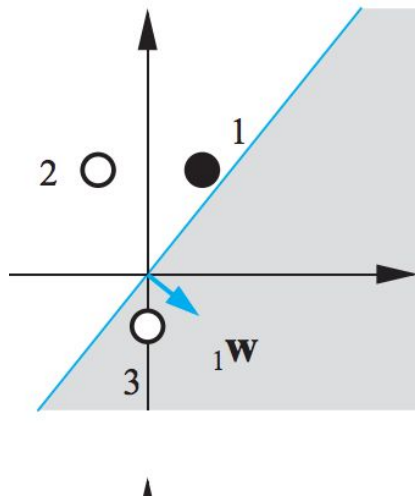
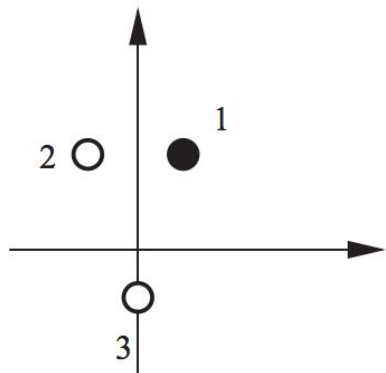
$$\cos \Theta = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| |\mathbf{B}|}$$

$$\cos \Theta = \frac{13}{3 \cdot 13}$$

$$\mathbf{B} = [12, 4, -3]$$

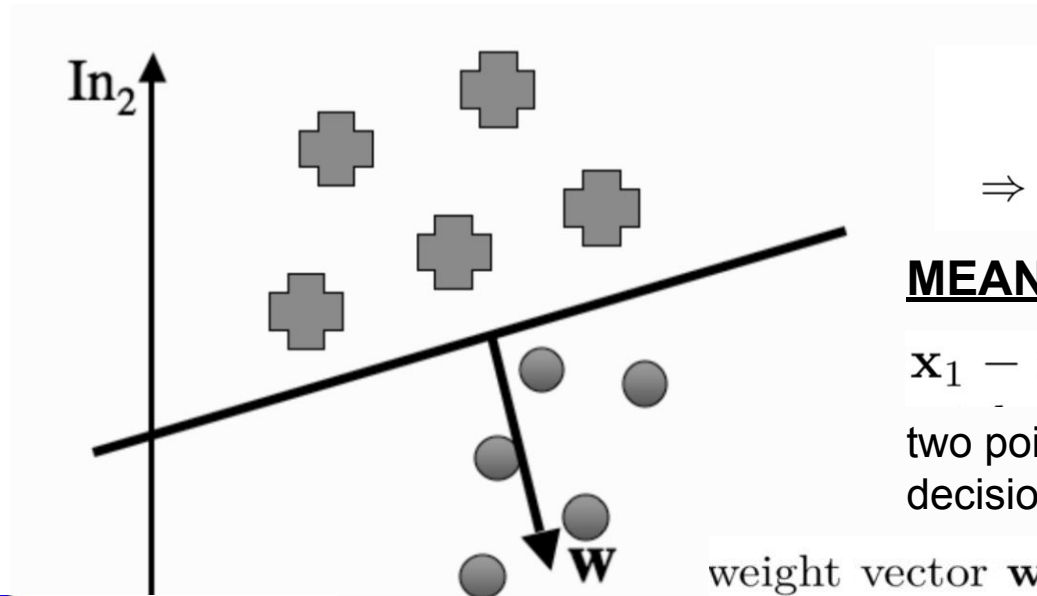
$$\Theta = \cos^{-1} \frac{1}{3}$$

$$\Theta = 70^\circ$$



Perceptron Learning Linear Separability(Binary)



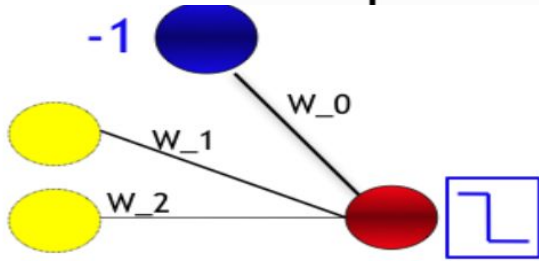


$$\begin{aligned} \mathbf{x}_1 \cdot \mathbf{w}^T &= \mathbf{x}_2 \cdot \mathbf{w}^T \\ \Rightarrow (\mathbf{x}_1 - \mathbf{x}_2) \cdot \mathbf{w}^T &= 0. \end{aligned}$$

### MEANING:

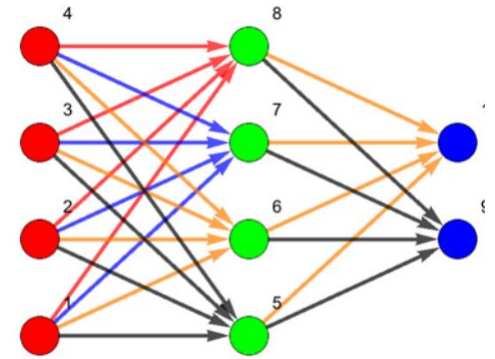
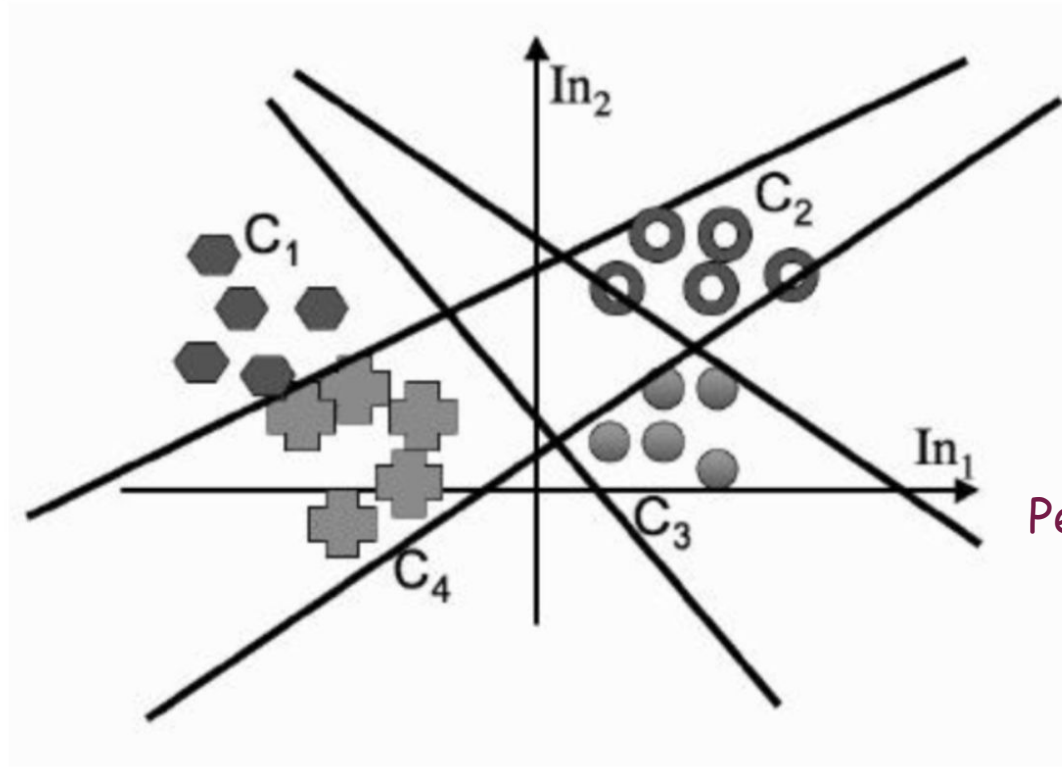
$\mathbf{x}_1 - \mathbf{x}_2$  Is a straight line between two points that lie in the decision boundary.

weight vector  $\mathbf{w}^T$  Is **perpendicular** to that.



Perceptron Learning Linear Separability(Binary) - Output of a Perceptron with **One** Neuron

A decision boundary separating two classes of data.



Perceptron Learning Linear  
Separability(Binary) -  
Output of a Perceptron  
with **Four** Neurons

Different decision boundaries computed by a Perceptron with four neurons.

# Perceptron Convergence Theorem

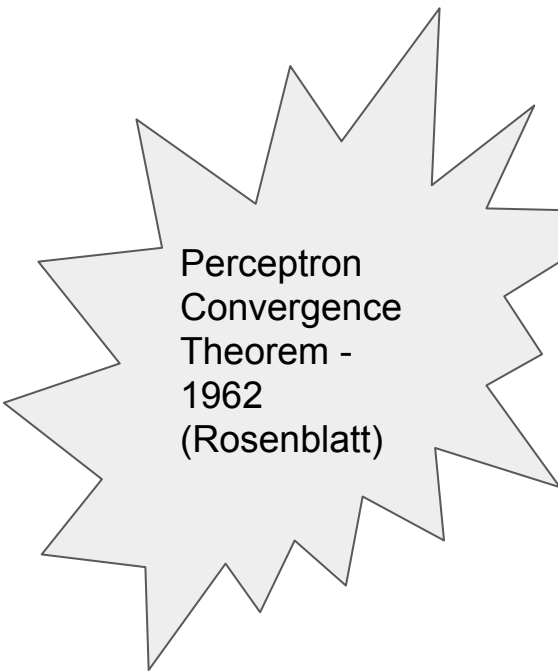
## Perceptron Convergence and Cycling Theorems

---

- **Perceptron convergence theorem:** If the data is linearly separable and therefore a set of weights exist that are consistent with the data, then the Perceptron algorithm will eventually converge to a consistent set of weights.
- **Perceptron cycling theorem:** If the data is not linearly separable, the Perceptron algorithm will eventually repeat a set of weights and threshold at the end of some epoch and therefore enter an infinite loop.
  - By checking for repeated weights+threshold, one can guarantee termination with either a positive or negative result.

(1958)  
F. Rosenblatt

**The perceptron: a probabilistic model  
for information storage and organization in the brain**  
*Psychological Review* 65:386-408



Perceptron  
Convergence  
Theorem -  
1962  
(Rosenblatt)

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

**Perceptron convergence theorem:** If the data is linearly separable and therefore a set of weights exist that are consistent with the data, then the Perceptron algorithm will eventually converge to a consistent set of weights.

In 1969

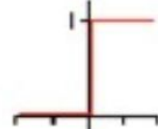
Minsky & Papert showed such weights exist if and only if the problem is *linearly separable*

**Given:** A linearly contained training set  $X'$  and any initial weight vector  $W_1$ .

Let  $S_W$  be the weight vector sequence generated in response to presentation of a training sequence  $S_X$  upon application of Perceptron learning law. Then for some finite index  $k_0$  we have:  $W_{k_0} = W_{k_0+1} = W_{k_0+2} = \cdot \quad \cdot \quad \cdot = W_S$  as a solution vector.



y 1.0



-1

0.4576

u



0.0118

w2

0.3694

w1

0.0409

1.0

x2

x1

0.0

Current Error: 1.0

Squared Error: 0.0

☐ Solve XOR

x2

x1

y

0

0

1

0

1

0

1

0

0

1

1

0

Learning rate:

0.05

Number of iterations:

20

Error threshold:

0.1

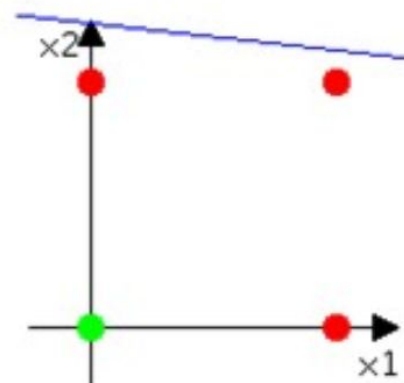
Train

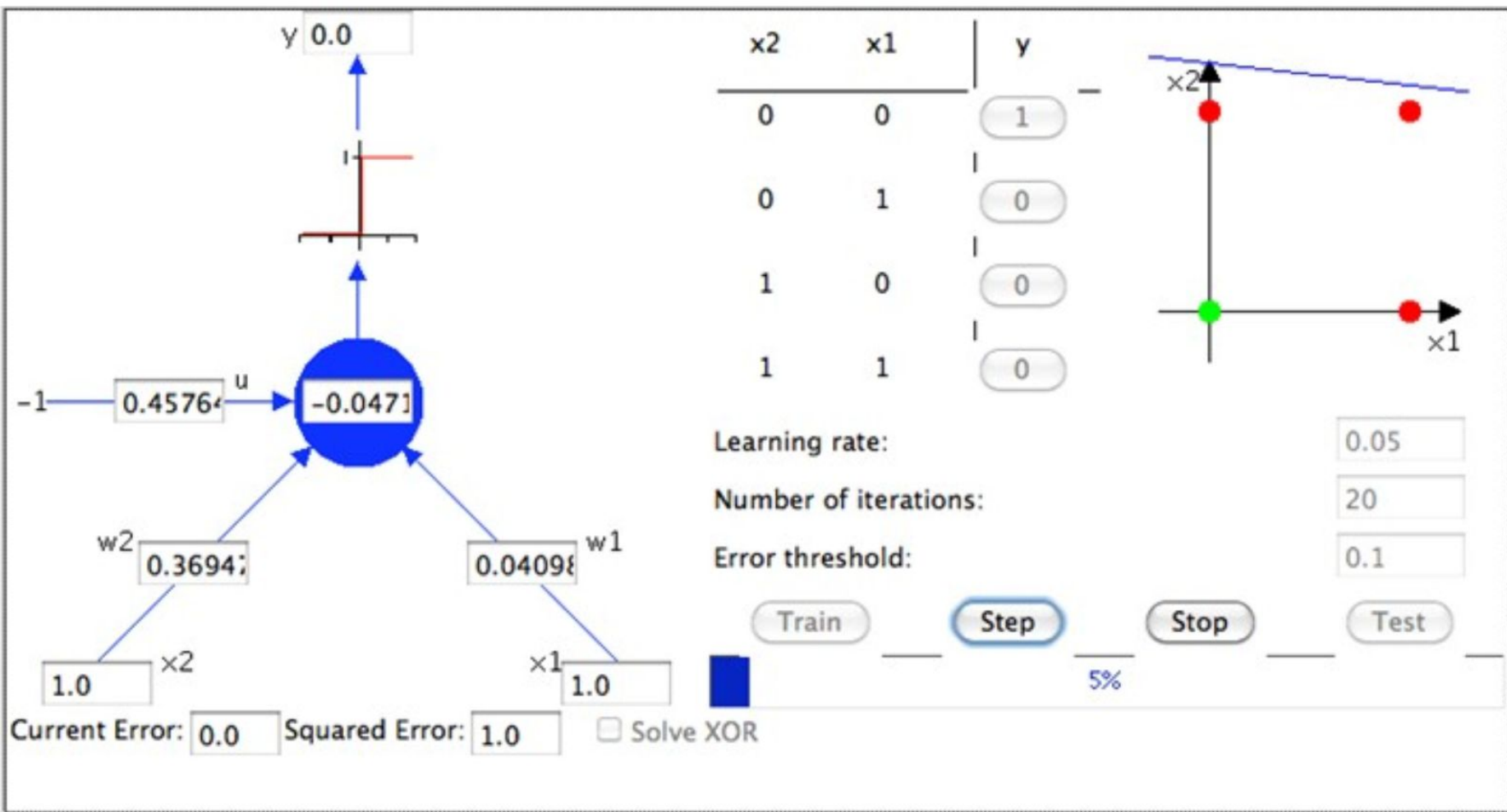
Step

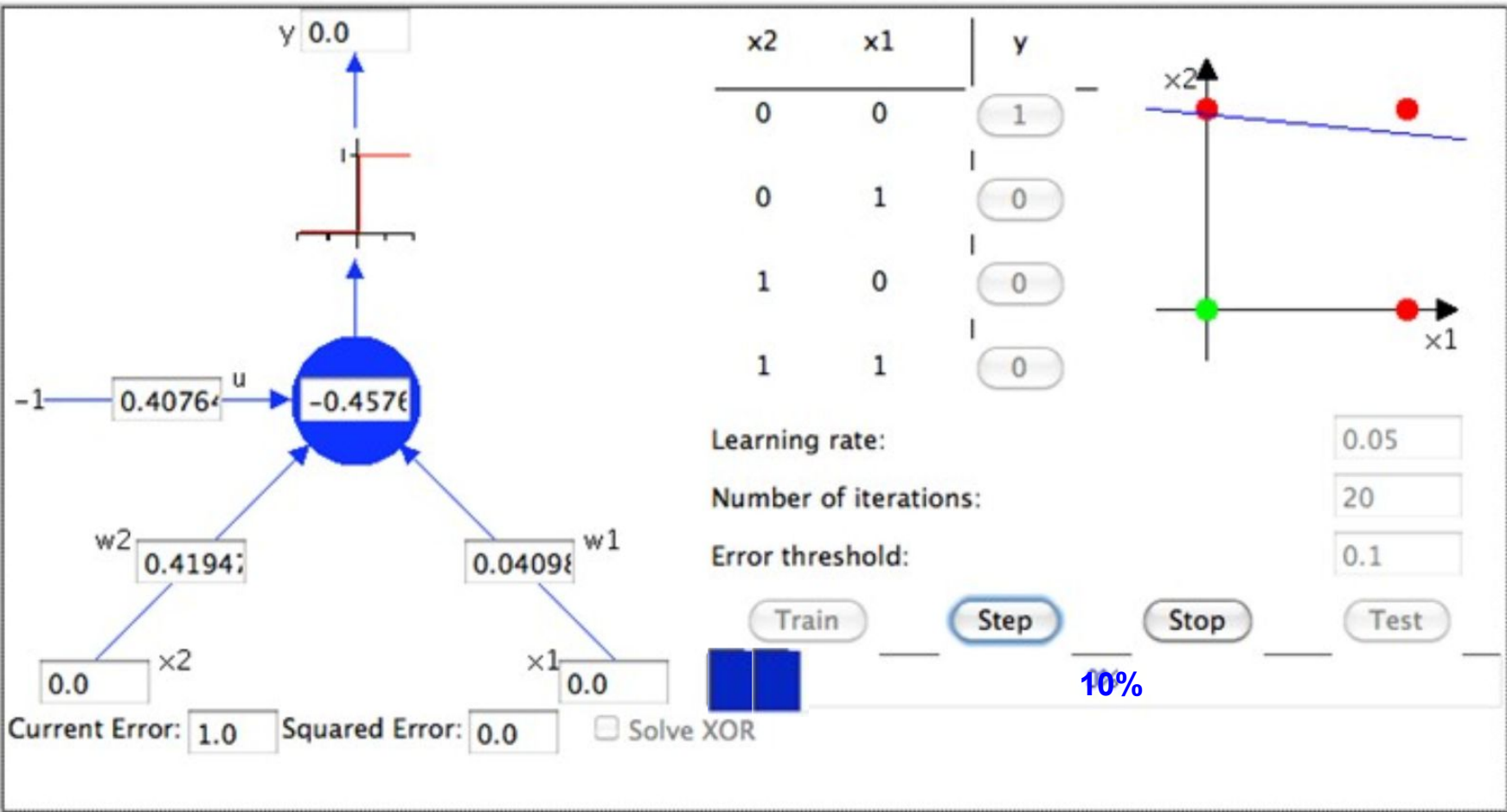
Stop

Test

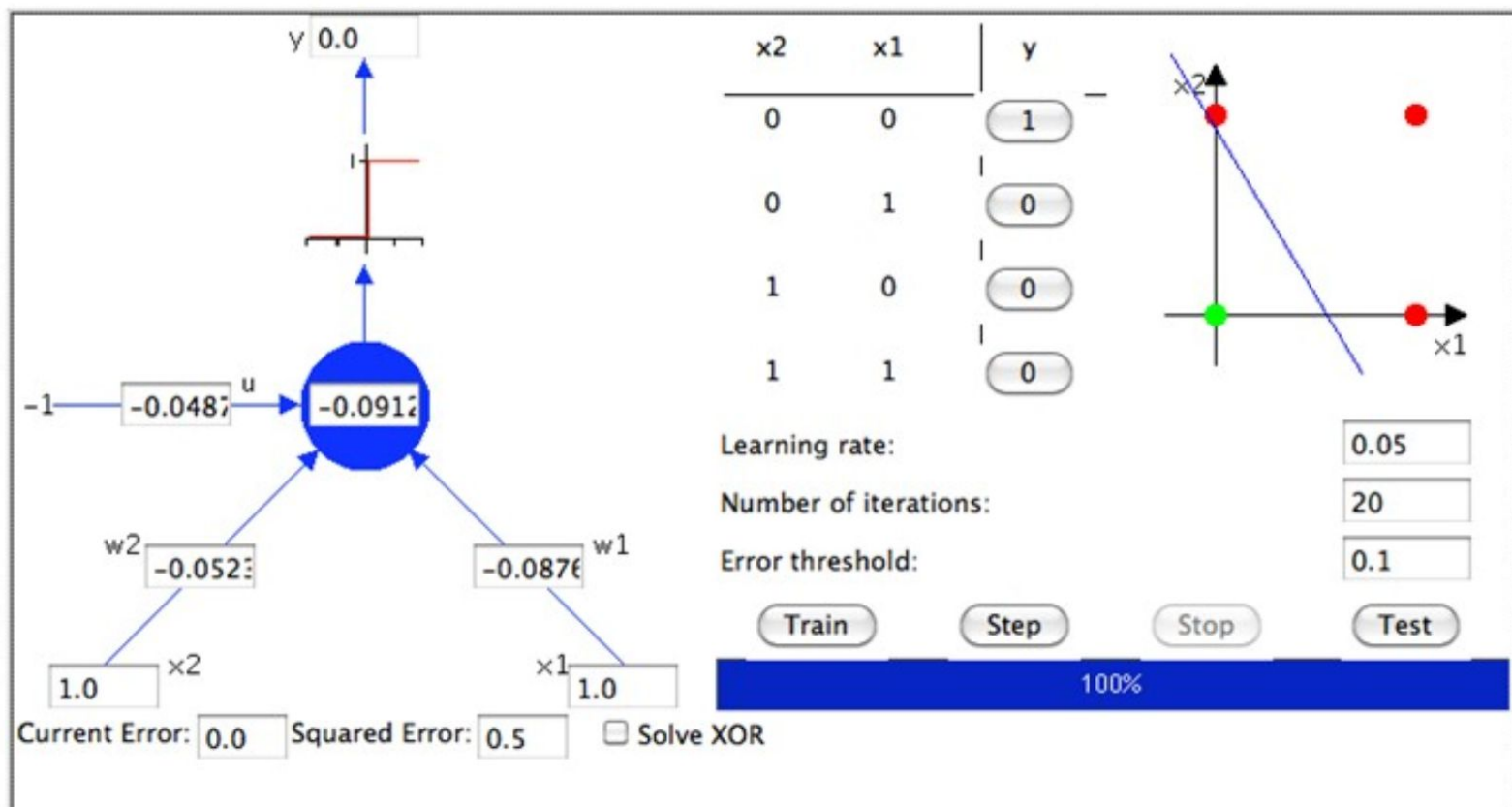
0%







# After 20 iterations



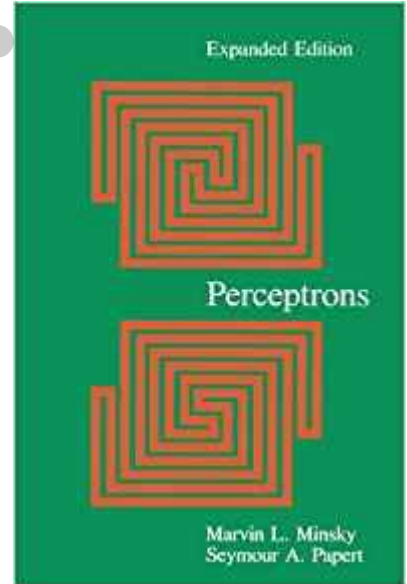


# Minsky and Papert

*Effectively killed neural network research for about 20 years.*

At that point they did not know the problems they showed as “unsolvable” in the book - like XOR Linearly Separable can be easily solved through addition of multiple layers into the perceptron - as it stands today!

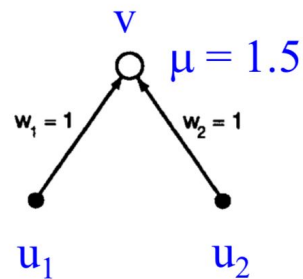
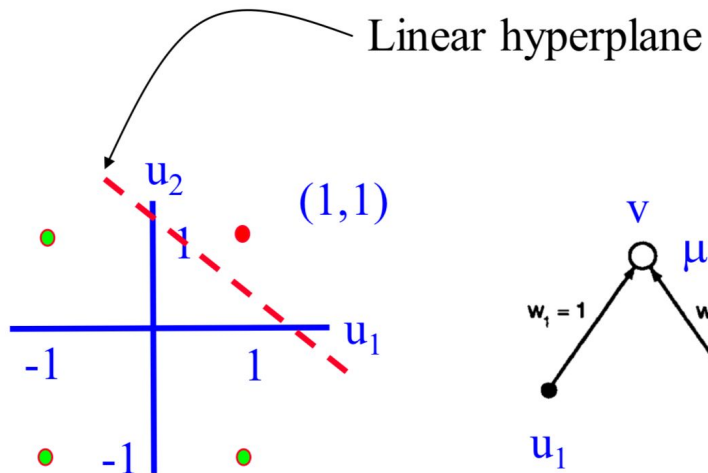
## Their Book Perceptrons (1969)



# Linear Separability

Example: AND is linearly separable

$u_1$	$u_2$	AND
-1	-1	-1
1	-1	-1
-1	1	-1
1	1	1

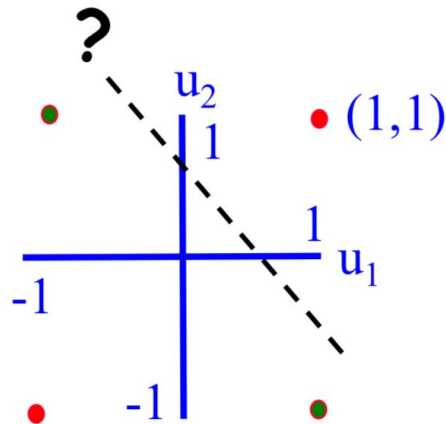


$$v = 1 \text{ iff } u_1 + u_2 - 1.5 > 0$$

Similarly for OR and NOT

# What about the XOR function?

$u_1$	$u_2$	XOR
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1



Can a perceptron separate the +1  
outputs from the -1 outputs?

XOR Problem - Linear Separability

# XOR Problem - Linear Separability

$In_1$	$In_2$	$In_3$	Output
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	1

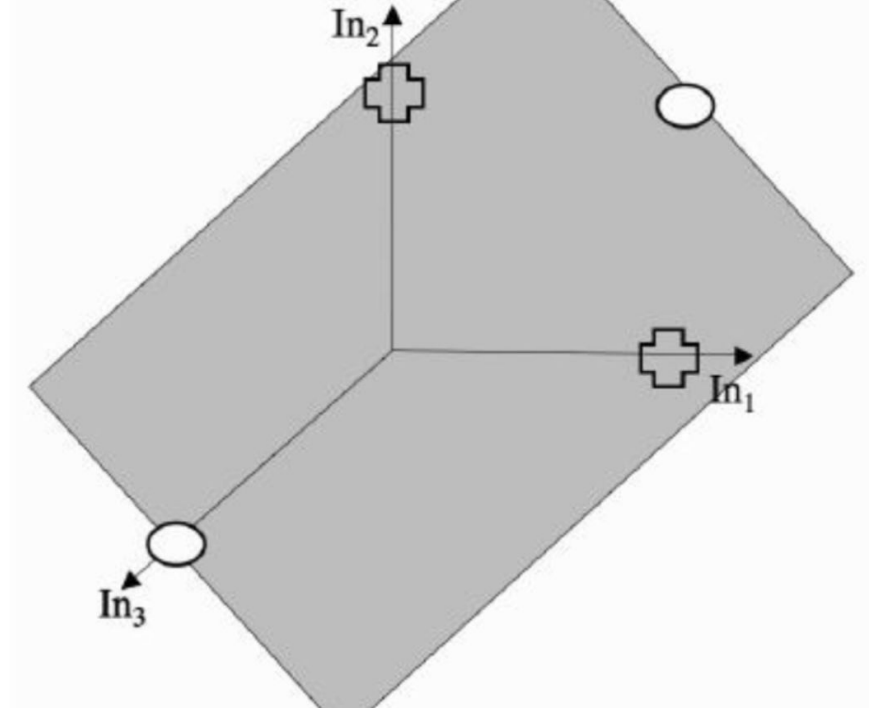


FIGURE 3.10 A decision boundary (the shaded plane) solving the XOR problem in 3D with the crosses below the surface and the circles above it.

(Image from Stephan Marsland Textbook)



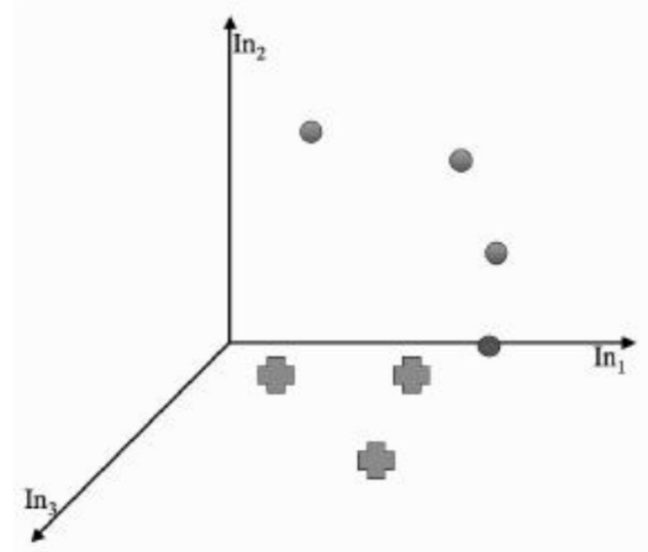
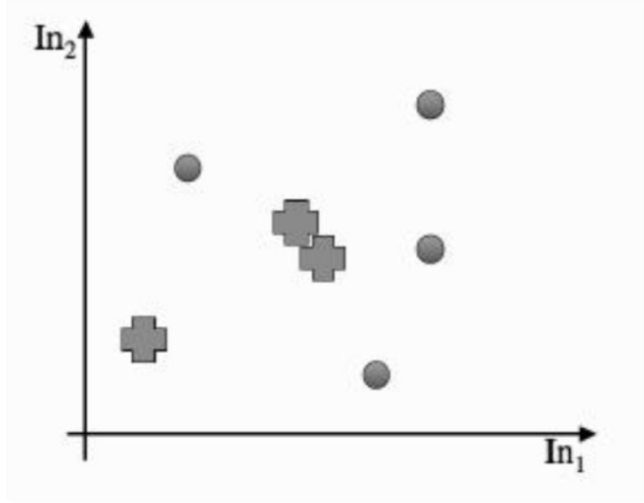
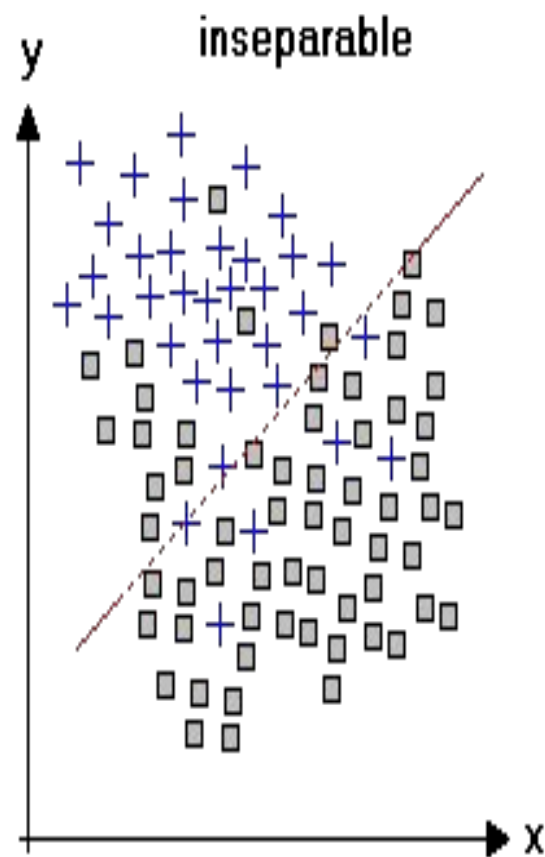
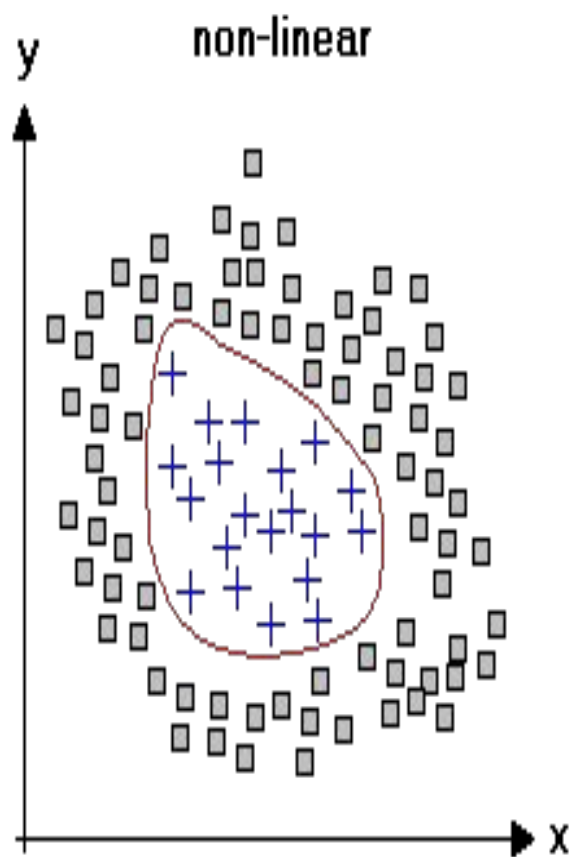
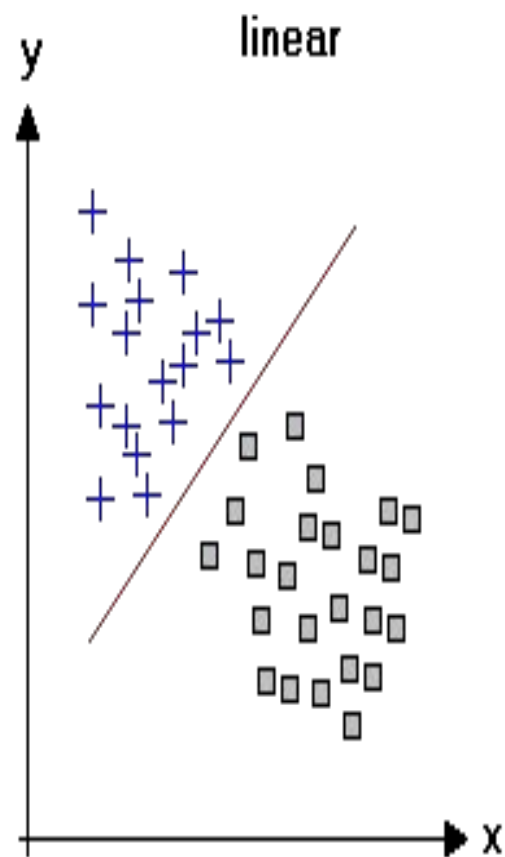
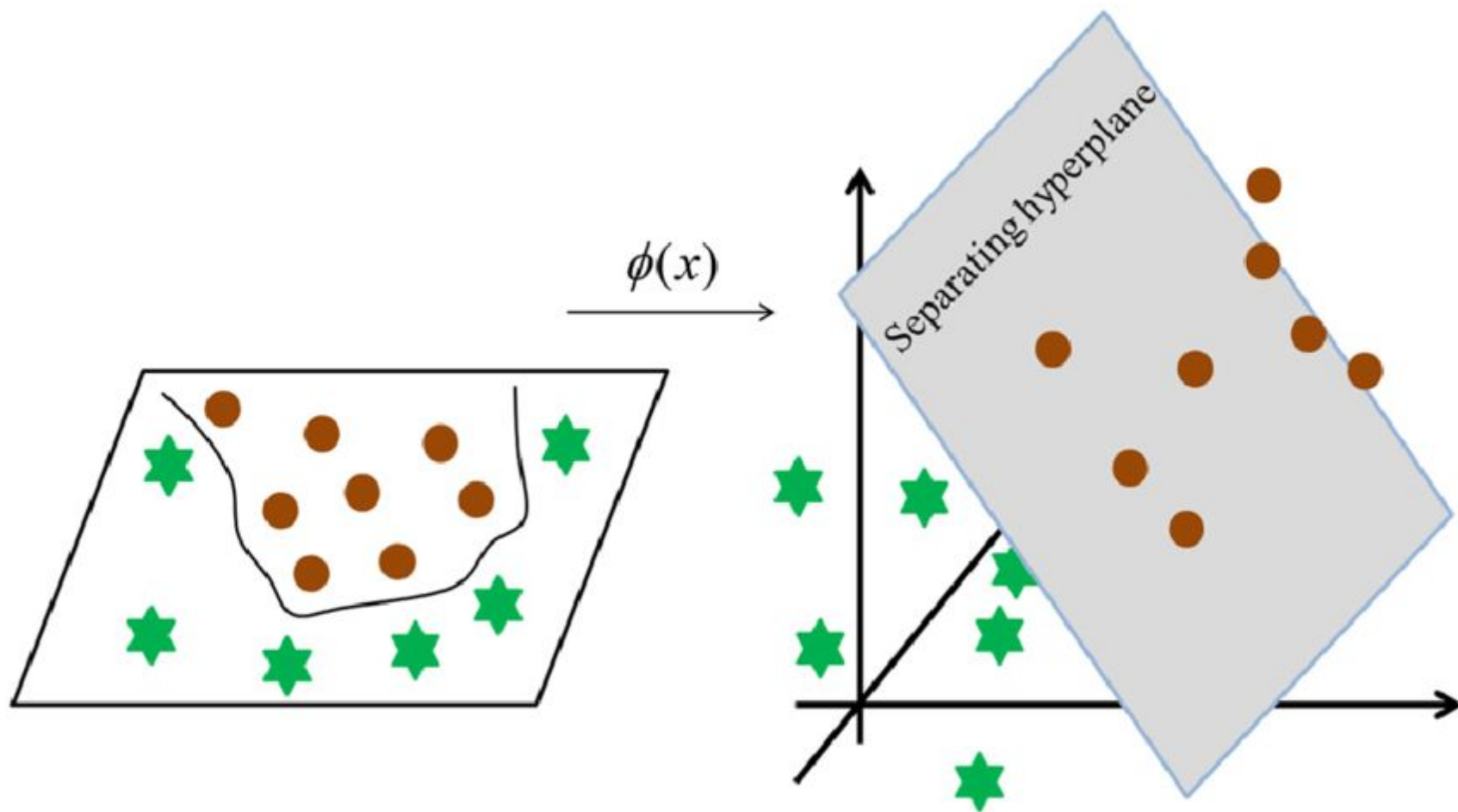


FIGURE 3.11 *Left:* Non-separable 2D dataset. *Right:* The same dataset with third coordinate  $x_1 \times x_2$ , which makes it separable.

(Image from Stephan Marsland Textbook)





Complex in low dimensions

Simple in higher dimensions