

# COMP9444

# Neural Networks and Deep Learning

## 2. Perceptrons

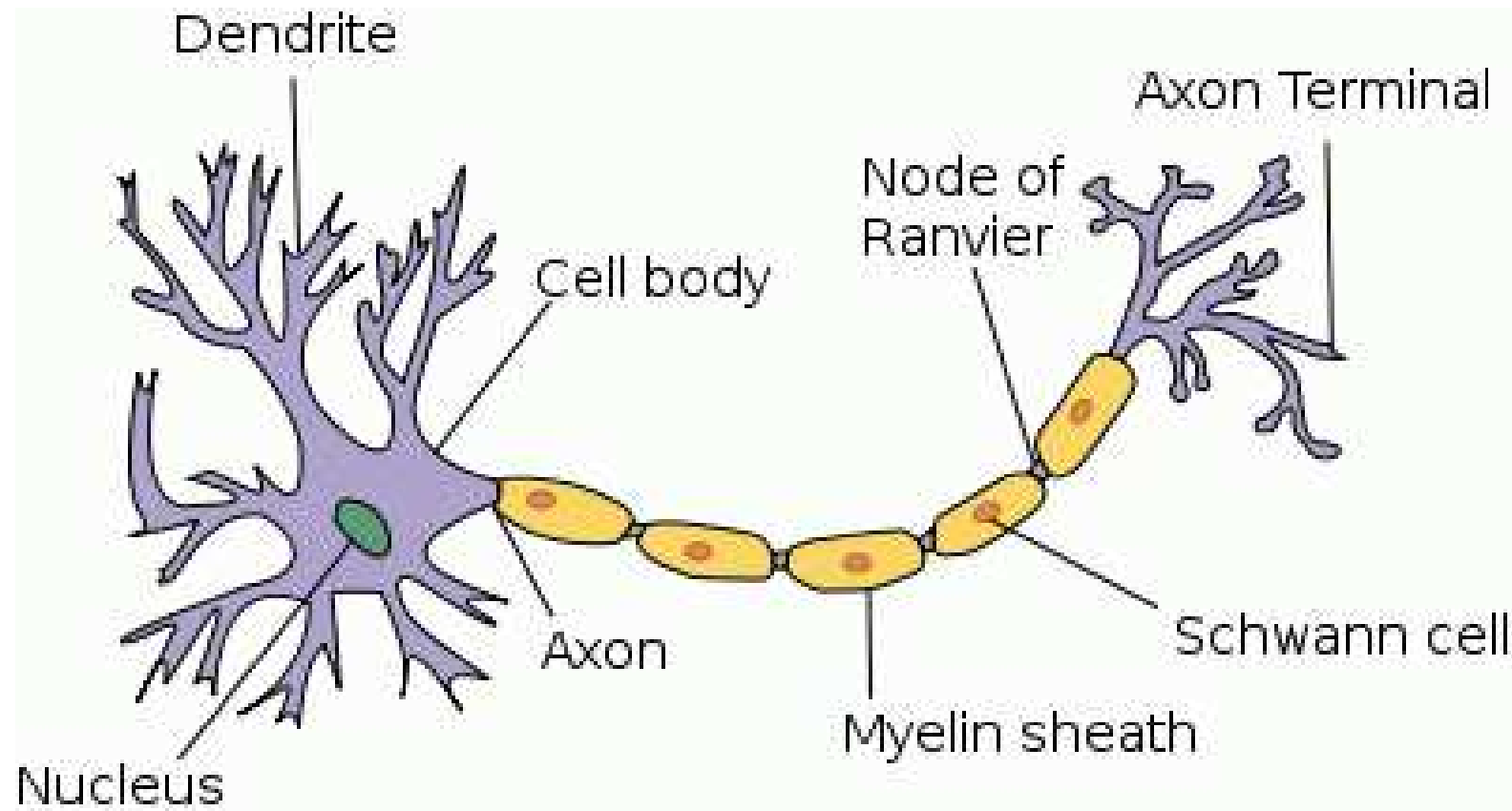
# Outline

---

- Neurons – Biological and Artificial
- Perceptron Learning
- Linear Separability
- Multi-Layer Networks

# Structure of a Typical Neuron

---



# Biological Neurons

---

The brain is made up of **neurons** (nerve cells) which have

- a cell body (soma)
- **dendrites** (inputs)
- an **axon** (outputs)
- **synapses** (connections between cells)

Synapses can be **excitatory** or **inhibitory** and may change over time.

When the inputs reach some threshold an **action potential** (electrical pulse) is sent along the axon to the outputs.

# Artificial Neural Networks

---

(Artificial) Neural Networks are made up of nodes which have

- inputs edges, each with some **weight**
- outputs edges (with **weights**)
- an **activation level** (a function of the inputs)

Weights can be positive or negative and may change over time (learning).

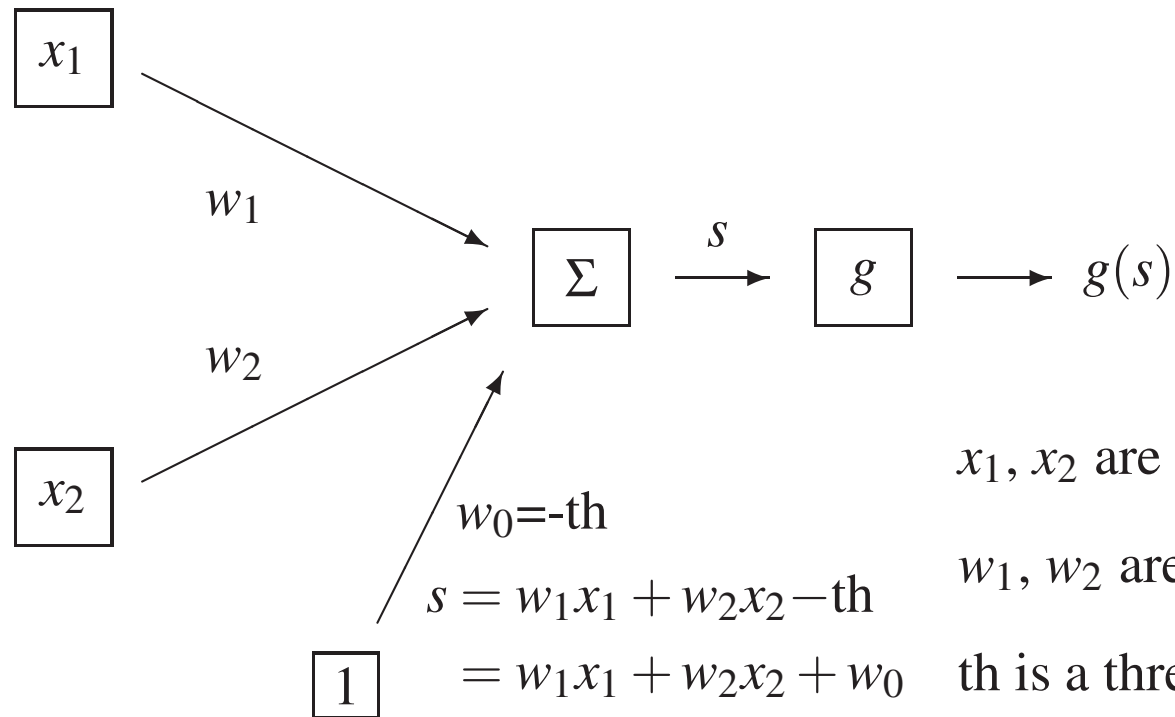
The **input function** is the weighted sum of the activation levels of inputs.

The activation level is a non-linear **transfer** function  $g$  of this input:

$$\text{activation}_i = g(s_i) = g\left(\sum_j w_{ij}x_j\right)$$

Some nodes are inputs (sensing), some are outputs (action)

# McCulloch & Pitts Model of a Single Neuron



$x_1, x_2$  are inputs

$w_1, w_2$  are synaptic weights

th is a threshold

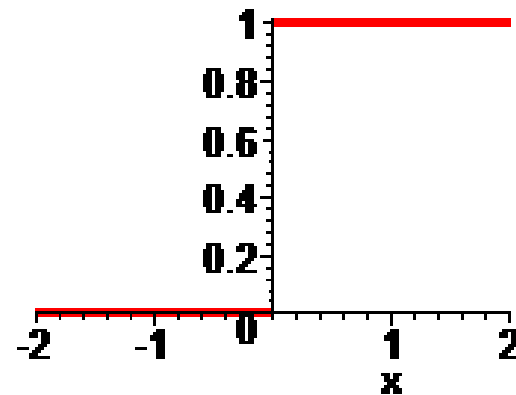
$w_0$  is a **bias** weight

$g$  is transfer function

# Transfer function

---

Originally, a (discontinuous) step function was used for the transfer function:



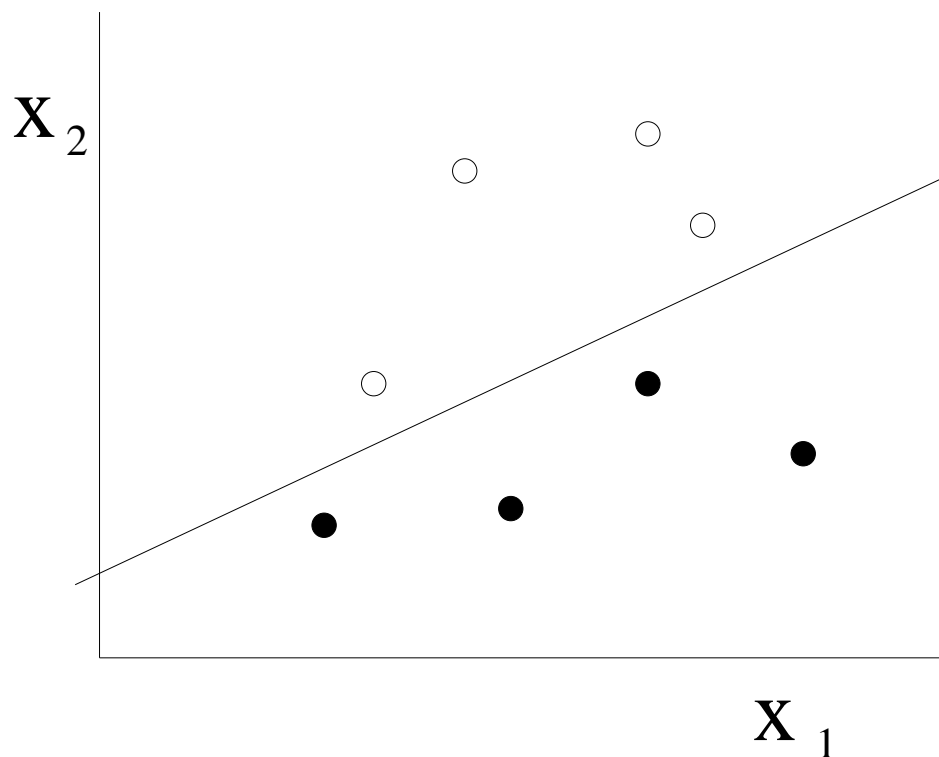
$$g(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ 0, & \text{if } s < 0 \end{cases}$$

(Later, other transfer functions were introduced, which are continuous and smooth)

# Linear Separability

---

Question: what kind of functions can a perceptron compute?



Answer: linearly separable functions



# Linear Separability

---

Examples of linearly separable functions:

AND  $w_1 = w_2 = 1.0, \quad w_0 = -1.5$

OR  $w_1 = w_2 = 1.0, \quad w_0 = -0.5$

NOR  $w_1 = w_2 = -1.0, \quad w_0 = 0.5$

Q: How can we train it to learn a new function?

# Rosenblatt Perceptron

---



# Perceptron Learning Rule

Adjust the weights as each input is presented.

recall:  $s = w_1x_1 + w_2x_2 + w_0$

if  $g(s) = 0$  but should be 1,

$$w_k \leftarrow w_k + \eta x_k$$

$$w_0 \leftarrow w_0 + \eta$$

$$\text{so } s \leftarrow s + \eta \left(1 + \sum_k x_k^2\right)$$

if  $g(s) = 1$  but should be 0,

$$w_k \leftarrow w_k - \eta x_k$$

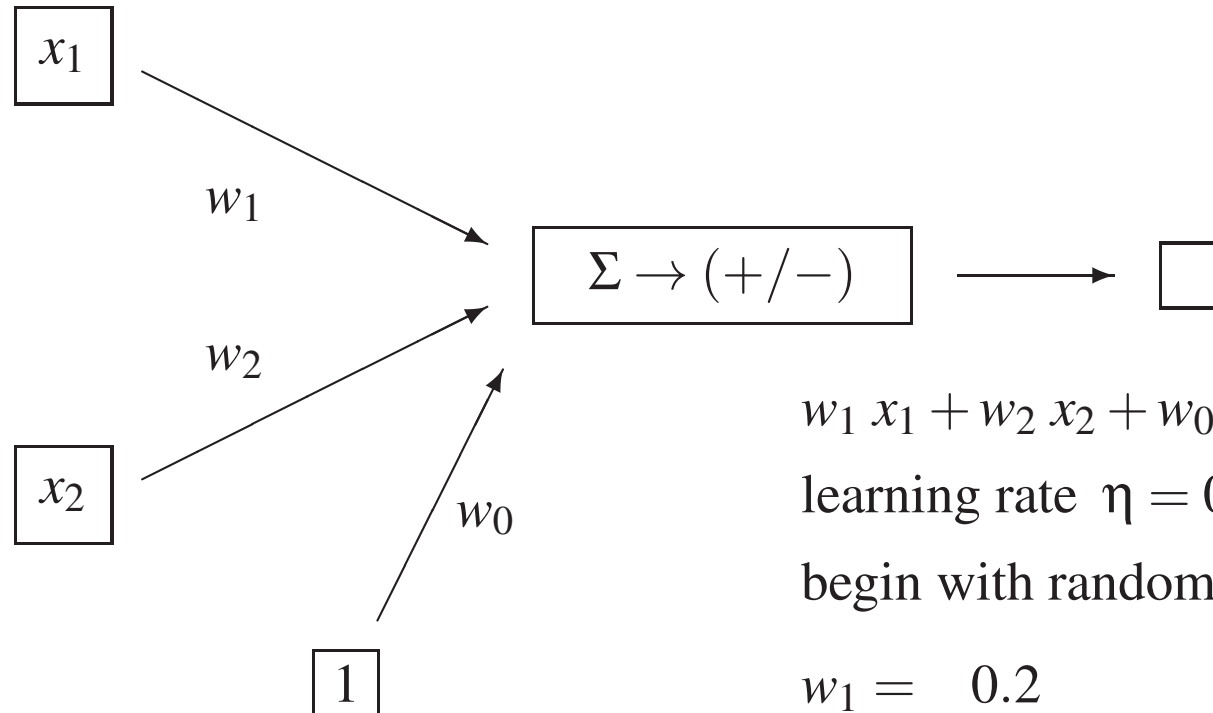
$$w_0 \leftarrow w_0 - \eta$$

$$\text{so } s \leftarrow s - \eta \left(1 + \sum_k x_k^2\right)$$

otherwise, weights are unchanged. ( $\eta > 0$  is called the **learning rate**)

**Theorem:** This will eventually learn to classify the data correctly,  
as long as they are linearly separable.

# Perceptron Learning Example



$$w_1 x_1 + w_2 x_2 + w_0 > 0$$

learning rate  $\eta = 0.1$

begin with random weights

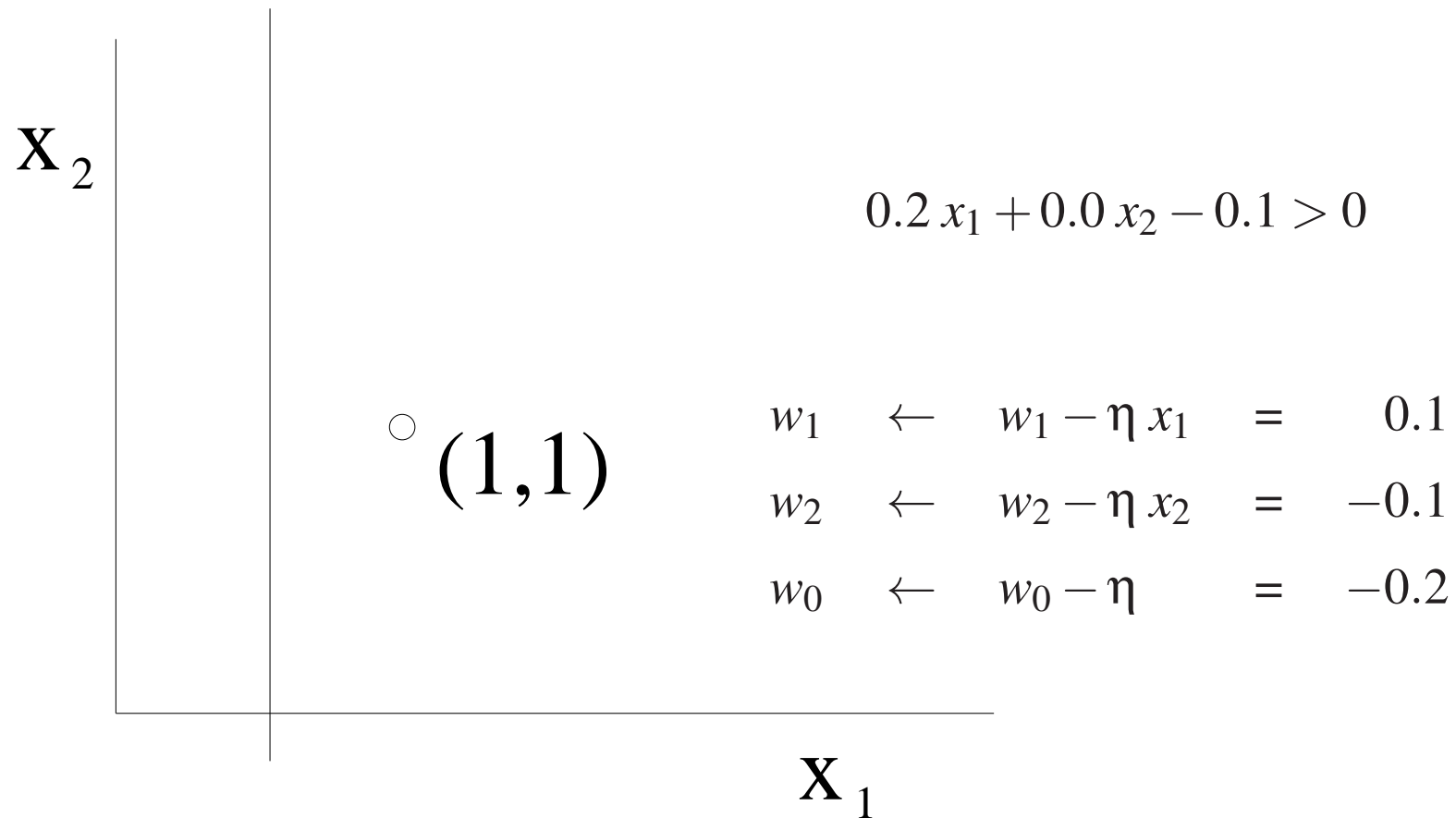
$$w_1 = 0.2$$

$$w_2 = 0.0$$

$$w_0 = -0.1$$

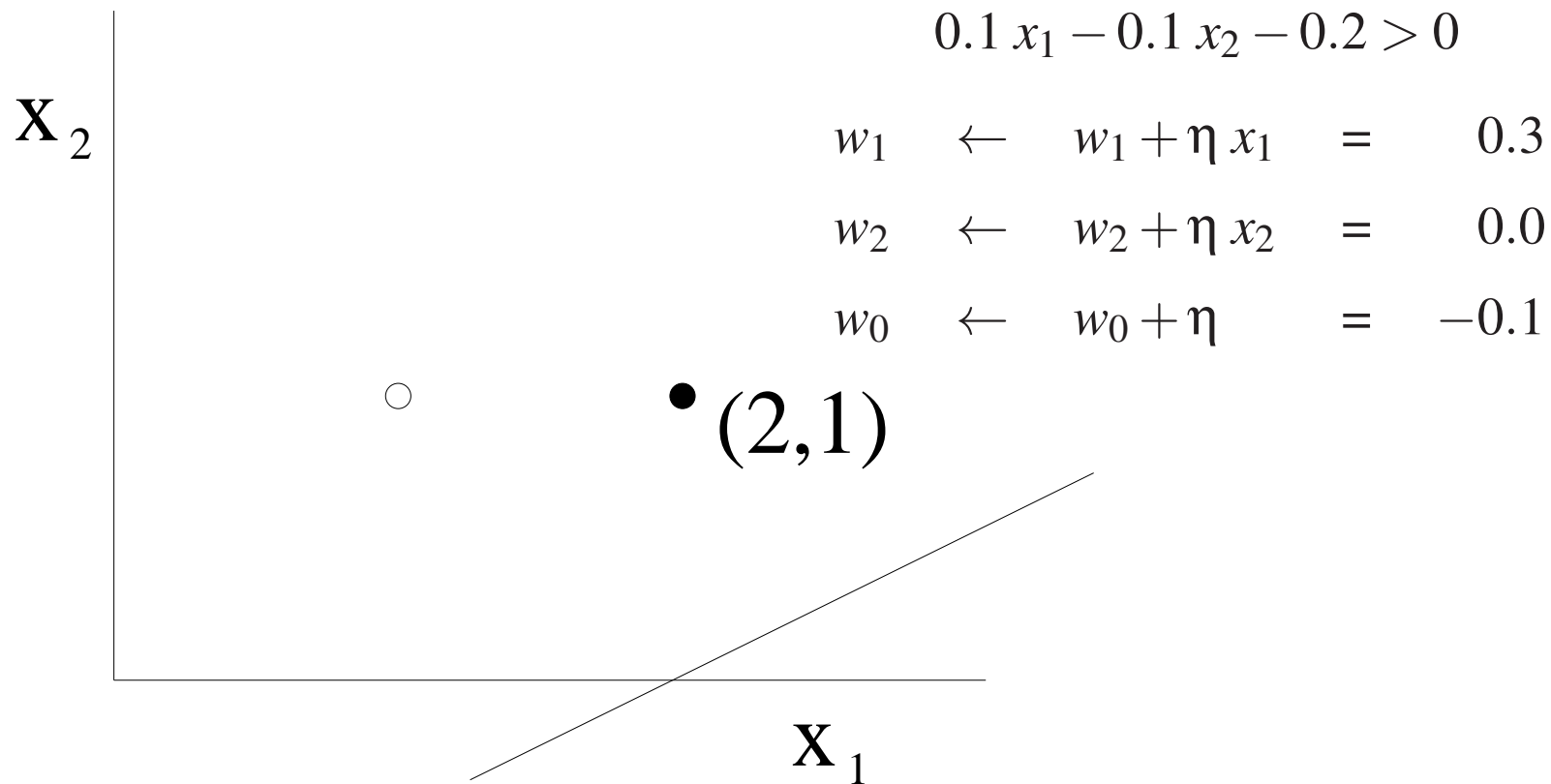
# Training Step 1

---

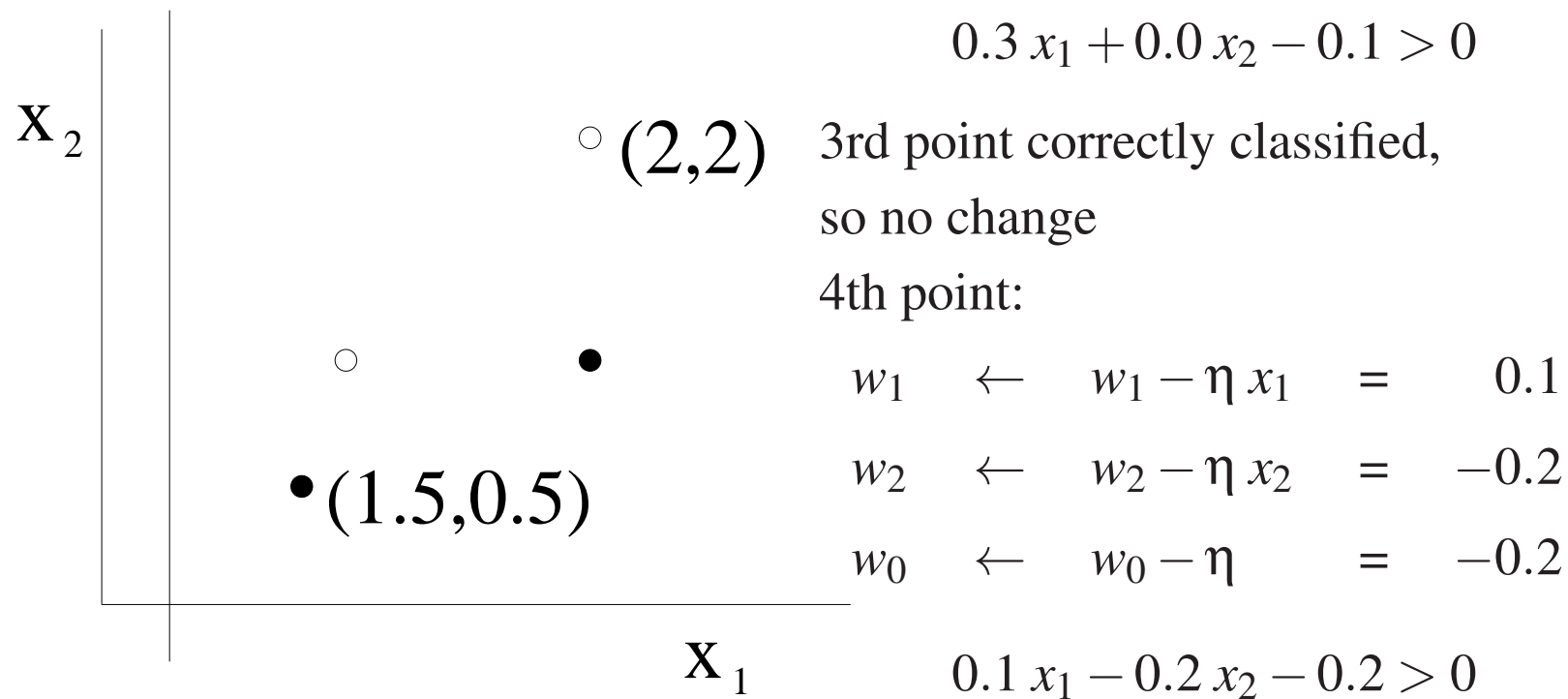


## Training Step 2

---

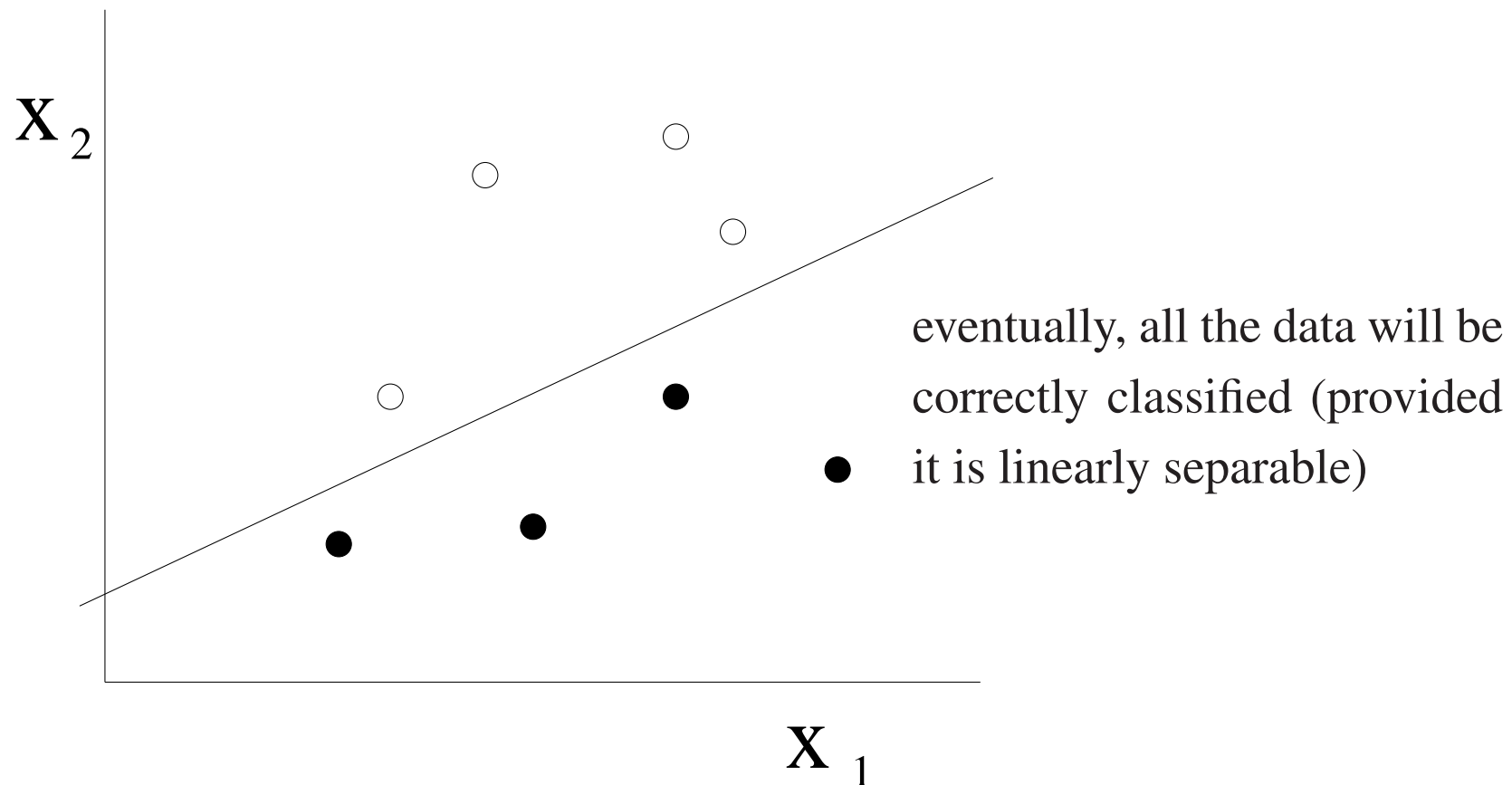


## Training Step 3



# Final Outcome

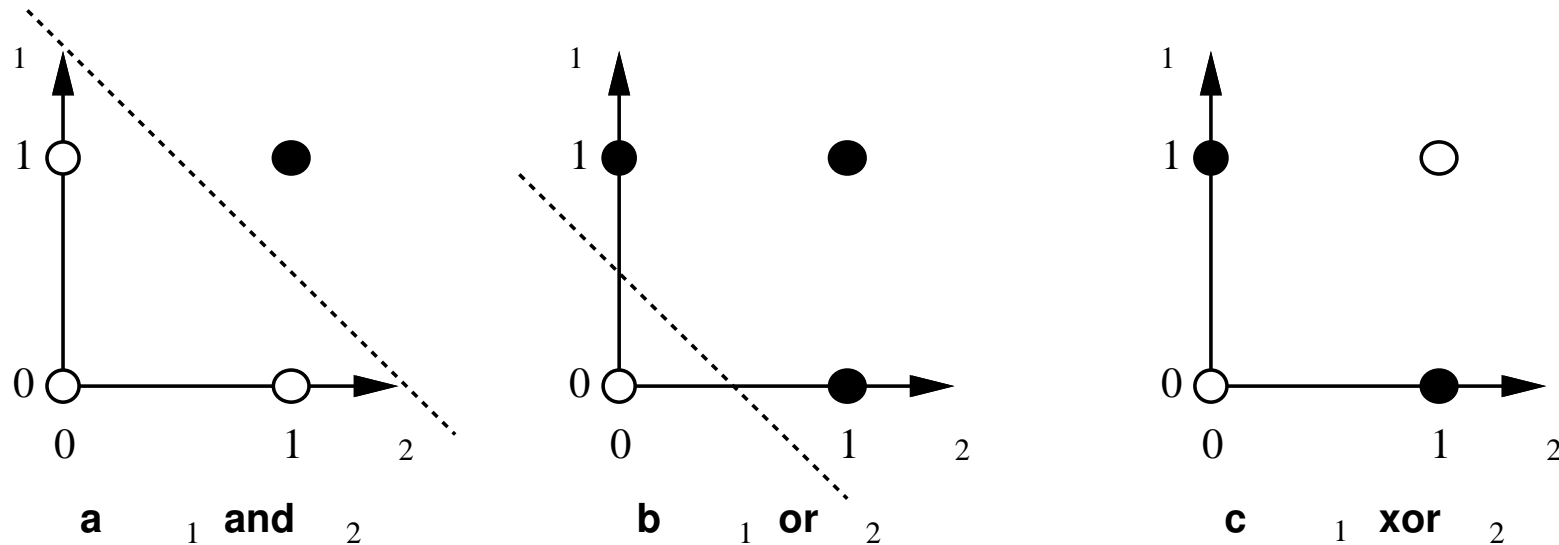
---





# Limitations of Perceptrons

Problem: many useful functions are not linearly separable (e.g. XOR)



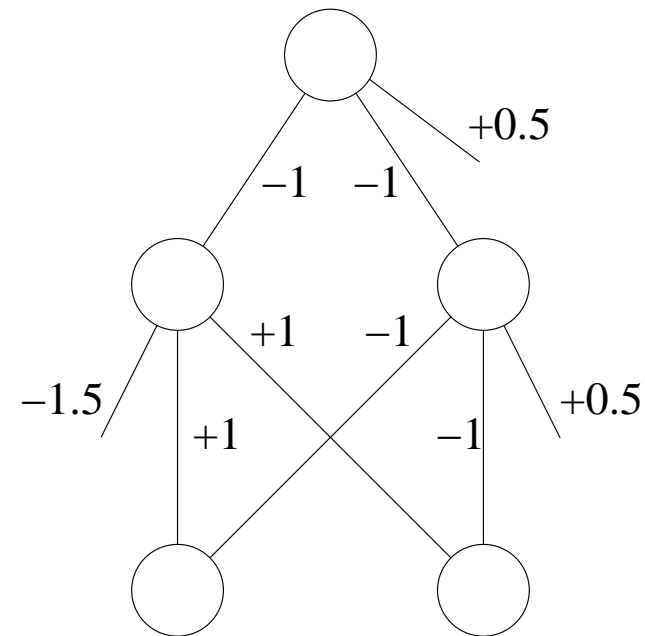
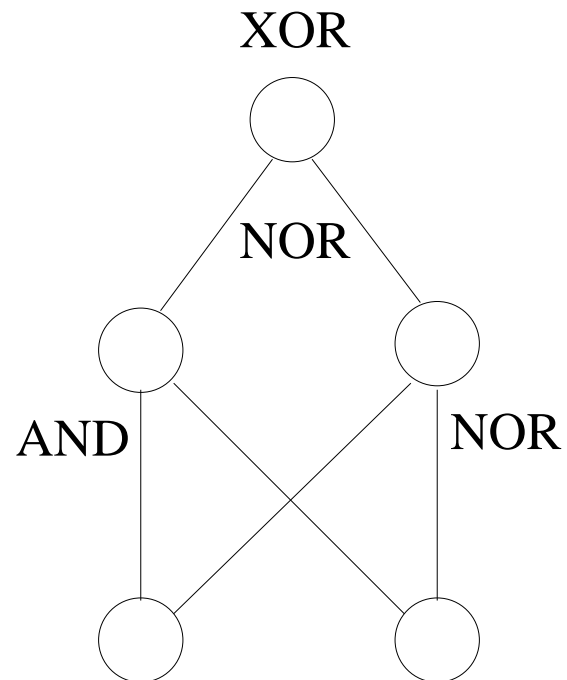
Possible solution:

$x_1 \text{ XOR } x_2$  can be written as:  $(x_1 \text{ AND } x_2) \text{ NOR } (x_1 \text{ NOR } x_2)$

Recall that AND, OR and NOR can be implemented by perceptrons.

# Multi-Layer Neural Networks

---



Problem: How can we train it to learn a new function? (credit assignment)

# Historical Context

---

In 1969, Minsky and Papert published a book highlighting the limitations of Perceptrons, and lobbied various funding agencies to redirect funding away from neural network research, preferring instead logic-based methods such as expert systems.

It was known as far back as the 1960's that any given logical function could be implemented in a 2-layer neural network with step function activations. But, the the question of how to learn the weights of a multi-layer neural network based on training examples remained an open problem. The solution, which we describe in the next section, was found in 1976 by Paul Werbos, but did not become widely known until it was rediscovered in 1986 by Rumelhart, Hinton and Williams.