Tufts
UNIVERSITY

# ARTIFICIAL NEURAL NETWORKS 1

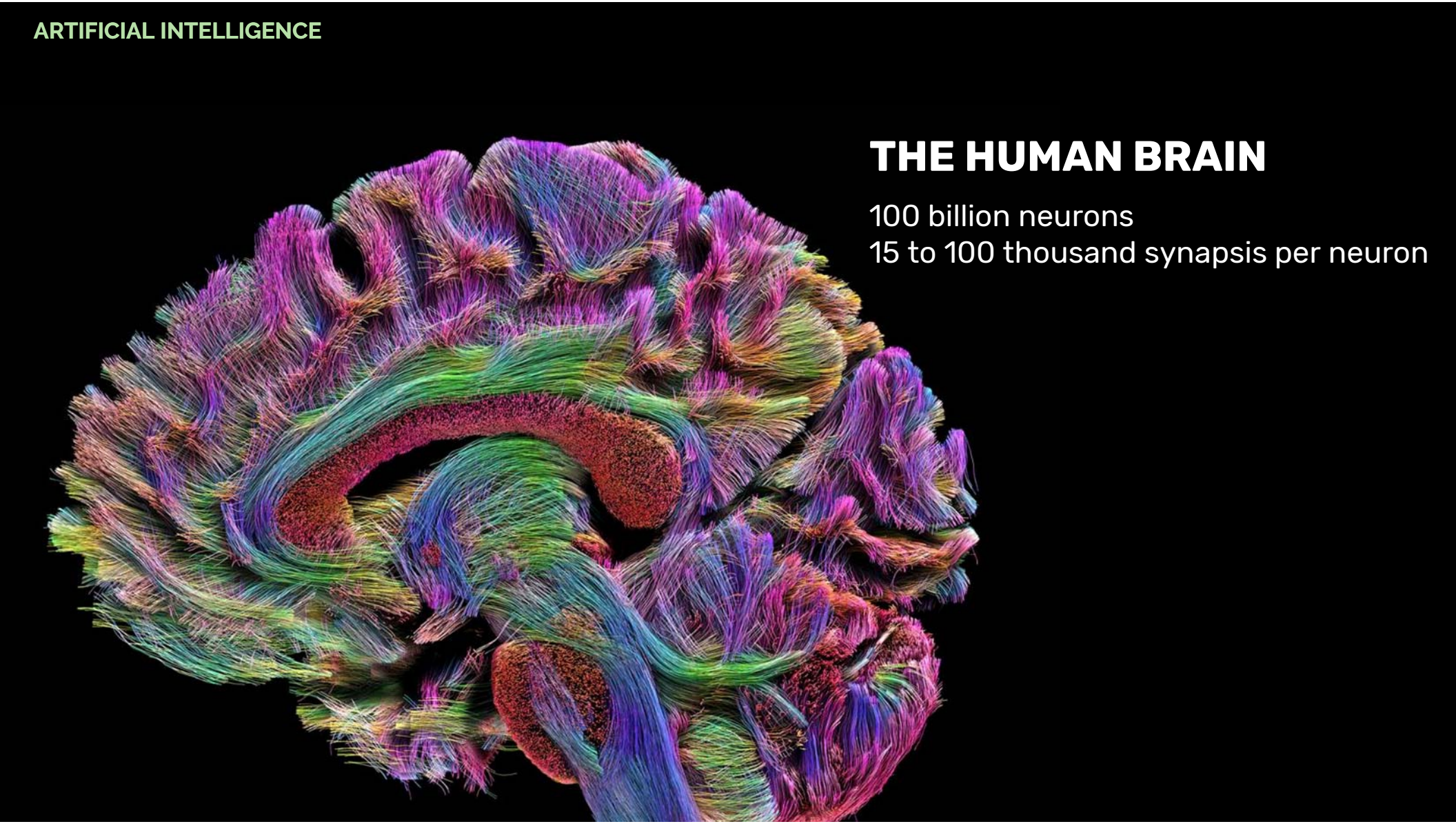
Fabrizio Santini | Comp 131

VERSION 1.2

# TODAY ON AI

- The neuron
- Perceptron
- The back-propagation algorithm
- Forward propagation
- Backward propagation
- Questions?

ARTIFICIAL INTELLIGENCE



# THE HUMAN BRAIN

100 billion neurons
15 to 100 thousand synapsis per neuron

ARTIFICIAL INTELLIGENCE

| OPERATION | COMPUTER | BRAIN |
|---|---|---|
| Type of operations | Digital | Analog |
| Memory storing / recall | Location-addressable memory | Content-addressable memory |
| Architectural paradigm | Modular and serial | Massively parallel |
| Synchronization and clock | Synchronized and fixed processing speed | Asynchronous and No clock |
| Hardware / Software | Distinct hardware and software | No distinction |
| Basic elements | Transistors and logic gates | Complex synapsis |
| Processing and memory | Dedicated components | Neurons implement processing and memory |
| Architecture | Fixed and pre-designed | The brain is a self-organizing system |
| Embodiment | None | Full body |
| Speed processor | 10M operations per second | 100 operations per second |
| Computational power | Few operations at the time | Millions of operations at the time |

## Biological inspiration

- Animals are able to **react adaptively** to changes in their external and internal environment, and they use their nervous system to perform these changes
- An **appropriate** model/simulation of the nervous system should be able to produce similar responses and behaviours in artificial systems
- The nervous system is build by **relatively simple units**, the **neurons**, so copying their behaviour and functionality should be the solution

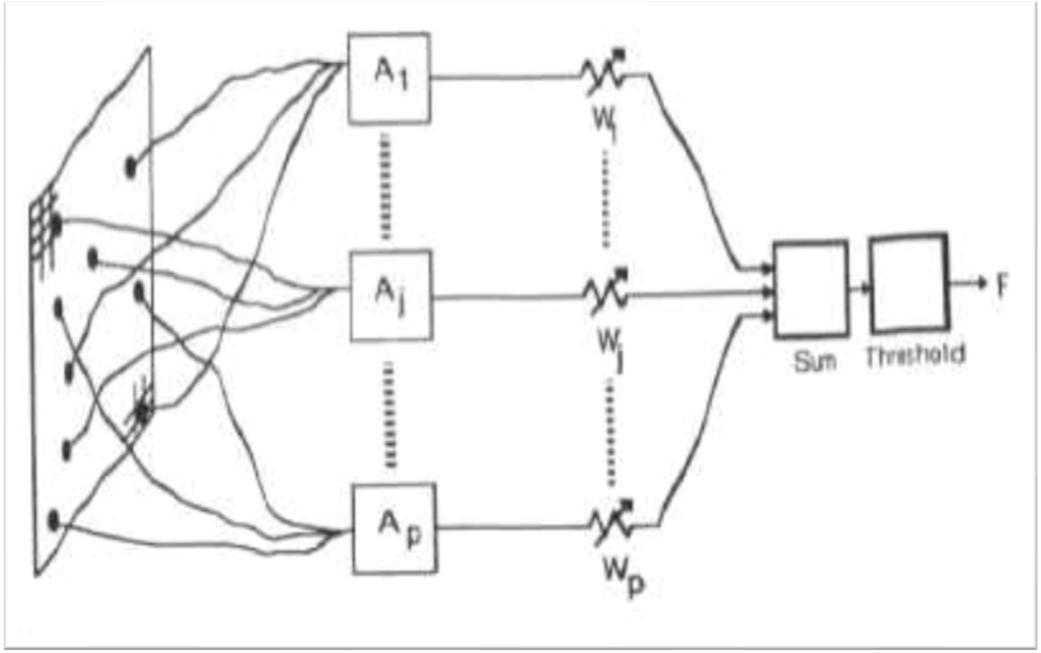
# Artificial neural networks

- An **Artificial Neural Network (ANN)** is an information processing paradigm that is inspired by the biological nervous systems, such as the human brain's information processing mechanism
- The key element of this paradigm is the **novel structure of the information processing system**. It is composed of a **large number of highly interconnected processing elements** (neurons) working in unison to solve specific problems. ANNs, like people, learn by example
- Inherent advantages: **distributed processing** and **representation**, **fault tolerance**, **graceful degradation**, **ability to generalize**
- An ANN is configured for a specific application, such as pattern recognition or data classification, **through a learning process**. Learning in biological systems involves **adjustments** to the **synaptic connections** that exist between the neurons. This is true of ANNs as well.

## History of ANNs

- **1890**: William James - First modeling of neural process of learning
- **1943**: McCulloch and Pitts - Earliest mathematical models based on their understanding of neurology. In their view neurons embedded simple logic functions like **a** ∨ **b**, **a** ∧ **b**
- **1949**: Hebb – He postulates that if two neurons on either side of a synapse are activated simultaneously, then the strength of that synapse is selectively increased
- **1950s**: Donald Hebb and IBM research group - Earliest simulations in which they tried to model biological behavior. They also consulted with neuro-scientists at McGill

- **1958**: Frank Rosenblatt – He proposes an idealized version of neuron that is able to learn:
  - Association units A1, A2, … extract features from user input
  - Output is weighted and associated
  - Function fires if weighted sum of input exceeds a threshold



## History

- **1969**: Minsky and Papert – They criticize Perceptron highlighting a fundamental limitation. They showed that a single layer perceptron **cannot learn** the XOR of two binary inputs.
- **1974**: Werbos – He introduces the idea of back-propagation (already used in Control Theory) for learning in ANNs. Three layers of neurons lead to learning of generic function.
- **1982**: Werbos – He applies the method to ANNs the same way it is used today
- **1986**: Rumelhart, Hilton, and Williams – They show experimentally that Back-propagation ANNs can generate useful internal representations of incoming data in hidden layers
- **1993**: Wan – He is the first one to win an international pattern recognition contest using Back-propagation

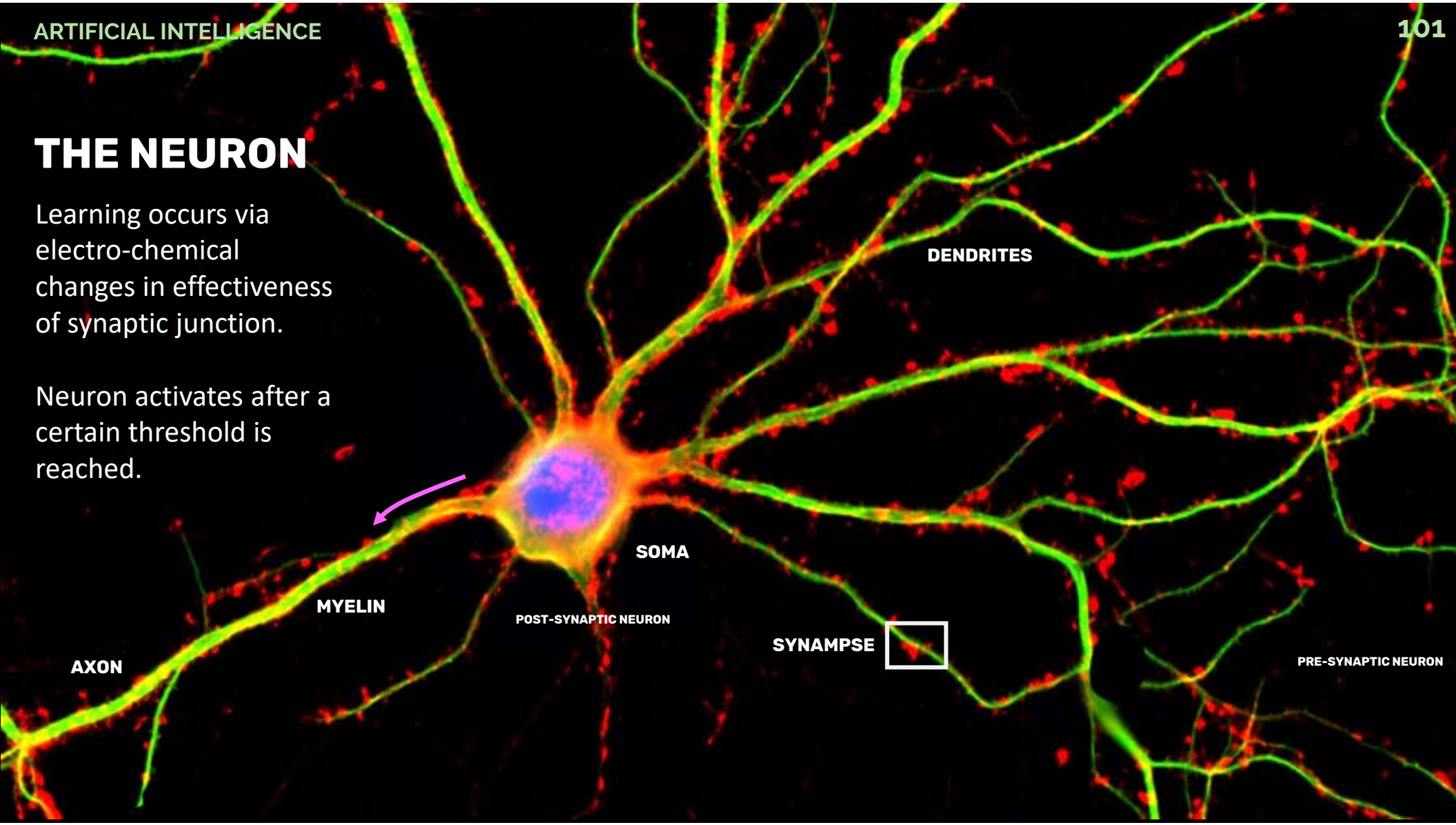
# The neuron

SECTION 01

# THE NEURON

Learning occurs via electro-chemical changes in effectiveness of synaptic junction.

Neuron activates after a certain threshold is reached.



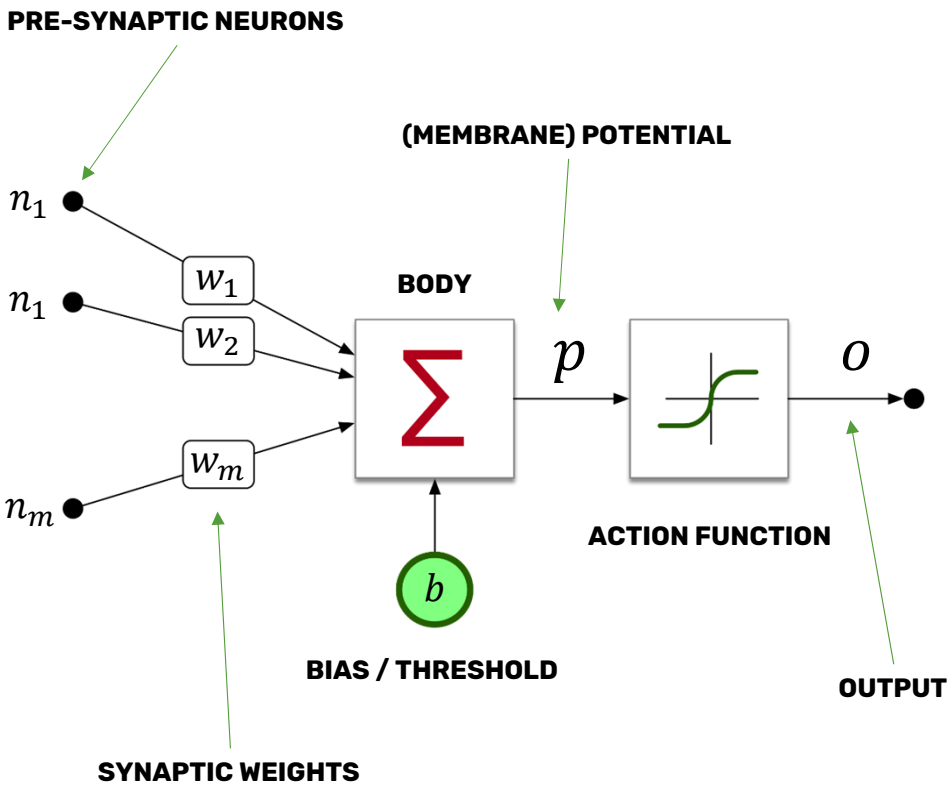
- The **spikes** travelling along the **axon** of the pre-synaptic neuron trigger the release of **neurotransmitter** substances at the synapse
- The **neurotransmitters** cause **excitation** or **inhibition** in the dendrite of the post-synaptic neuron
- The **integration** of the excitatory and inhibitory signals may produce spikes in the post-synaptic neuron
- The **contribution** of the signals depends on the **strength** of the synaptic connection

# Artificial neurons

## THE MCCULLOGH-PITTS MODEL



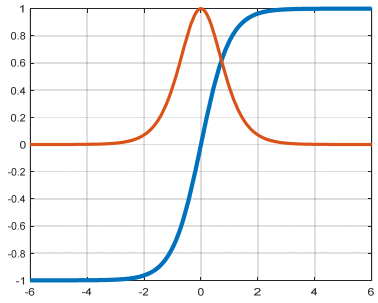
- Spikes are interpreted as spike rates
- Synaptic strength are translated as synaptic weights
- Excitation means positive product between the incoming spike rate and the corresponding synaptic weight
- Inhibition means negative product between the incoming spike rate and the corresponding synaptic weight
- When the total sum of spikes is greater than a threshold, the neuron sends a spike of electrical activity down its axon
- Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes



- Hyperbolic tangent activation function:

$$f(p) = \tanh(p)$$

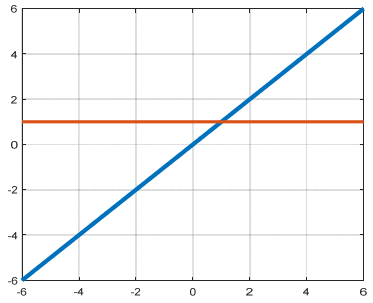
$$f'(p) = 1 - \tanh^2(p)$$



- Linear activation function:

$$f(p) = p$$

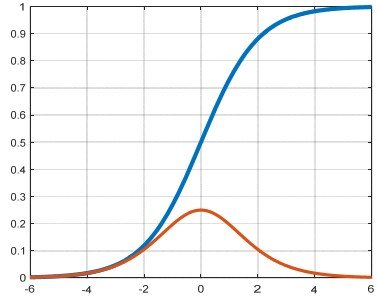
$$f'(p) = 1$$



- Logistic or Sigmoid activation function:

$$f(p) = \frac{1}{1 + e^{-p}}$$

$$f'(p) = f(p)[1 - f(p)]$$

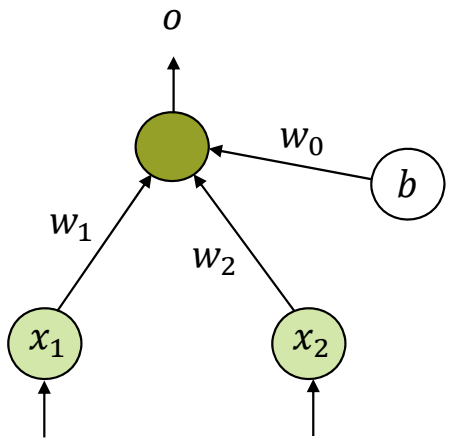


# PERCEPTRON

SECTION 02

**AND FUNCTION**

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




$$f(p) = \begin{cases} 0 & p \leq 0 \\ 1 & p > 0 \end{cases}$$

The Perceptron Learning Algorithm uses a single neuron as a basic learning unit:

1. Initialize weights in a **random** fashion
2. **Present** a pattern and target output
3. **Compute** the potential: $p = \sum_{i=0}^{n} w_i x_i$
4. **Compute** the output: $o = f(p)$
5. **Update** weights: $w_i(t+1) = w_i(t) + \Delta w_i$
6. Repeat from 2 until acceptable level of error

- Widrow and Hoff suggest a rule, called **Delta Rule**, for weight modification:

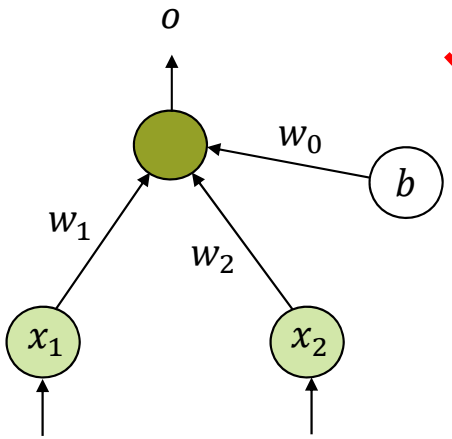
$$w_i(t+1) = w_i(t) + \Delta w_i \qquad \Delta w_i = \eta\, d\, x_i(t)$$

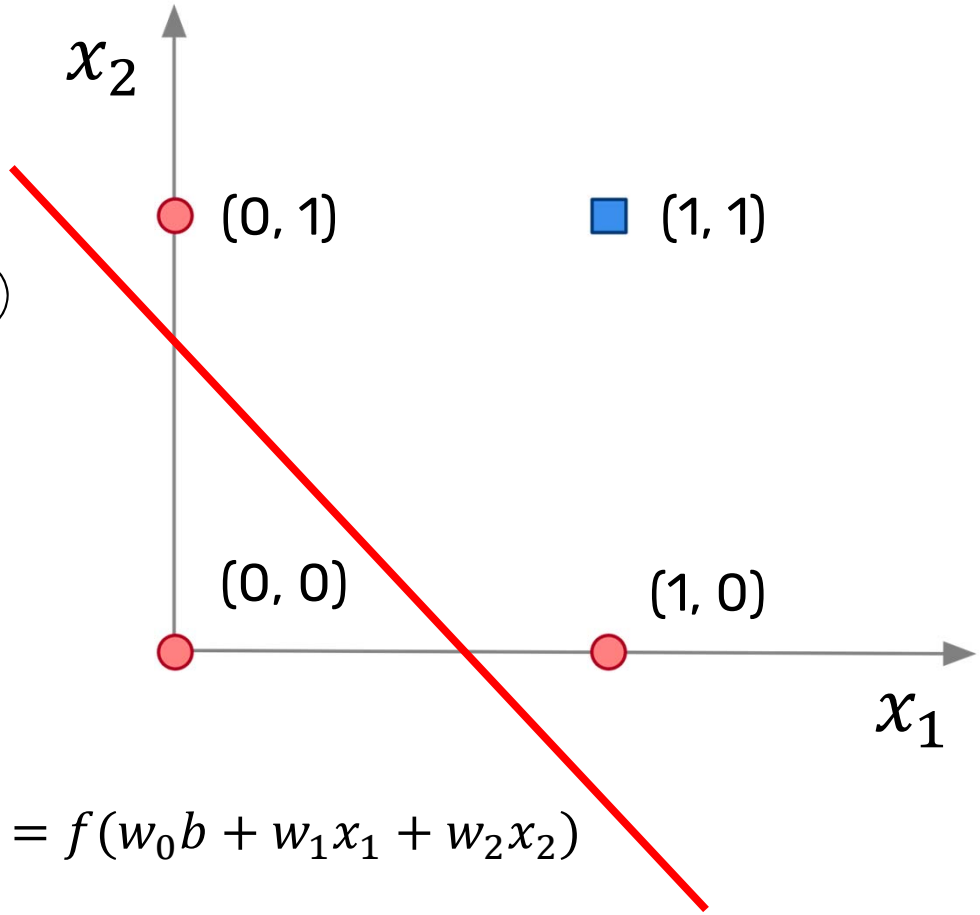
where:

- $\eta$: **learning rate** ($0 < \eta \leq 1$, typically $1$)
- $d$: **error signal**, which is usually defined as $d = desired\ ouput\ - neuron\ output$

**AND FUNCTION**

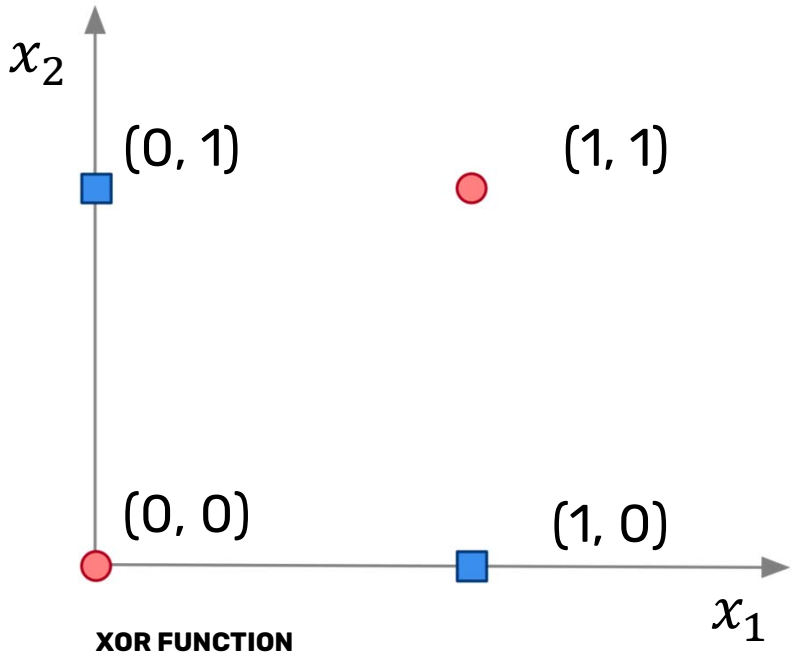
| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



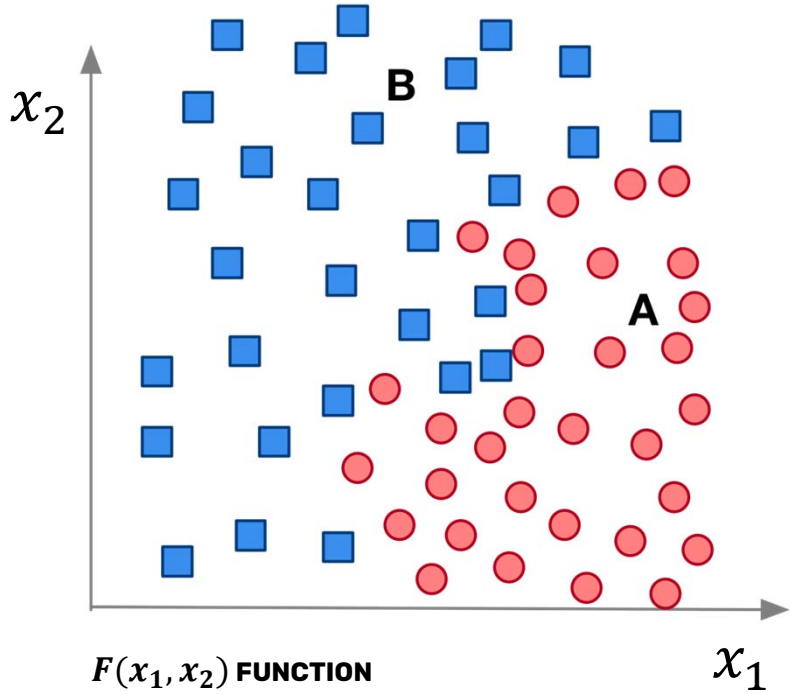




$$y = f(w_0 b + w_1 x_1 + w_2 x_2)$$

- Minsky and Papert discovered that Perceptron is able to form **only linear discriminate functions**
- In reality, most functions are far **more complex**



XOR FUNCTION



$F(x_1, x_2)$ FUNCTION

# The back-propagation algorithm

SECTION 03

In 1982 and then 1986, Werbos applies a Control Theory method to ANNs with multiple hidden layers, creating the most important learning algorithm in the history of ANNs:

- The algorithm is conceptually simple: the global error is **backward propagated** to network nodes, and the weights are **modified proportional** to their contribution.

- The algorithm searches for weight values that **minimize** the **total error** of the network over the training set
- It consists of the repeated application of the following two passes:
  1. **Forward pass**: the network is **activated** on one example and the error of each neuron of the output layer is **calculated**

  2. **Backward pass**: the network error is used for **updating** the weights; starting from the output layer, the error is **propagated backwards** through the network, layer by layer, recursively calculating the local gradient of each neuron

# Forward propagation

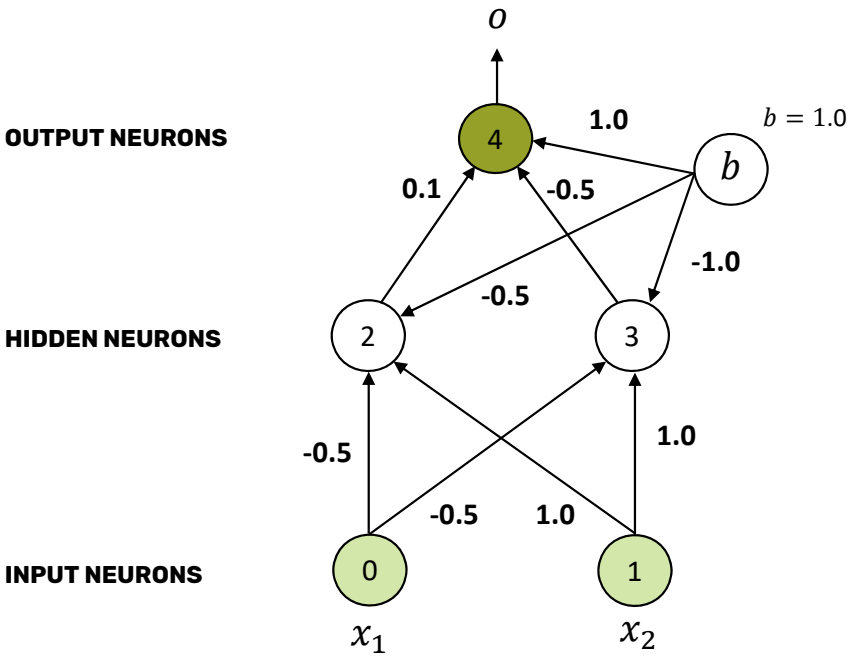
SECTION 04

1. Apply input vector $X$ to layer of neurons

2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$




Function to learn:

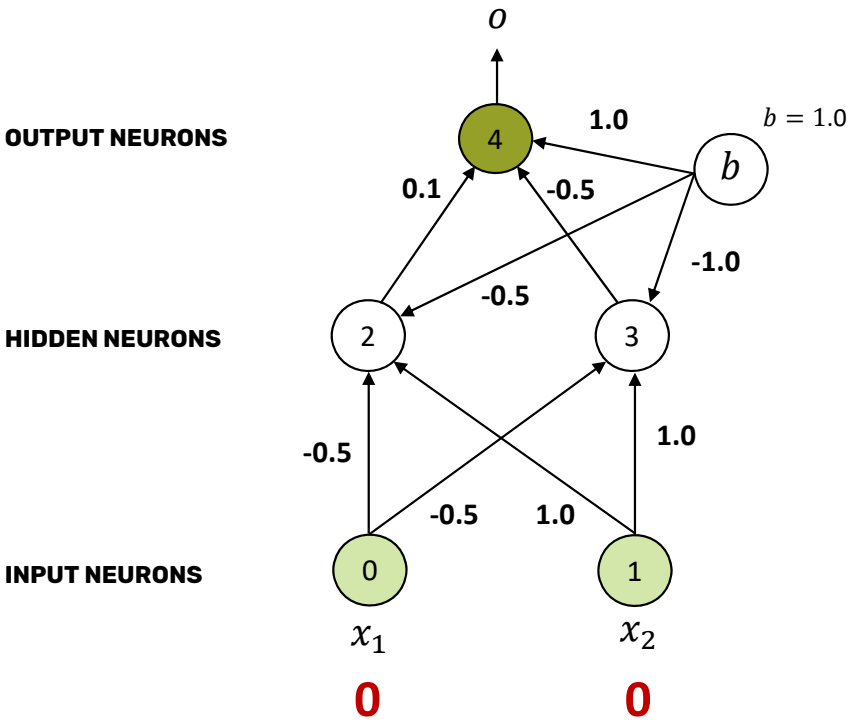
| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | - |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

1. Apply input vector $X$ to layer of neurons

2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$




Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | - |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

1. Apply input vector $X$ to layer of neurons

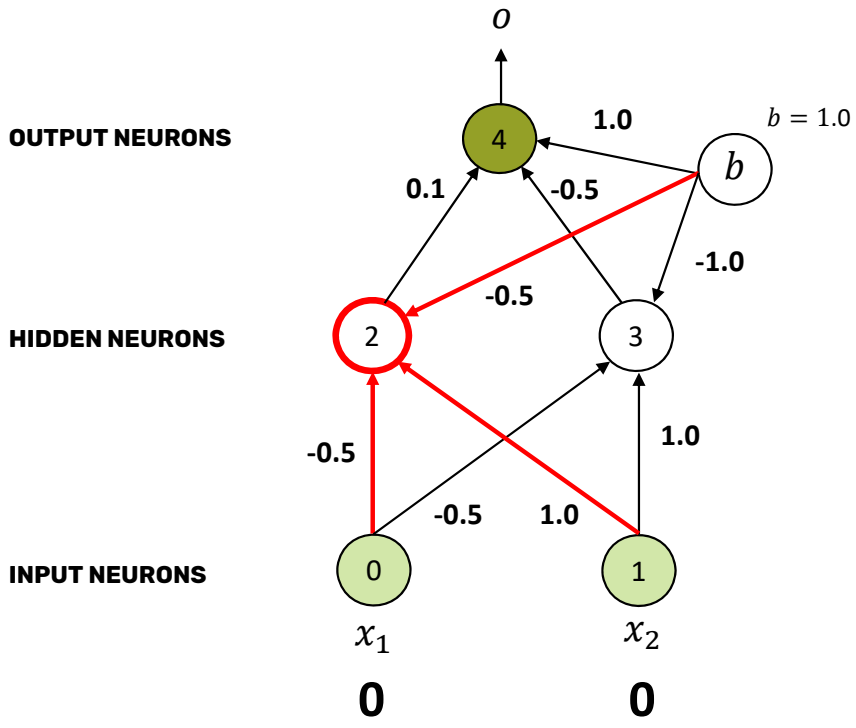
2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $o(p) = \frac{1}{1 + e^{-p}}$

Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | - |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |




$o_2(-0.5 \times 0 + 1.0 \times 0 \ - 0.5 \times 1.0) = o_2(-0.5) = \mathbf{0.3775}$

1. Apply input vector $X$ to layer of neurons

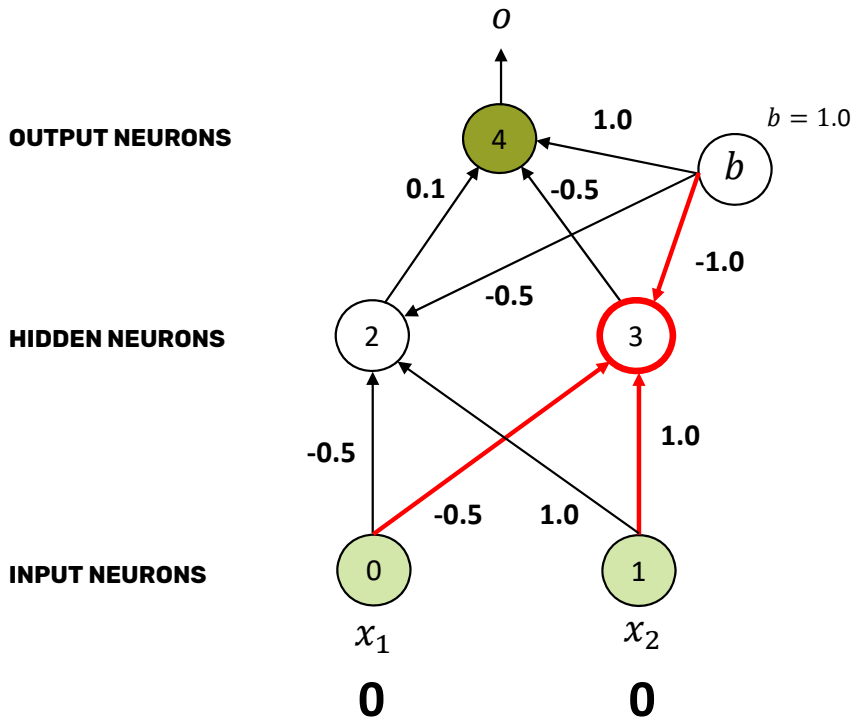
2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$

Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | - |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |




$o_2(-0.5 \times 0 + 1.0 \times 0 \ - 0.5 \times 1.0) = o_2(-0.5) = 0.3775$

$o_3(-0.5 \times 0 - 1.0 \times 0 \ - 1.0 \times 1.0) = o_3(-1.0) = \mathbf{0.2689}$

1. Apply input vector $X$ to layer of neurons

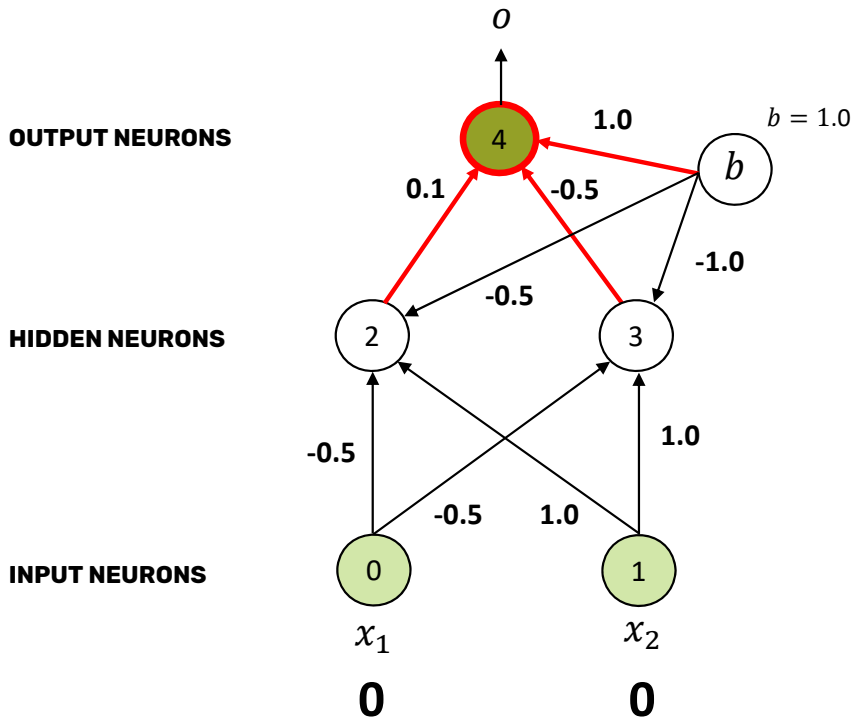
2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$

Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |




$o_2(-0.5 \times 0 + 1.0 \times 0 - 0.5 \times 1.0) = o_2(-0.5) = 0.3775$

$o_3(-0.5 \times 0 - 1.0 \times 0 - 1.0 \times 1.0) = o_3(-1.0) = 0.2689$

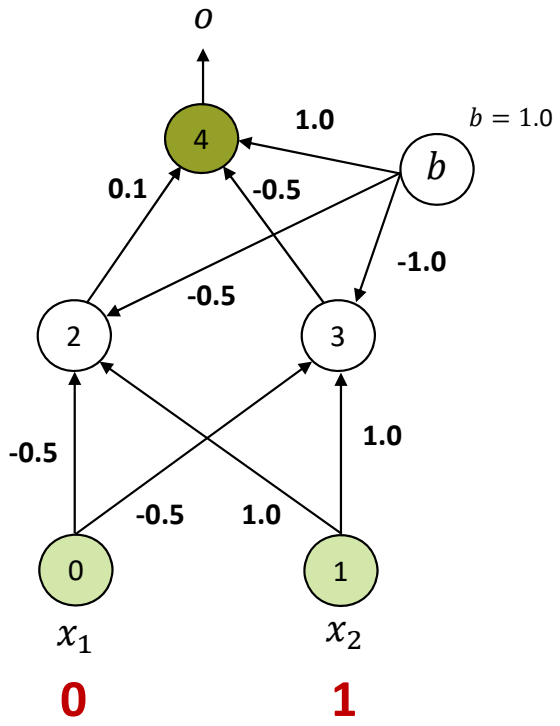
$o_4(0.1 \times 0.3775 - 0.5 \times 0.2689 + 1.0 \times 1.0) = o_4(0.9033) = \mathbf{0.7116}$

1. Apply input vector $X$ to layer of neurons

2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$




Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

1. Apply input vector $X$ to layer of neurons

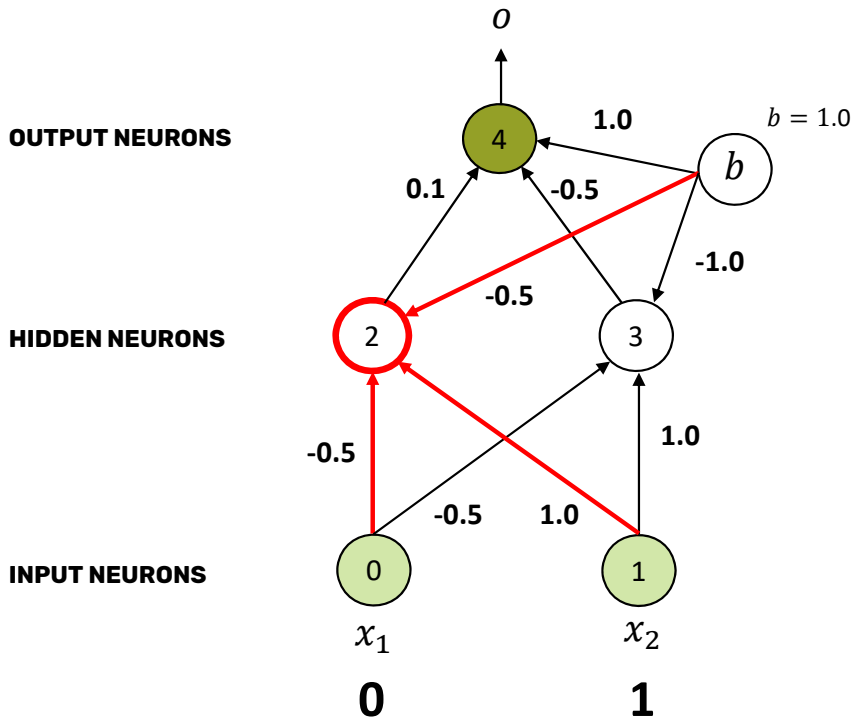
2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$

Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |




$o_2(-0.5 \times 0 + 1.0 \times 1 - 0.5 \times 1.0) = o_2(0.5) = \mathbf{0.6225}$

1. Apply input vector $X$ to layer of neurons

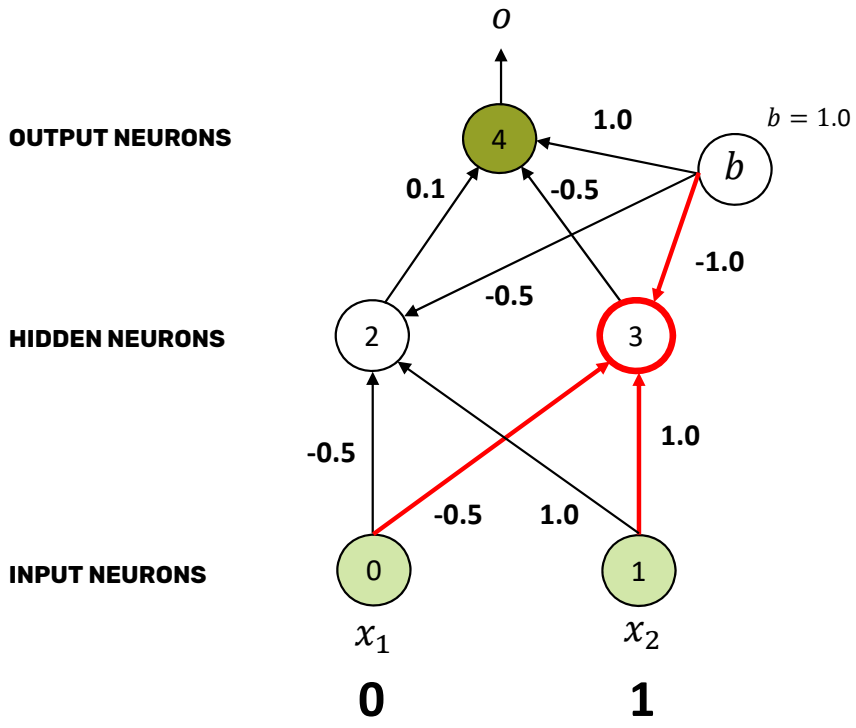
2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$

Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |




$o_2(-0.5 \times 0 + 1.0 \times 1 - 0.5 \times 1.0) = o_2(0.5) = 0.6225$

$o_3(-0.5 \times 0 - 1.0 \times 1 - 1.0 \times 1.0) = o_3(-2.0) = \mathbf{0.1192}$

1. Apply input vector $X$ to layer of neurons

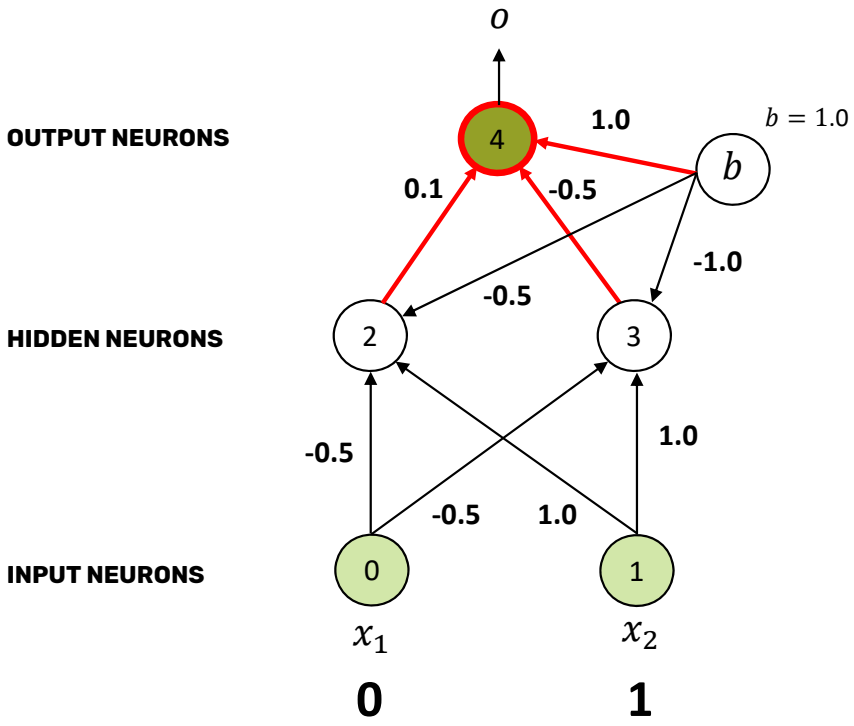
2. Calculate: $p_i(X, W) = \sum_{j=0}^{N} w_{ji} o_j + w_i b$

   where:
   - $o_j$: the output of previous layer neuron $j$
   - $w_{ji}$: the weight of going from node $j$ to node $i$
   - $N$: the number of neurons in the previous layer

3. Calculate output activation: $f(p) = \frac{1}{1 + e^{-p}}$

Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | 0.7316 |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |




$o_2(-0.5 \times 0 + 1.0 \times 1 - 0.5 \times 1.0) = o_2(0.5) = 0.6225$

$o_3(-0.5 \times 0 - 1.0 \times 1 - 1.0 \times 1.0) = o_3(-2.0) = 0.1192$

$o_4(0.1 \times 0.6225 - 0.5 \times 0.1192 + 1.0 \times 1.0) = o_4(1.0027) = \mathbf{0.7316}$

# Backward propagation

SECTION 05

1. The **calculation of error** is based on difference between target and actual output:

$$\delta_i = f'(p_i)(t_i - o_i)$$

| | |
|---|---|
| $t_i$ | Target output for the neuron $i$ |
| $o_i$ | Output of the neuron $i$ |
| $p_i$ | Potential of the neuron $i$ |

1. Calculate the contribution to the error by the **hidden neuron**:

$$\delta_i = f'(p_i) \sum_{j=1}^{M} w_{ij}\delta_j$$

| | |
|---|---|
| $w_{ij}$ | The weight from node $i$ to node $j$ |
| $\delta_j$ | The signal error for the neuron $j$ |
| $p_i$ | Potential of the neuron $i$ |
| $M$ | Number of neurons in the next layer |

2. The **rate of change** of the error which is the important feedback through the network:

$$w_{ij}(t+1) = w_{ij}(t) + \eta o_i \delta_j$$

| | |
|---|---|
| $w_{ji}$ | The weight from node $i$ to node $j$ |
| $\eta$ | Learning rate |
| $o_i$ | The output of the neuron $i$ |
| $\delta_j$ | The signal error for the neuron $j$ |

3. Repeat from 2 till the **total error** is less than a threshold or a maximum number of iterations

$$E = \frac{1}{2}\sum_{i=1}^{N}(t_i - o_i)^2$$

| | |
|---|---|
| $t_i$ | Target output for the neuron $i$ |
| $o_i$ | Output of the neuron $i$ |
| $N$ | Number of neurons in the output layer |

A way to speed-up learning is to use a technique called **momentum descent**, that is analogous to physical momentum of a ball

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) + \alpha\, \Delta w_{ij}(t-1) \qquad 0 < \alpha < 1$$

- It augments the effective learning rate $\eta$ to vary the amount a weight is updated
- It can skip over small local minima

## Back-propagation

PROS



- Proven training method for multi-layer nets
- It's able to learn any arbitrary function (XOR)
- It's most useful for non-linear mappings
- It works well with noisy data
- It generalizes well given sufficient examples
- Rapid recognition speed
- It has inspired many new learning algorithms

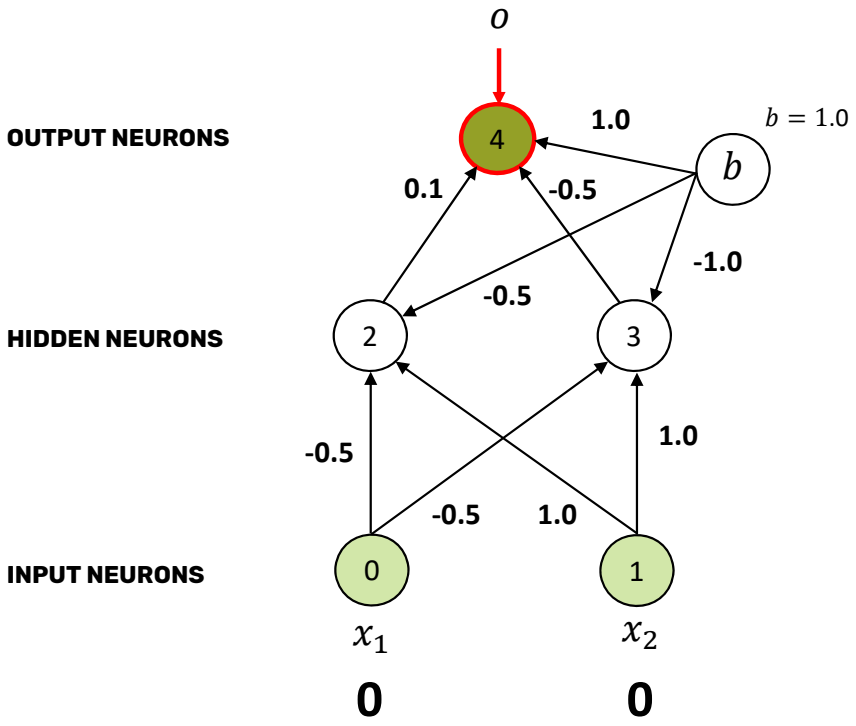
## CONS

- It can get stuck in local minimum - but not generally a concern
- It seems biologically implausible
- High space and time complexity: $O(W^3)$
- It is not possible to see how the decision is made
- It works best with suited for supervised learning
- It works poorly on dense data with few input variables

1. The **calculation of error** is based on difference between target and actual output:

$$\delta_i = f'(p_i)(t_i - o_i)$$

$$\delta_i = o_i\,(1 - o_i)(t_i - o_i)$$




Function to learn:

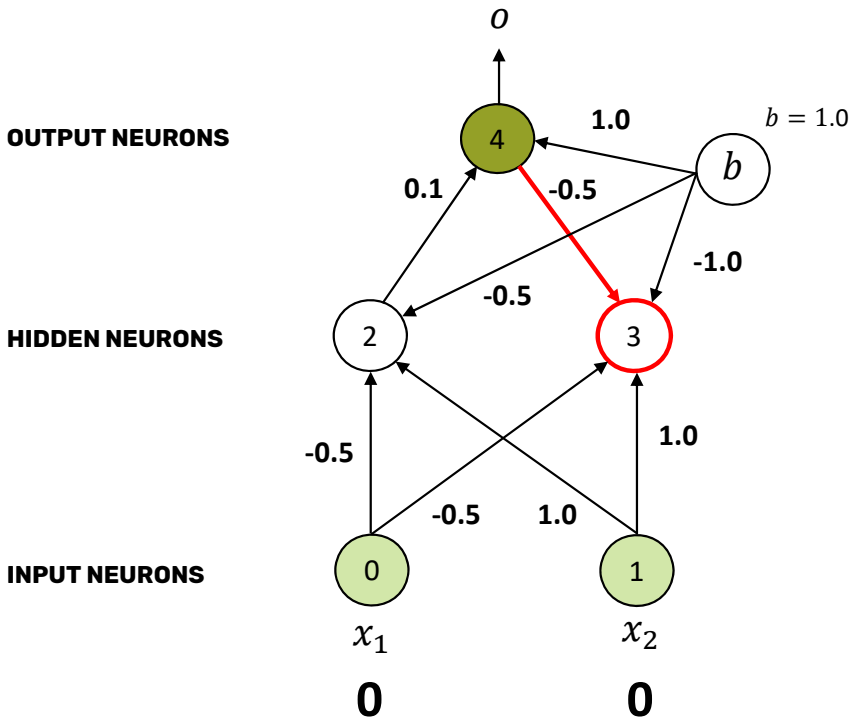
| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

$\delta_4 = 0.7116 \times (1 - 0.7116) \times (0 - 0.7116) = \mathbf{-0.1460}$

1. Calculate the contribution to the error by the hidden neuron:

$$\delta_i = f'(p_i) \sum_{j=1}^{M} w_{ij}\delta_j$$

$$\delta_i = o_i\,(1 - o_i) \sum_{j=1}^{M} w_{ij}\delta_j$$




Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

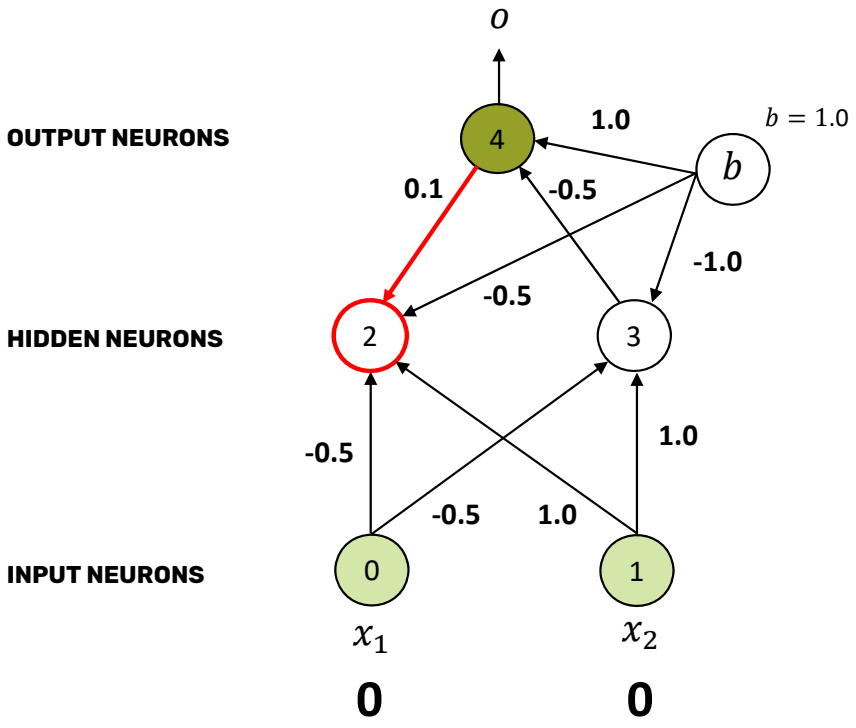
$\delta_4 = 0.7116 \times (1 - 0.7116) \times (0 - 0.7116) = -0.1460$

$\delta_3 = 0.2689 \times (1 - 0.2689) \times (-0.5 \times -0.1460) = \mathbf{0.0144}$

1. Calculate the contribution to the error by the hidden neuron:

$$\delta_i = f'(p_i) \sum_{j=1}^{M} w_{ij}\delta_j$$

$$\delta_i = o_i\,(1 - o_i) \sum_{j=1}^{M} w_{ij}\delta_j$$




Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

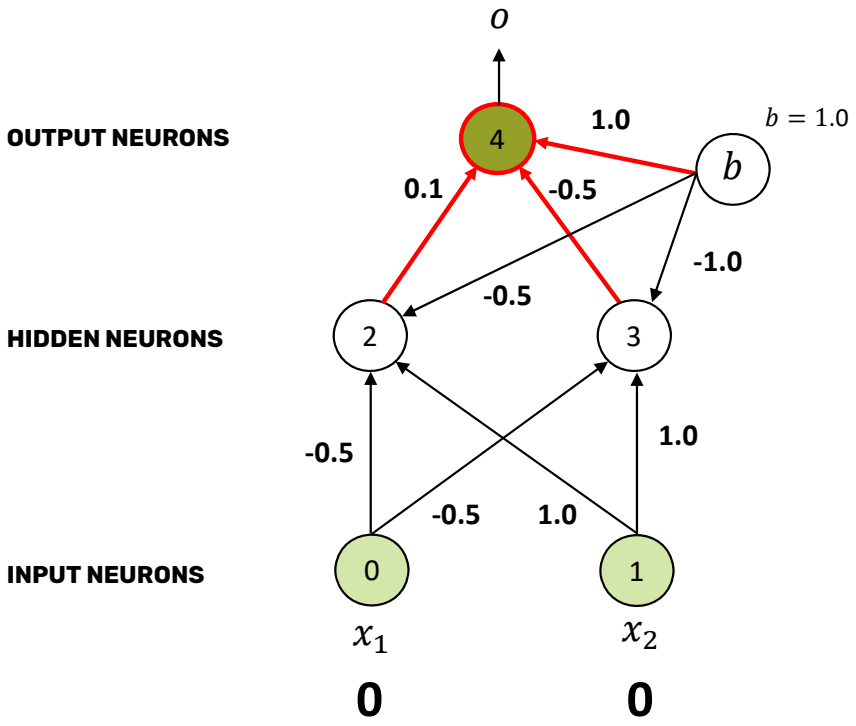
$\delta_4 = 0.7116 \times (1 - 0.7116) \times (0 - 0.7116) = -0.1460$

$\delta_3 = 0.2689 \times (1 - 0.2689) \times (-0.5 \times -0.1460) = 0.0144$

$\delta_2 = 0.3775 \times (1 - 0.3775) \times (0.1 \times -0.1460) = \mathbf{-0.0034}$

2. The **rate of change** of the error which is the important feedback through the network:

$$\Delta w_{ij} = \eta o_i \delta_j$$



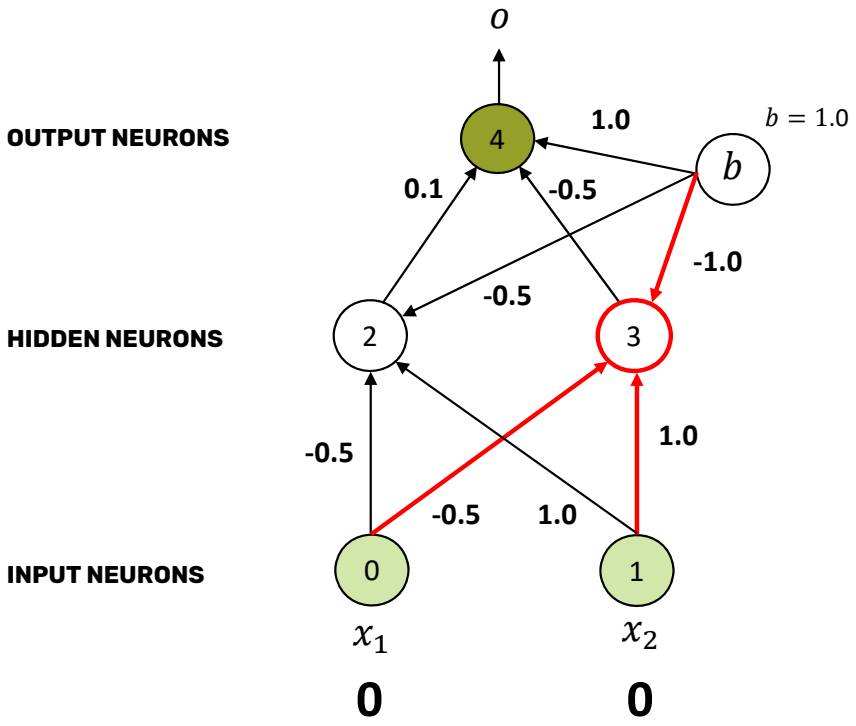

Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

$\Delta w_{24} = \eta o_2 \delta_4 \quad o_2 = 0.6225 \quad \delta_4 = -0.1460 \quad \Delta w_{24} = \mathbf{-0.0091}$
$\Delta w_{34} = \eta o_3 \delta_4 \quad o_3 = 0.1192 \quad \delta_4 = -0.1460 \quad \Delta w_{34} = \mathbf{-0.0017}$
$\Delta w_{b4} = \eta o_b \delta_4 \quad o_b = 1.0 \quad \delta_4 = -0.1460 \quad \Delta w_{b4} = \mathbf{-0.0146}$

2. The **rate of change** of the error which is the important feedback through the network:

$$\Delta w_{ji} = \eta o_j \delta_j$$




Function to learn:

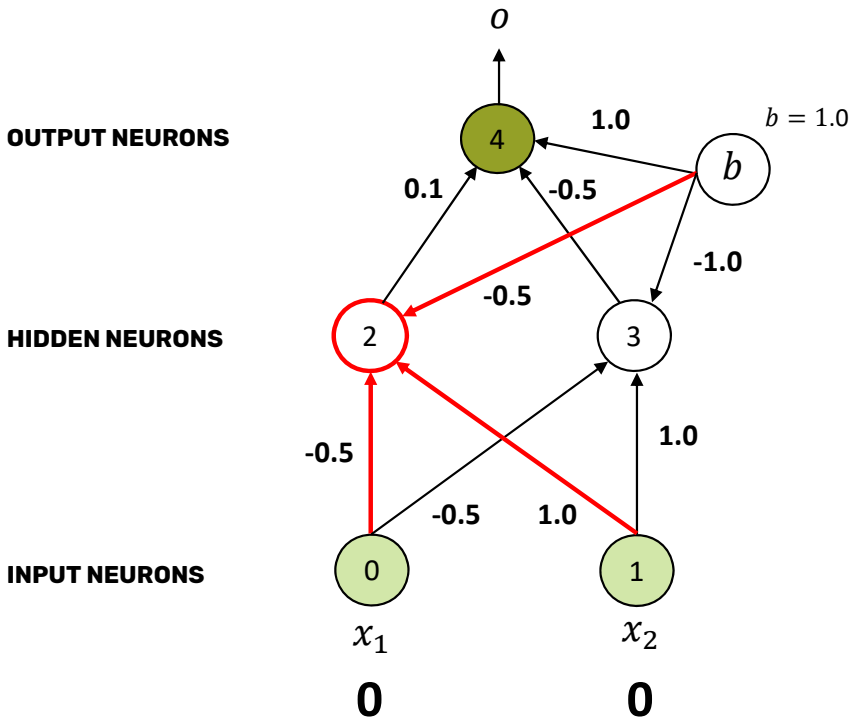
| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

$\Delta w_{24} = \eta o_2 \delta_4 \quad o_2 = 0.6225 \quad \delta_4 = -0.1460 \quad \Delta w_{24} = -0.0091$
$\Delta w_{34} = \eta o_3 \delta_4 \quad o_3 = 0.1192 \quad \delta_4 = -0.1460 \quad \Delta w_{34} = -0.0017$
$\Delta w_{b4} = \eta o_b \delta_4 \quad o_b = 1.0 \quad \delta_4 = -0.1460 \quad \Delta w_{b4} = -0.0146$

$\Delta w_{03} = \eta o_0 \delta_3 \quad o_0 = 0.0 \quad \delta_3 = 0.0144 \quad \Delta w_{03} = \mathbf{0.0}$
$\Delta w_{13} = \eta o_1 \delta_3 \quad o_1 = 0.0 \quad \delta_3 = 0.0144 \quad \Delta w_{13} = \mathbf{0.0}$
$\Delta w_{b3} = \eta o_b \delta_3 \quad o_b = 1.0 \quad \delta_3 = 0.0144 \quad \Delta w_{b3} = \mathbf{0.0014}$

2. The **rate of change** of the error which is the important feedback through the network:

$$\Delta w_{ji} = \eta o_j \delta_j$$




Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

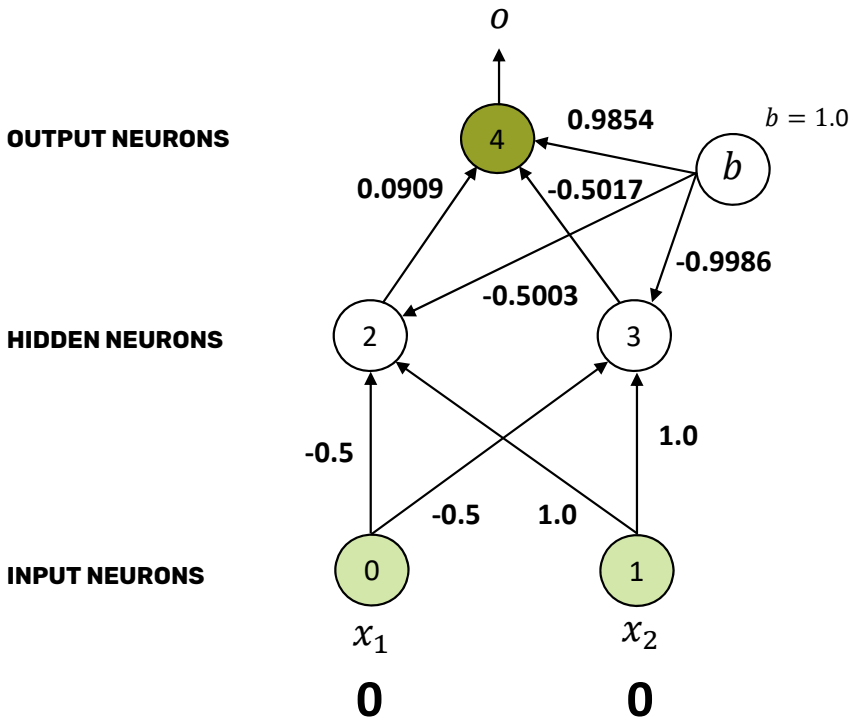
$\Delta w_{24} = \eta o_2 \delta_4 \quad o_2 = 0.6225 \quad \delta_4 = -0.1460 \quad \Delta w_{24} = -0.0091$
$\Delta w_{34} = \eta o_3 \delta_4 \quad o_3 = 0.1192 \quad \delta_4 = -0.1460 \quad \Delta w_{34} = -0.0017$
$\Delta w_{b4} = \eta o_b \delta_4 \quad o_b = 1.0 \quad \delta_4 = -0.1460 \quad \Delta w_{b4} = -0.0146$

$\Delta w_{03} = \eta o_0 \delta_3 \quad o_0 = 0.0 \quad \delta_3 = 0.0730 \quad \Delta w_{03} = 0.0$
$\Delta w_{13} = \eta o_1 \delta_3 \quad o_1 = 0.0 \quad \delta_3 = 0.0730 \quad \Delta w_{13} = 0.0$
$\Delta w_{b3} = \eta o_b \delta_3 \quad o_b = 1.0 \quad \delta_3 = 0.0730 \quad \Delta w_{b3} = 0.0014$

$\Delta w_{02} = \eta o_0 \delta_2 \quad o_0 = 0.0 \quad \delta_2 = -0.0034 \quad \Delta w_{02} = \mathbf{0.0}$
$\Delta w_{12} = \eta o_1 \delta_2 \quad o_1 = 0.0 \quad \delta_2 = -0.0034 \quad \Delta w_{12} = \mathbf{0.0}$
$\Delta w_{b2} = \eta o_b \delta_2 \quad o_b = 1.0 \quad \delta_2 = -0.0034 \quad \Delta w_{b2} = \mathbf{-0.0003}$

The state of the weights after the iteration of one example.

$$w_{ij}(t+1) = w_{ij}(t) + \eta o_i \delta_j$$




Function to learn:

| $x_1$ | $x_2$ | $t$ | $o$ |
|---|---|---|---|
| 0 | 0 | 0 | 0.7116 |
| 0 | 1 | 1 | - |
| 1 | 0 | 1 | - |
| 1 | 1 | 0 | - |

# QUESTIONS ?