ARTIFICIAL NEURAL NETWORKS 1

ARTIFICIAL INTELLIGENCE | COMP 131

TODAY ON AI

- Introduction and history
- The neuron
- Perceptron
- The back-propagation algorithm
- Forward propagation
- Backward propagation
- Questions?

THE HUMAN BRAIN

100 billion neurons
15 to 100 thousand synapsis per neuron

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

In nature, animals' nervous systems evolved so that can **react adaptively** to changes in their external and internal environment.

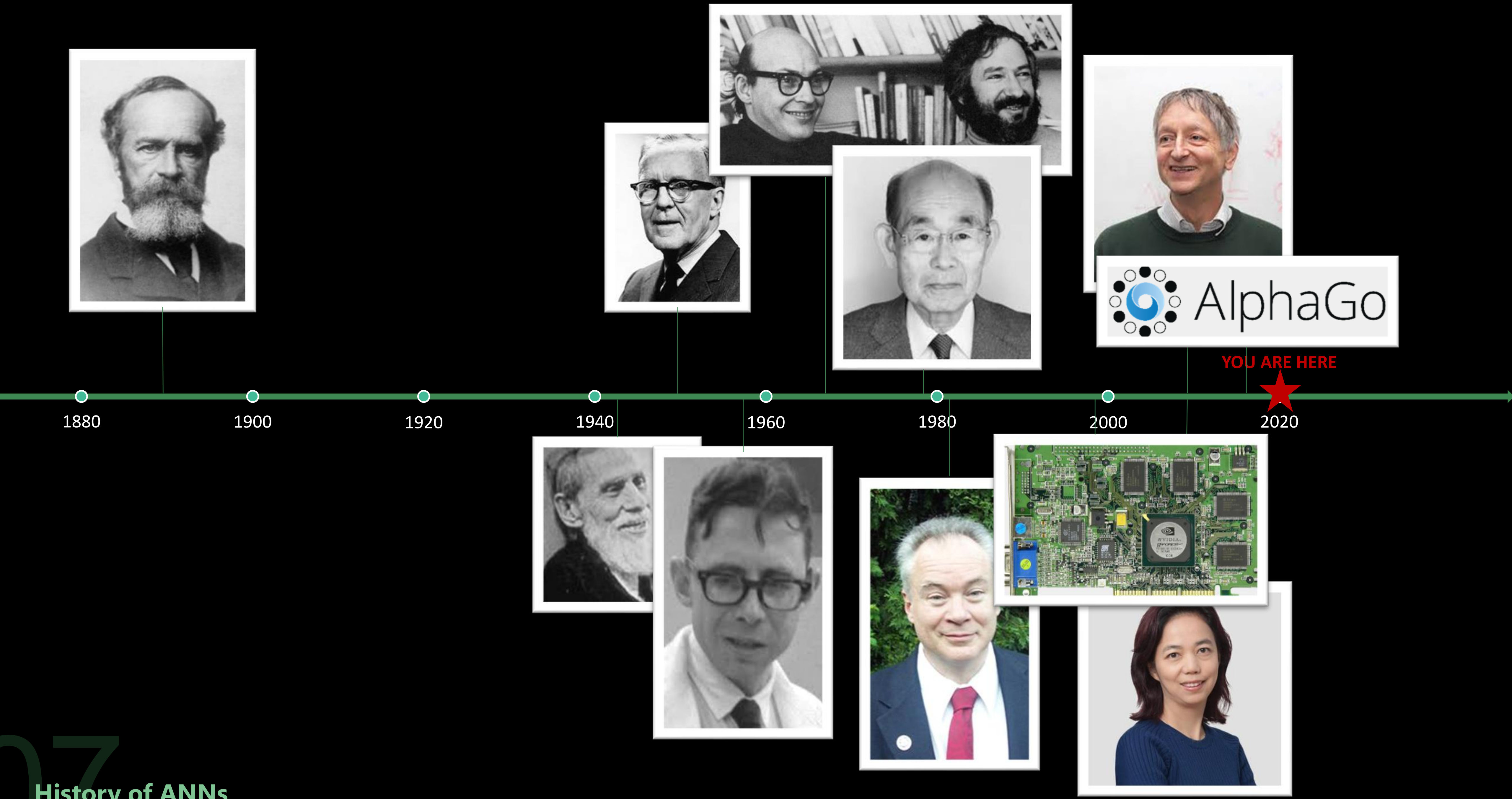
Their nervous system is built by **relatively simple units**, the **neurons**.

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.

ANNs inherent their advantages from the biological counterparts: **distributed processing** and **representation**, **fault tolerance**, **graceful degradation**, **ability to generalize**

The key element of this paradigm are:

- The **novel structure of the information processing system**: It is composed of **many highly interconnected processing elements** (neurons) working in unison to solve specific problems.
- The **topology** is configured for a specific application (pattern recognition, data classification, etc.)
- Like in biological systems, learning involves **adjustments** to the **synaptic connections** that exist between the neurons.



07

History of ANNs

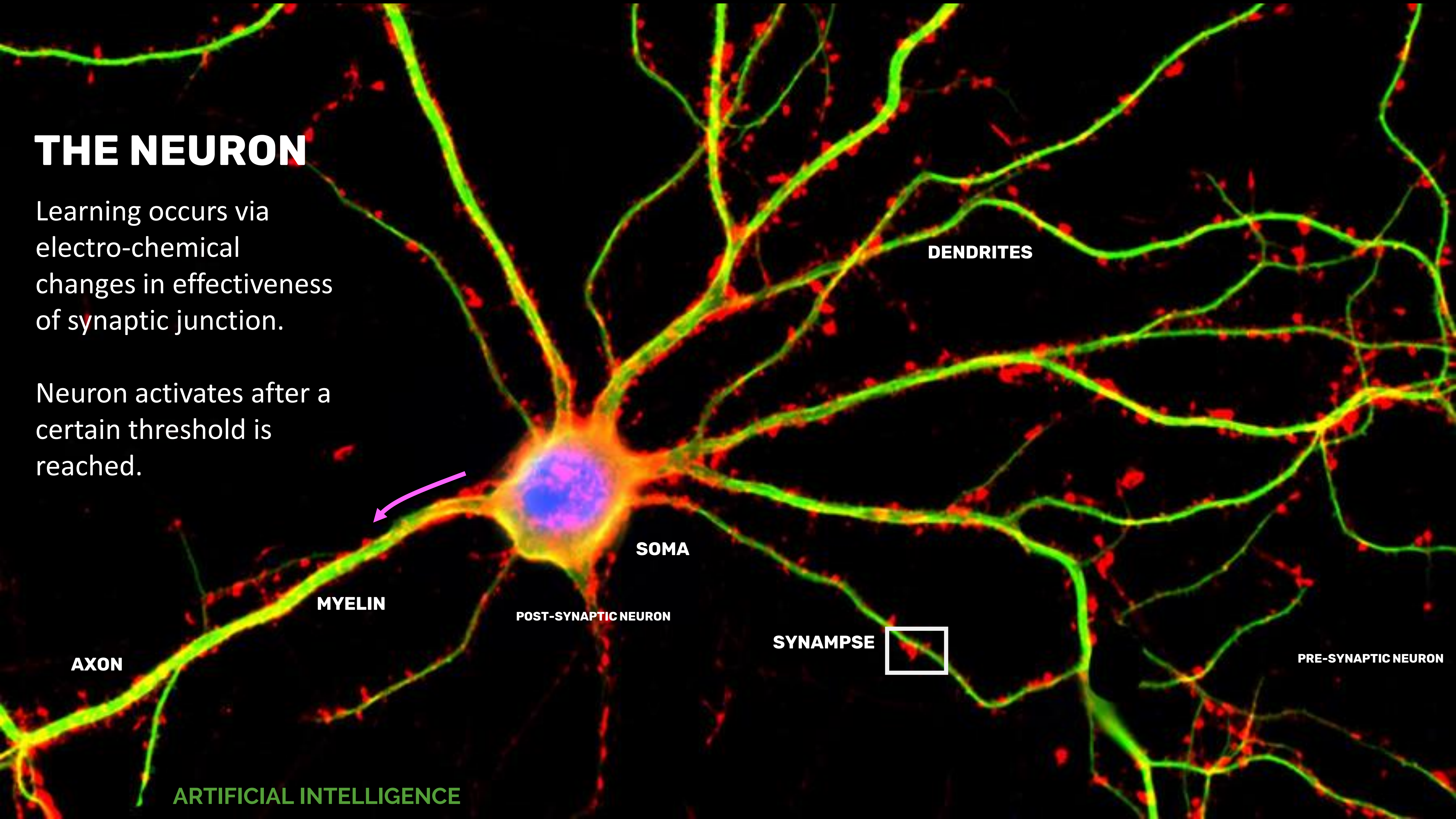
SECTION 01

The neuron

THE NEURON

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

Neuron activates after a certain threshold is reached.



The information processing happens in stages:

1. The **spikes** travelling along the **axon** of the pre-synaptic neuron trigger the release of **neurotransmitter** substances at the synapse

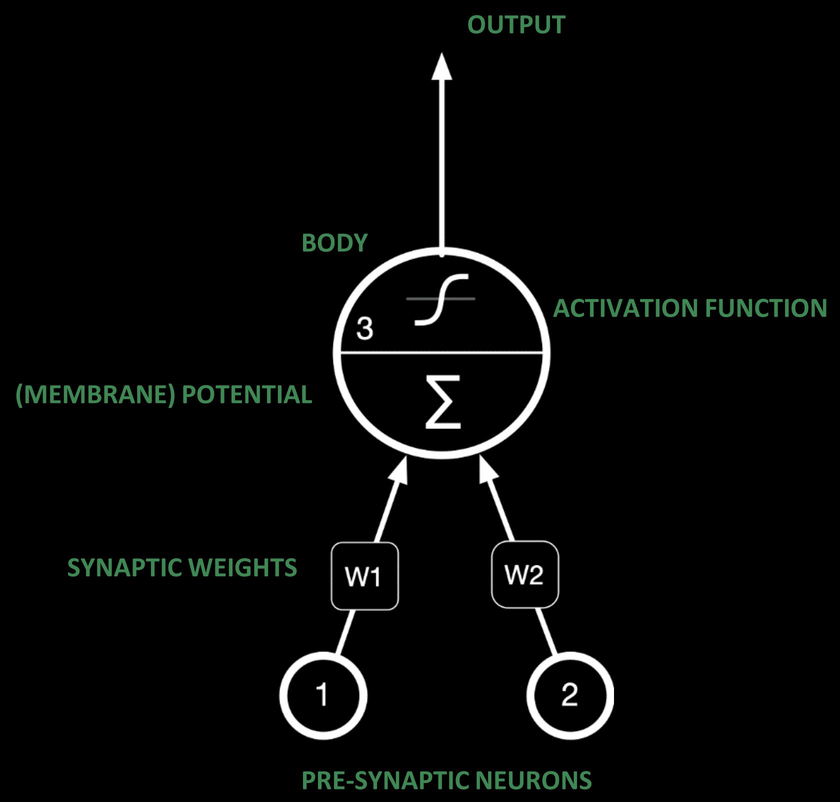
2. The **neurotransmitters** cause **excitation** or **inhibition** in the dendrite of the post-synaptic neuron

3. The **integration** of the excitatory and inhibitory signals into a **membrane potential** may produce spikes in the axon

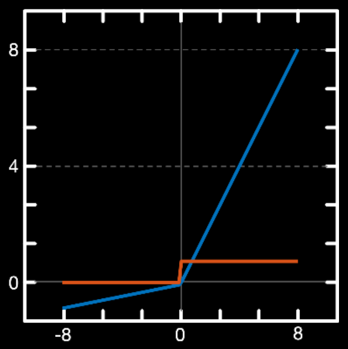
4. The signals produced is **communicated** to the post-neurons with a **strength** that depends on the synaptic connection

The **McCullogh-Pitts model** for an artificial neuron:

- **Spikes** are interpreted as potentials
- **Synaptic strength** are translated as synaptic weights
- **Excitation** means product between the incoming potential and a positive synaptic weight
- **Inhibition** means product between the incoming potential and the negative synaptic weight
- When **the total sum of potentials is greater than a threshold**, the neuron sends an activation potential down its axon



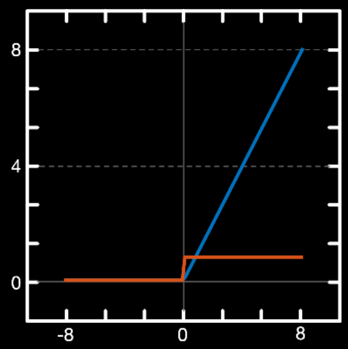
LEAKY RELU



$$O(p) = \begin{cases} p & \text{for} \quad p \geq 0 \\ 0.01p & \text{for} \quad p < 0 \end{cases}$$

$$O'(p) = \begin{cases} 1 & \text{for} \quad p \geq 0 \\ 0.01 & \text{for} \quad p < 0 \end{cases}$$

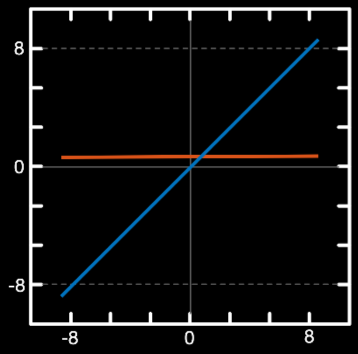
RELU



$$O(p) = \begin{cases} p & \text{for} \quad p \geq 0 \\ 0 & \text{for} \quad p < 0 \end{cases}$$

$$O'(p) = \begin{cases} 1 & \text{for} \quad p \geq 0 \\ 0 & \text{for} \quad p < 0 \end{cases}$$

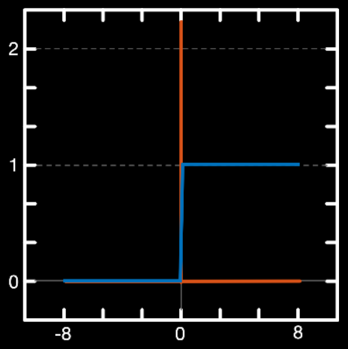
LINEAR / IDENTITY



$$O(p) = p$$

$$O'(p) = 1$$

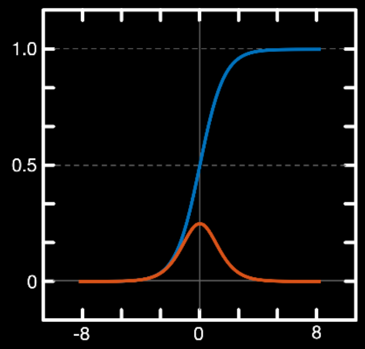
STEP



$$O(p) = \begin{cases} 1 & \text{for} \quad p \geq 0 \\ 0 & \text{for} \quad p < 0 \end{cases}$$

$$O'(p) = \begin{cases} 0 & \text{for} \quad p \neq 0 \\ ? & \text{for} \quad p = 0 \end{cases}$$

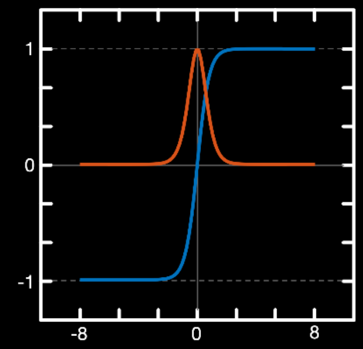
SIGMOID / LOGISTIC



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

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

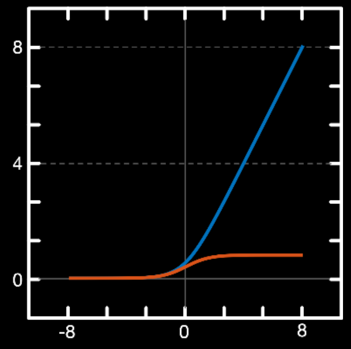
TANH



$$O(p) = \tanh(p)$$

$$O'(p) = 1 - O(p)^2$$

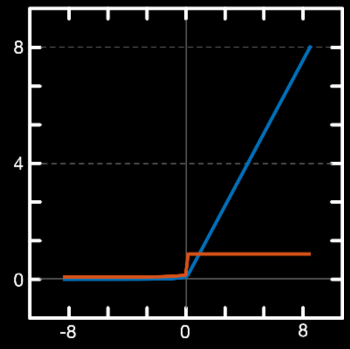
SOFTPLUS



$$O(p) = \ln(1 + e^{p})$$

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

ELU



$$O(p) = \begin{cases} p & \text{for} \quad p \geq 0 \\ \alpha(e^{p} - 1) & \text{for} \quad p < 0 \end{cases}$$

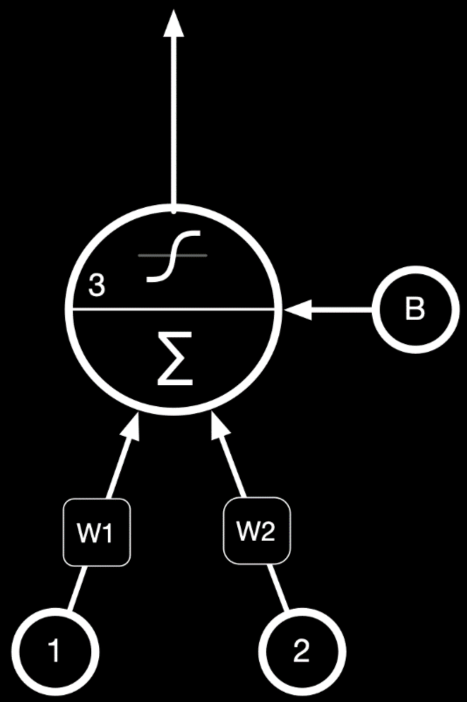
$$O'(p) = \begin{cases} 1 & \text{for} \quad p \geq 0 \\ O(p) + \alpha & \text{for} \quad p < 0 \end{cases}$$

SECTION 02

PERCEPTRON

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: $p(t) = \sum_{i=1}^{N} w_i o_i$
4. **Compute** the output: $o(t) = \begin{cases} 1 & \text{for} \quad p(t) + B \geq 0 \\ 0 & \text{otherwise} \end{cases}$
5. **Update** weights: $w_i(t+1) = w_i(t) + \Delta w_i(t)$
6. **Calculate** output error
7. Repeat from 2 until acceptable level of error



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

$$w_i(t+1) = w_i(t) + \Delta w_i(t) \qquad \Delta w_i(t) = \eta\, o'_i(t)\, [d_i(t) - o_i(t)]$$

where:

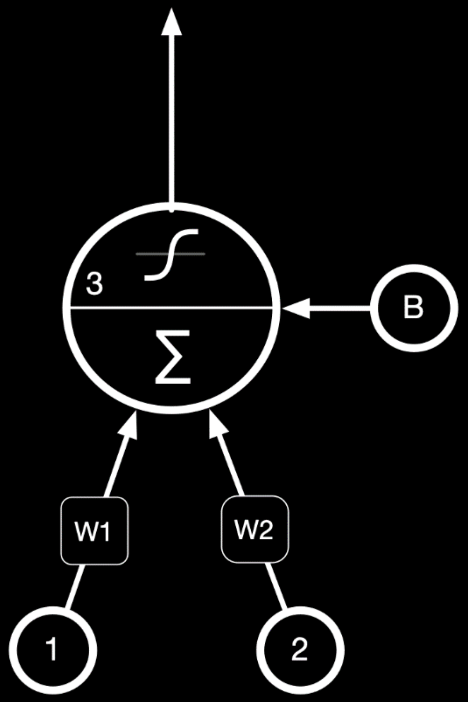
- η: **learning rate** ($0 < \eta \leq 1$, typically 1)
- $d_i(t)$: **desired output** of the i neuron at time t
- $o_i(t)$: **actual output** of the i neuron at time t
- $o'_i(t)$: **actual output derivative** of the i neuron at time t

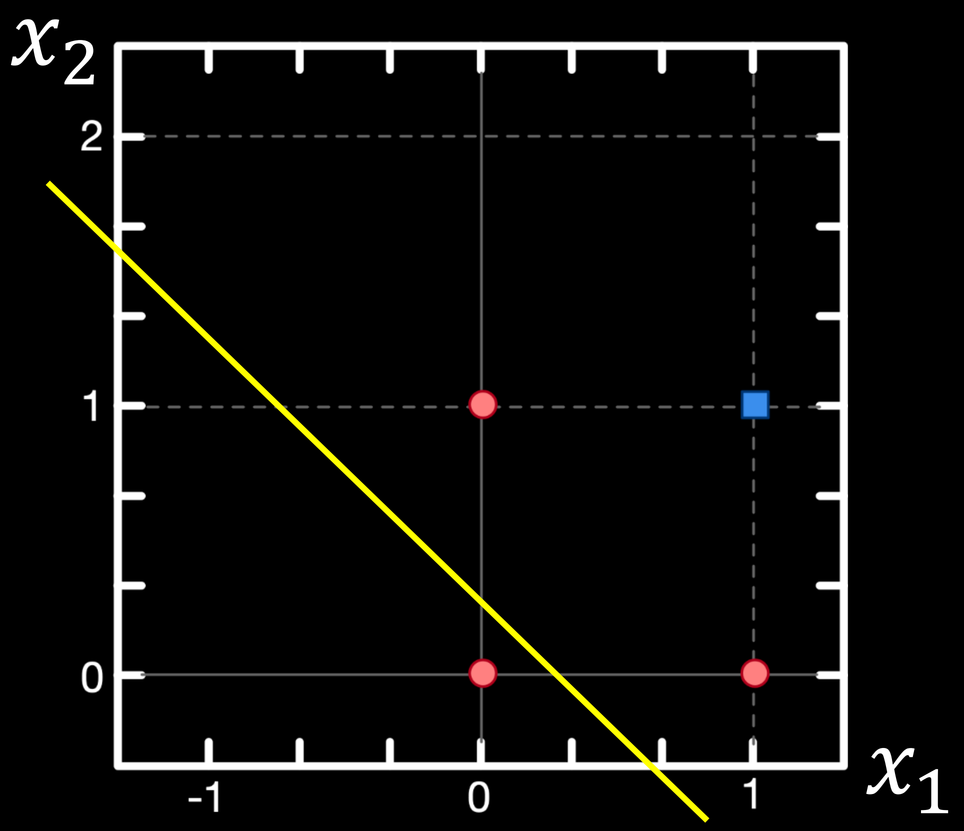
AND FUNCTION

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

$$o(t) = \begin{cases} 1 & \text{for} \quad p(t) + B \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

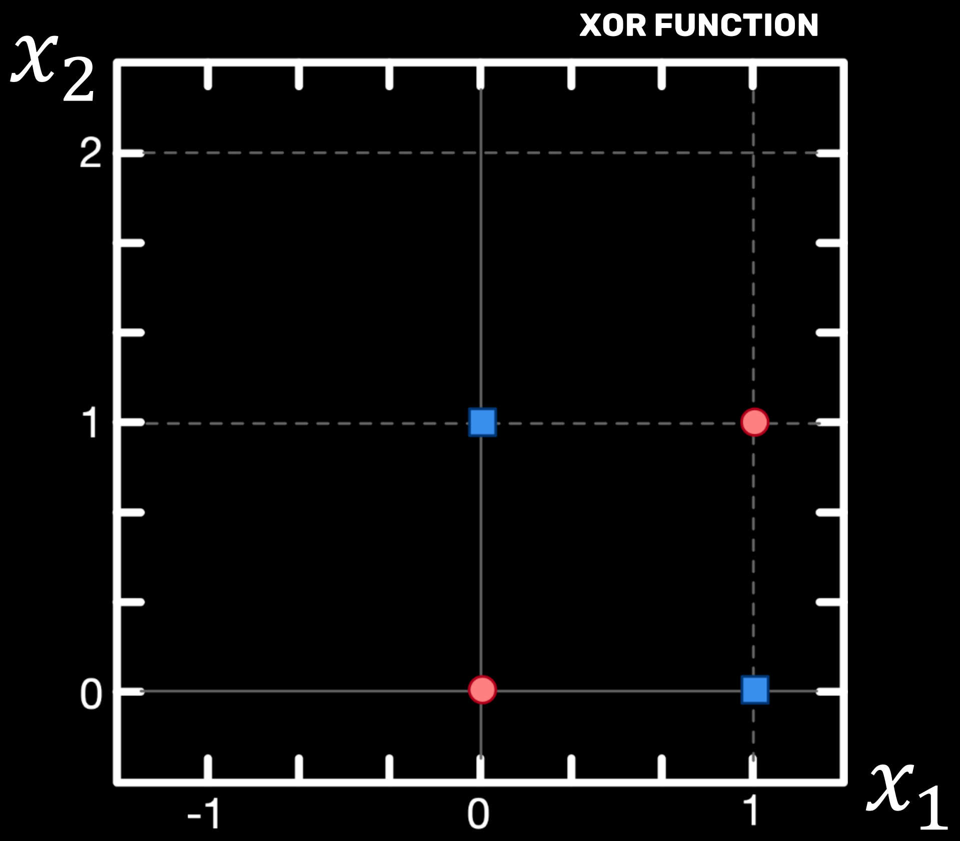
Learning in a Perceptron means to find w_1 and w_2 that minimize the error on the output.

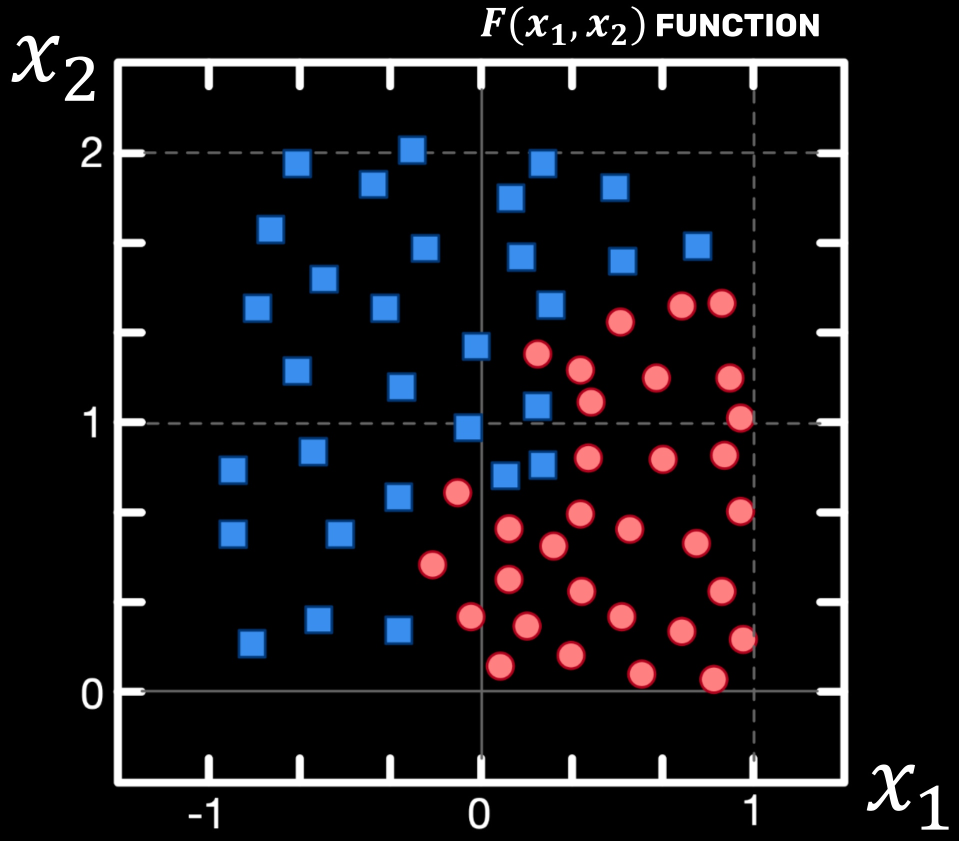




$$o(t) = B + w_1(t)x_1(t) + w_2(t)x_2(t)$$

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





SECTION 03

The Back-propagation Algorithm

In 1982 and then in 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 Back-propagation Algorithm**

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

1. **Forward propagation**: the network is **activated** on one example and the error of each neuron of the output layer is **calculated**

2. **Backward propagation**: 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

GOOD

- Proven training method for multi-layer nets
- It's able to learn any arbitrary function
- 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

BAD

- 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

Back-propagation

PROS

SECTION 04

Forward propagation

1. Apply the example X to the input neurons:

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

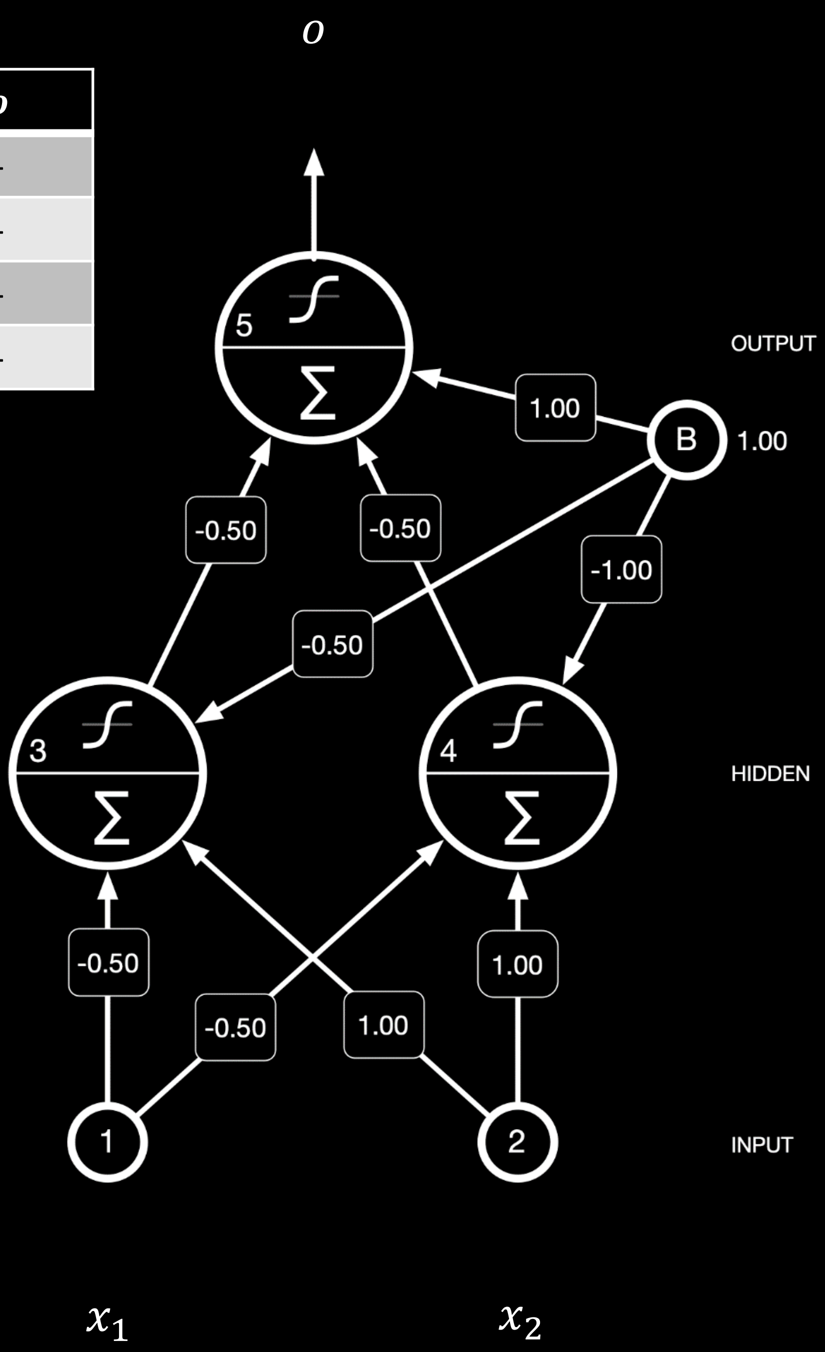
where:

- o_j : Output for neuron j
- w_{ji}: Weight from neuron j to neuron i
- N : Number of neurons of the previous layer

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

XOR FUNCTION

x_1	x_2	d	o
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	-



Forward propagation

EXAMPLE: LEARNING XOR

1. Apply the example X to the input neurons:

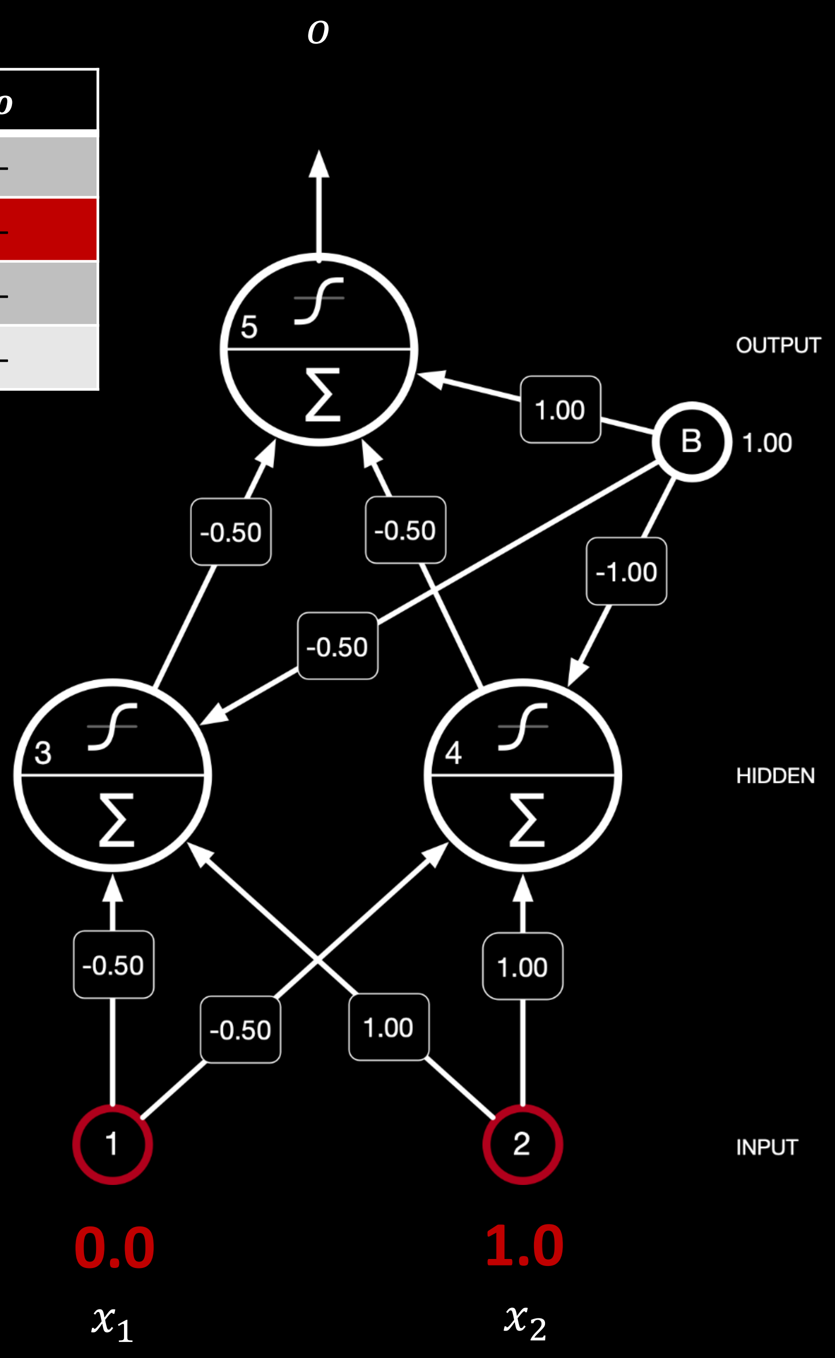
2. Calculate: $$p_i(X,W) = \sum_{j=0}^{N} w_{ji} o_j + w_i B$$

   where:
   - o_j : Output for neuron j
   - w_{ji}: Weight from neuron j to neuron i
   - N : Number of neurons of the previous layer

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

XOR FUNCTION

x_1	x_2	d	o
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	-



1. Apply the example X to the input neurons:

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

where:

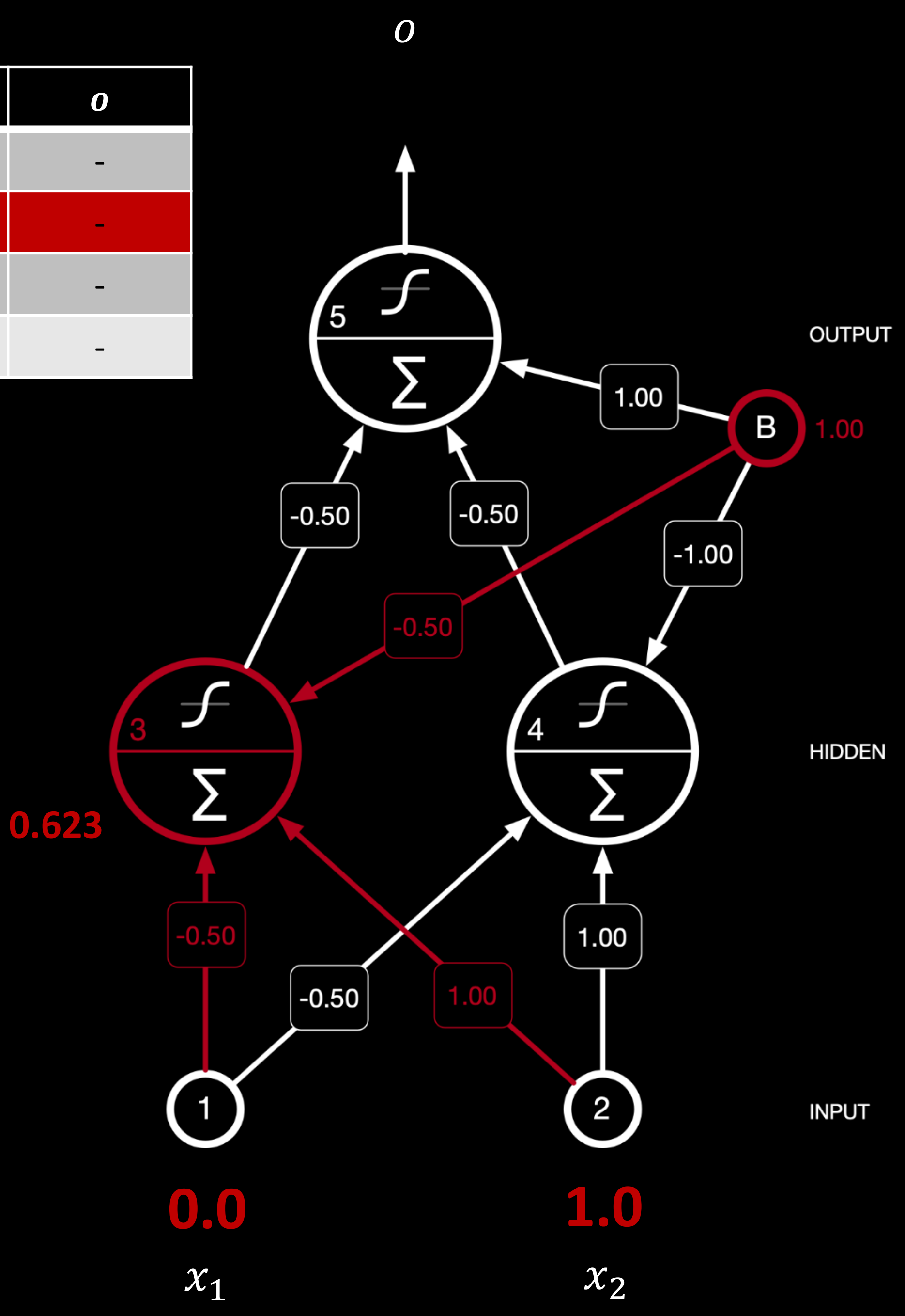
- o_j : Output for neuron j
- w_{ji}: Weight from neuron j to neuron i
- N : Number of neurons of the previous layer

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

$$O_3(-0.5 \times 0 + 1.0 \times 1 \ - 0.5 \times 1.0) = O_3(0.5) = \mathbf{0.623}$$

XOR FUNCTION

x_1	x_2	d	o
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	-



Forward propagation

EXAMPLE: LEARNING XOR

1. Apply the example X to the input neurons:

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

where:

- o_j : Output for neuron j
- w_{ji}: Weight from neuron j to neuron i
- N : Number of neurons of the previous layer

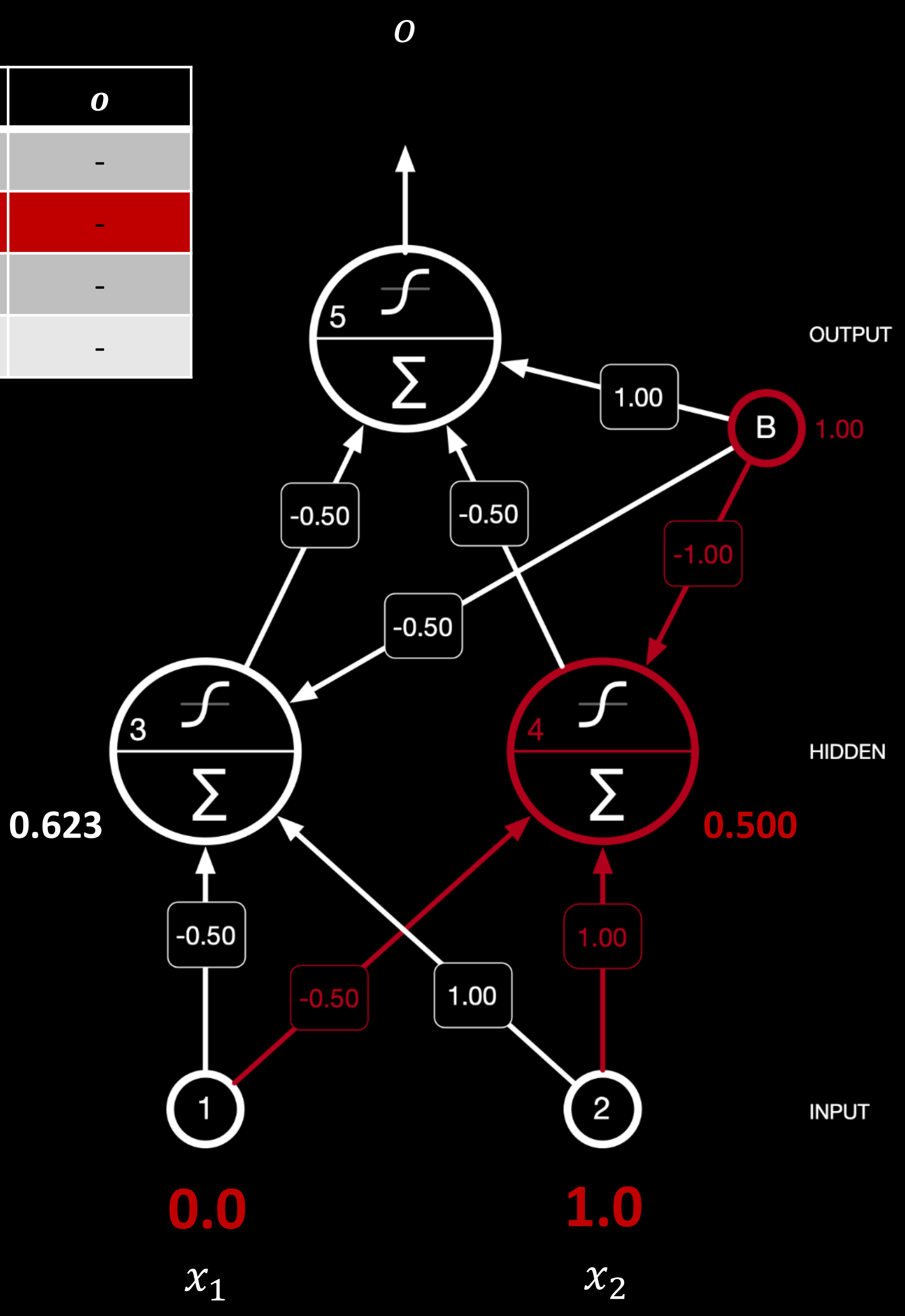
3. Calculate output activation: $O(p_i) = \dfrac{1}{1+e^{-p_i}}$

$$O_3(-0.5 \times 0 + 1.0 \times 1\ -0.5 \times 1.0) = O_3(0.5) = 0.623$$

$$O_4(-0.5 \times 0 + 1.0 \times 1\ -1.0 \times 1.0) = O_4(0.0) = 0.500$$

XOR FUNCTION

x_1	x_2	d	o
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	-



Forward propagation

EXAMPLE: LEARNING XOR

1. Apply the example X to the input neurons:

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

where:

- o_j : Output for neuron j
- w_{ji}: Weight from neuron j to neuron i
- N : Number of neurons of the previous layer

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

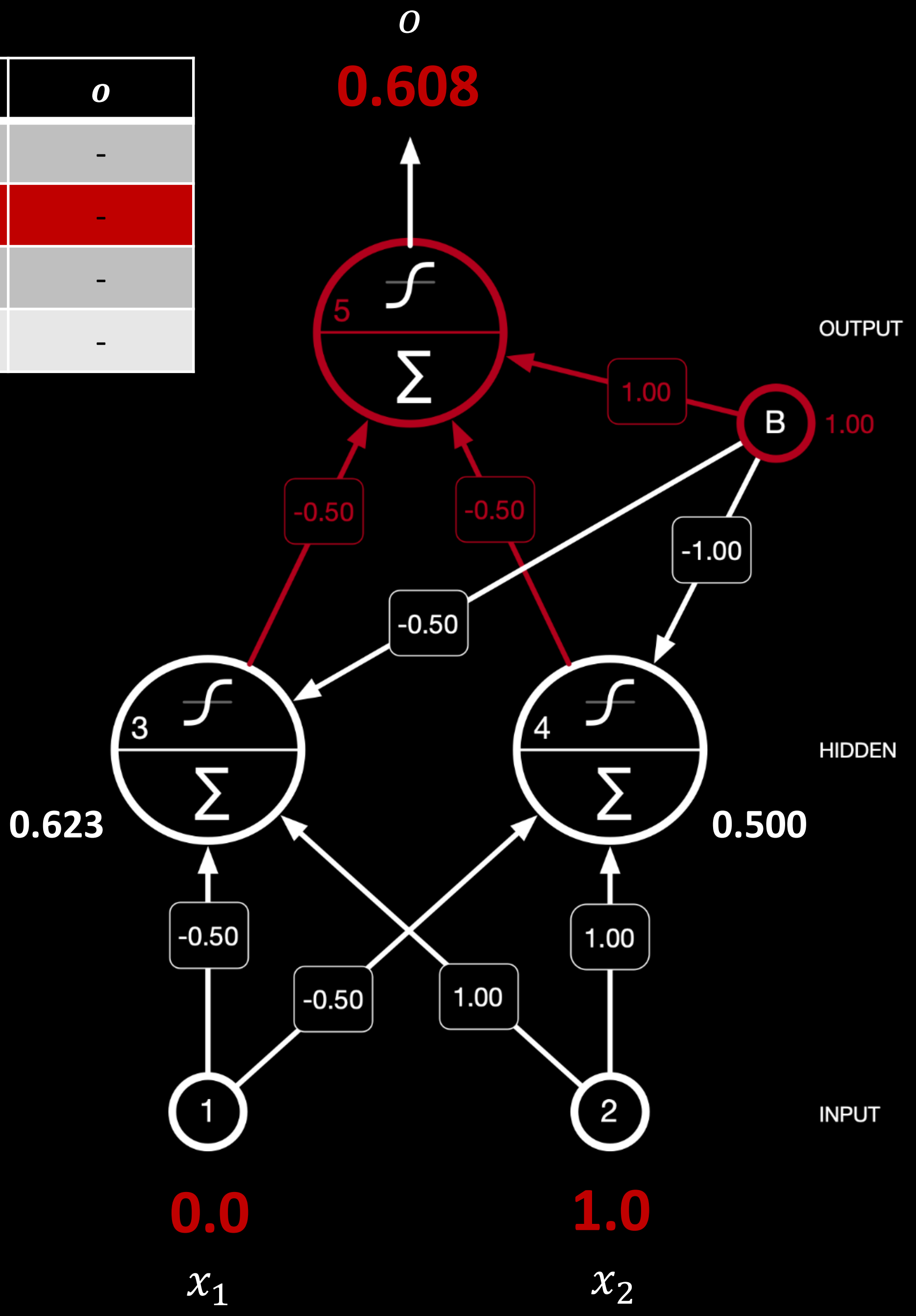
$$O_3(-0.5 \times 0 + 1.0 \times 1 \ - 0.5 \times 1.0) = O_3(0.5) = 0.623$$

$$O_4(-0.5 \times 0 + 1.0 \times 1 \ - 1.0 \times 1.0) = O_4(0.0) = 0.500$$

$$O_5(-0.5 \times 0.623 - 0.5 \times 0.500 + 1.0 \times 1.0) = O_5(0.439) = 0.608$$

XOR FUNCTION

x_1	x_2	d	o
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	-



1. Apply the example X to the input neurons:

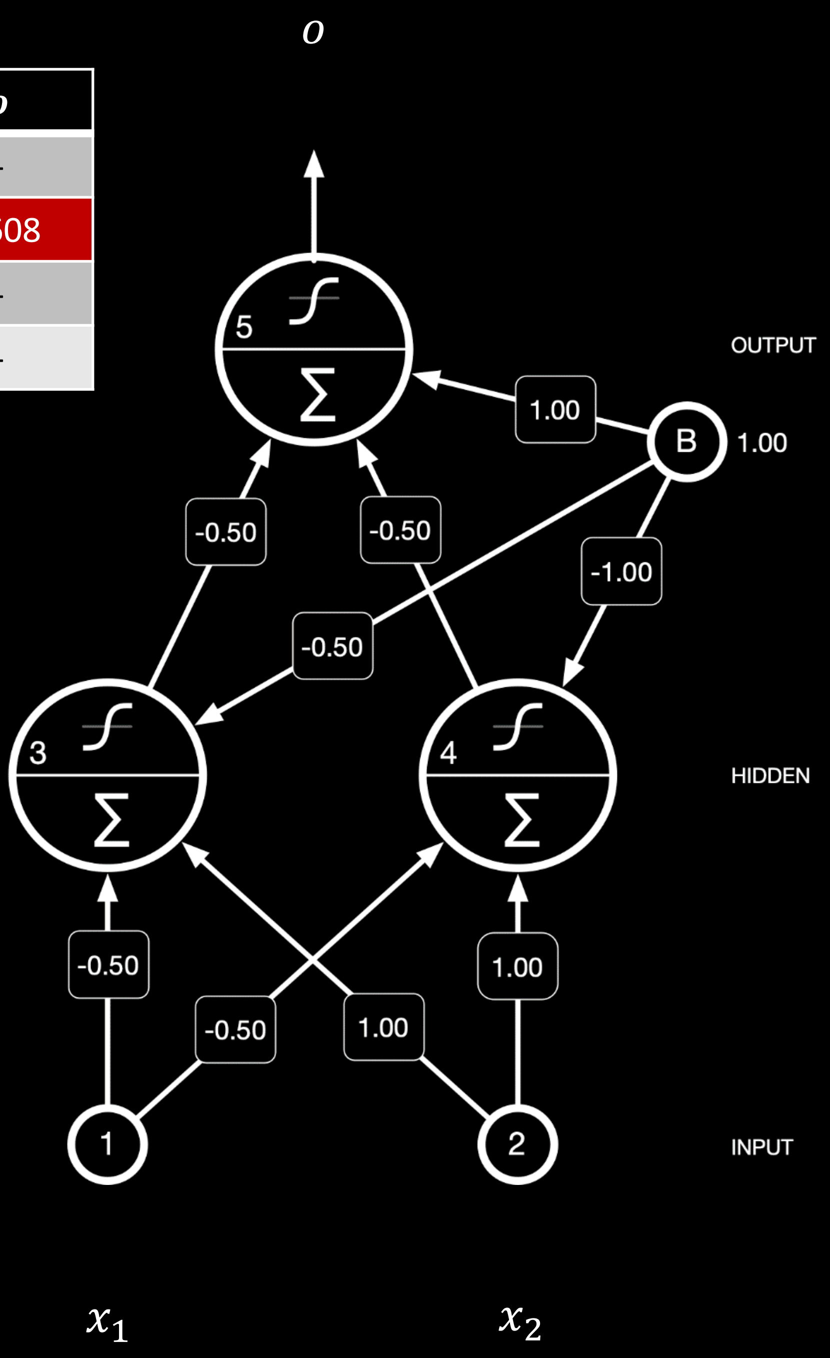
2. Calculate: $$p_i(X,W) = \sum_{j=0}^{N} w_{ji}o_j + w_i B$$

   where:
   - o_j : Output for neuron j
   - w_{ji}: Weight from neuron j to neuron i
   - N : Number of neurons of the previous layer

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

XOR FUNCTION

x_1	x_2	d	o
0	0	0	-
0	1	1	0.608
1	0	1	-
1	1	0	-



SECTION 05

Backward propagation

The **calculation of the error** is the difference between desired and actual output:

$$\delta_i(t) = o'_i\,(p_i)\,[d_i(t) - o_i(t)]$$

d_i	Target output for the neuron i
o_i	Output of the neuron i
p_i	Potential of the neuron i

Calculate the contribution to the error by each **hidden neuron**:

$$\delta_i(t) = o'(p_i) \sum_{j=1}^{M} w_{ij}(t)\, \delta_j(t)$$

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

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

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

w_{ji}	The weight from node i to node j
η	Learning rate
o_i	The output of the neuron i
δ_j	The signal error for the neuron j

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

$$MSE(t) = \frac{1}{N} \sum_{i=1}^{N} [d_i(t) - o_i(t)]^2$$

d_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:

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

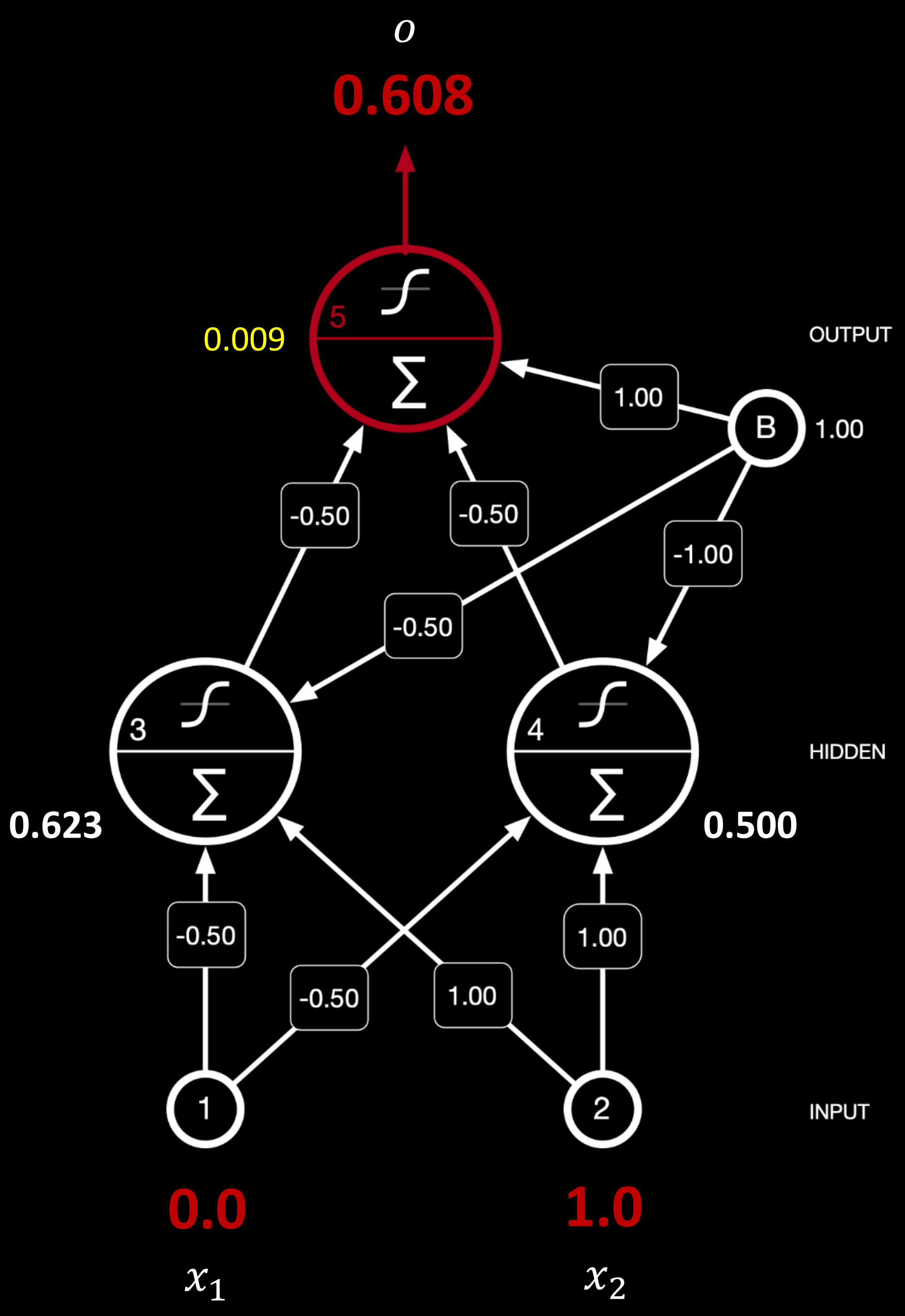
- It augments the effective learning rate η to vary the amount a weight is updated
- It can skip over small local minima

In the output layer the **calculation of error** is based on the difference between target and actual output:

$$\delta_i(t) = o'_i(t)\,[d_i(t) - o_i(t)]$$

$$\delta_i(t) = o_i(t)[1 - o_i(t)]\,[d_i(t) - o_i(t)]$$

$$\delta_5 = 0.608 \times (1 - 0.608) \times (1 - 0.608) = 0.009$$



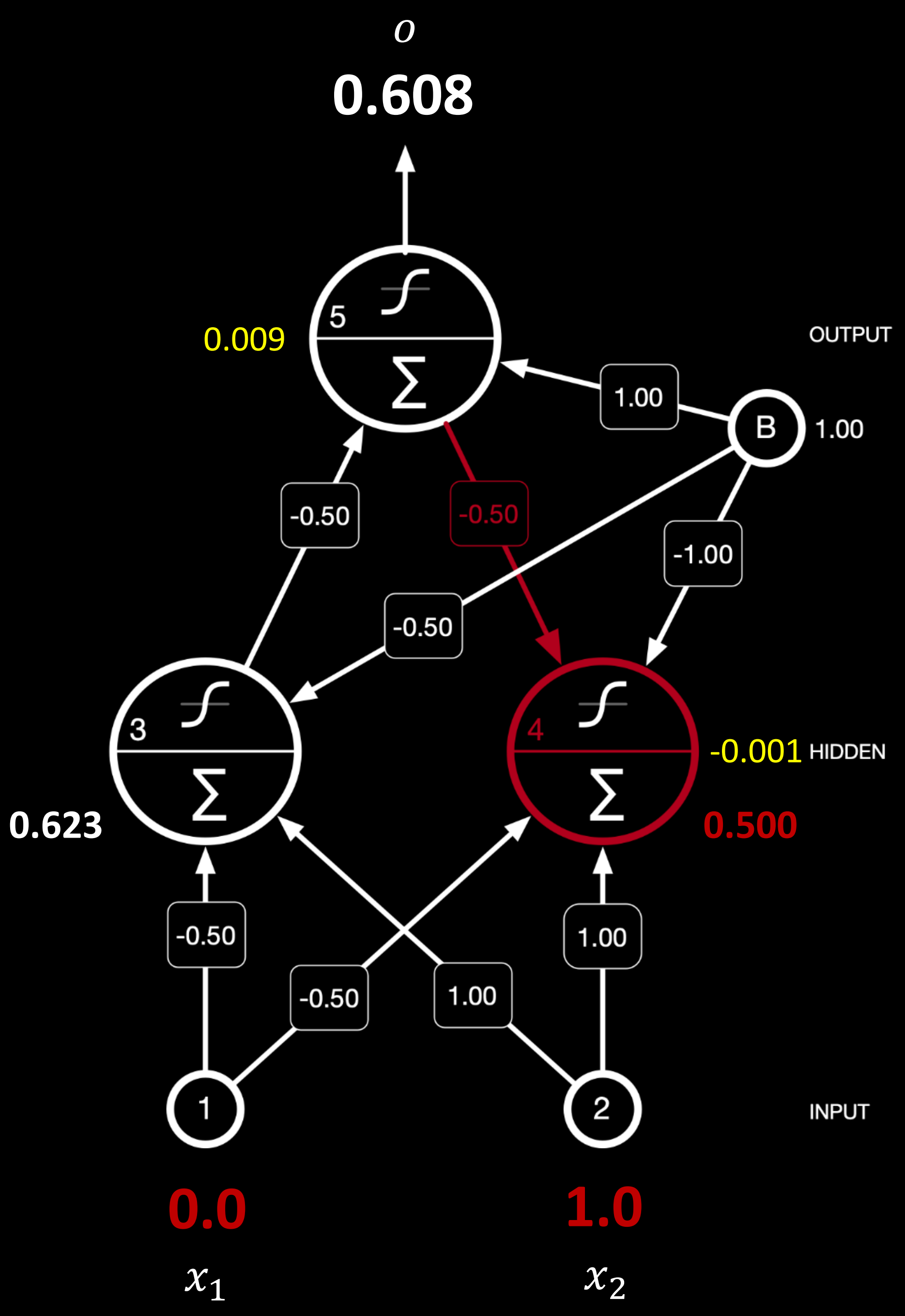
In the hidden layer the **calculation of error** is based on the contribution to the error by the hidden neuron:

$$\delta_i(t) = o'(t) \sum_{j=1}^{M} w_{ij}(t)\, \delta_j(t)$$

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

$$\delta_5 = 0.608 \times (1 - 0.608) \times (1 - 0.608) = 0.009$$

$$\delta_4 = 0.500 \times (1 - 0.500) \times (-0.50 \times 0.009) = \mathbf{-0.001}$$



In the hidden layer the **calculation of error** is based on the contribution to the error by the hidden neuron:

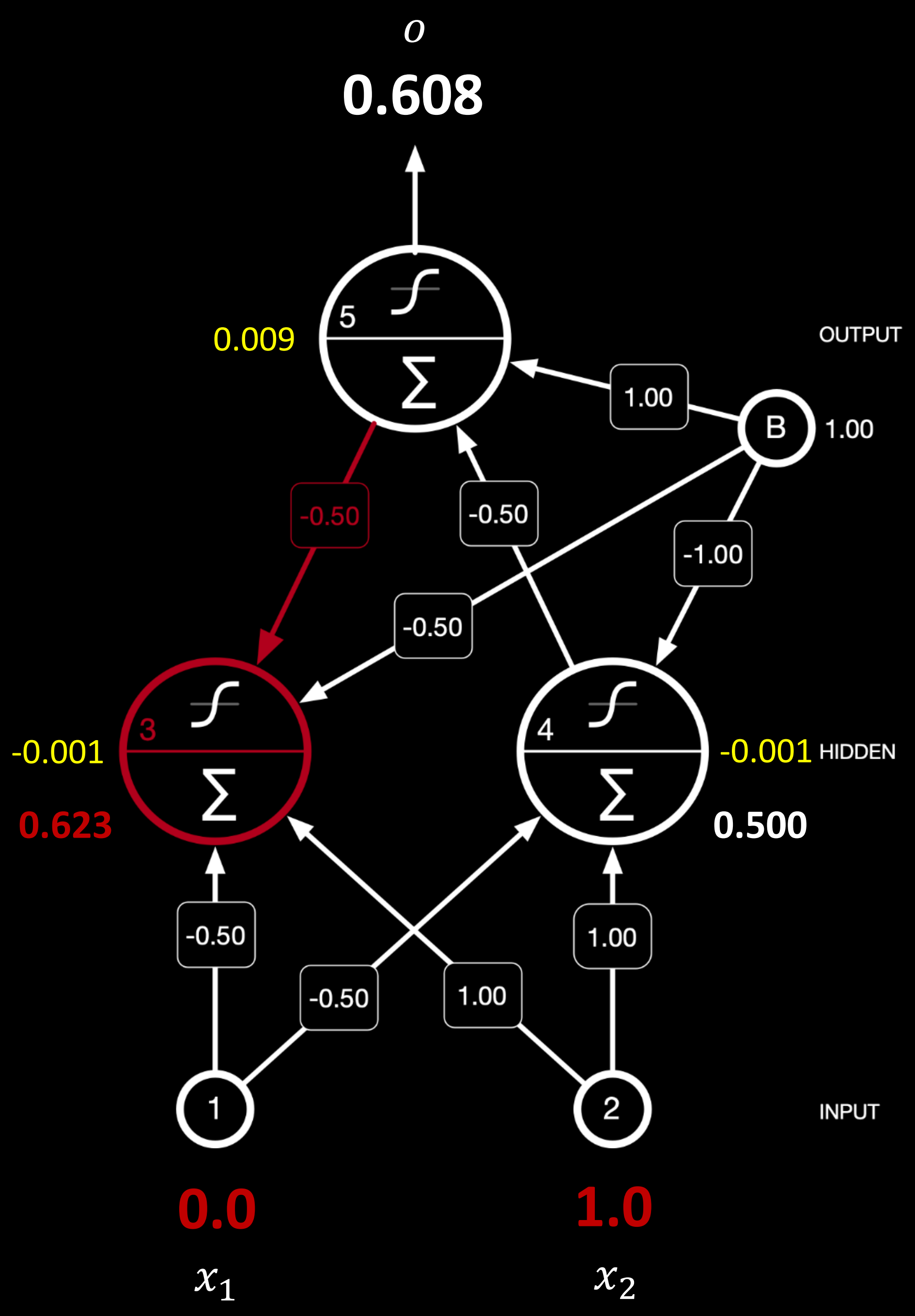
$$\delta_i(t) = o'(t) \sum_{j=1}^{M} w_{ij}(t)\, \delta_j(t)$$

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

$$\delta_5 = 0.608 \times (1 - 0.608) \times (1 - 0.608) = 0.009$$

$$\delta_4 = 0.500 \times (1 - 0.500) \times (-0.50 \times 0.009) = -0.001$$

$$\delta_3 = 0.623 \times (1 - 0.623) \times (-0.50 \times 0.009) = -0.001$$

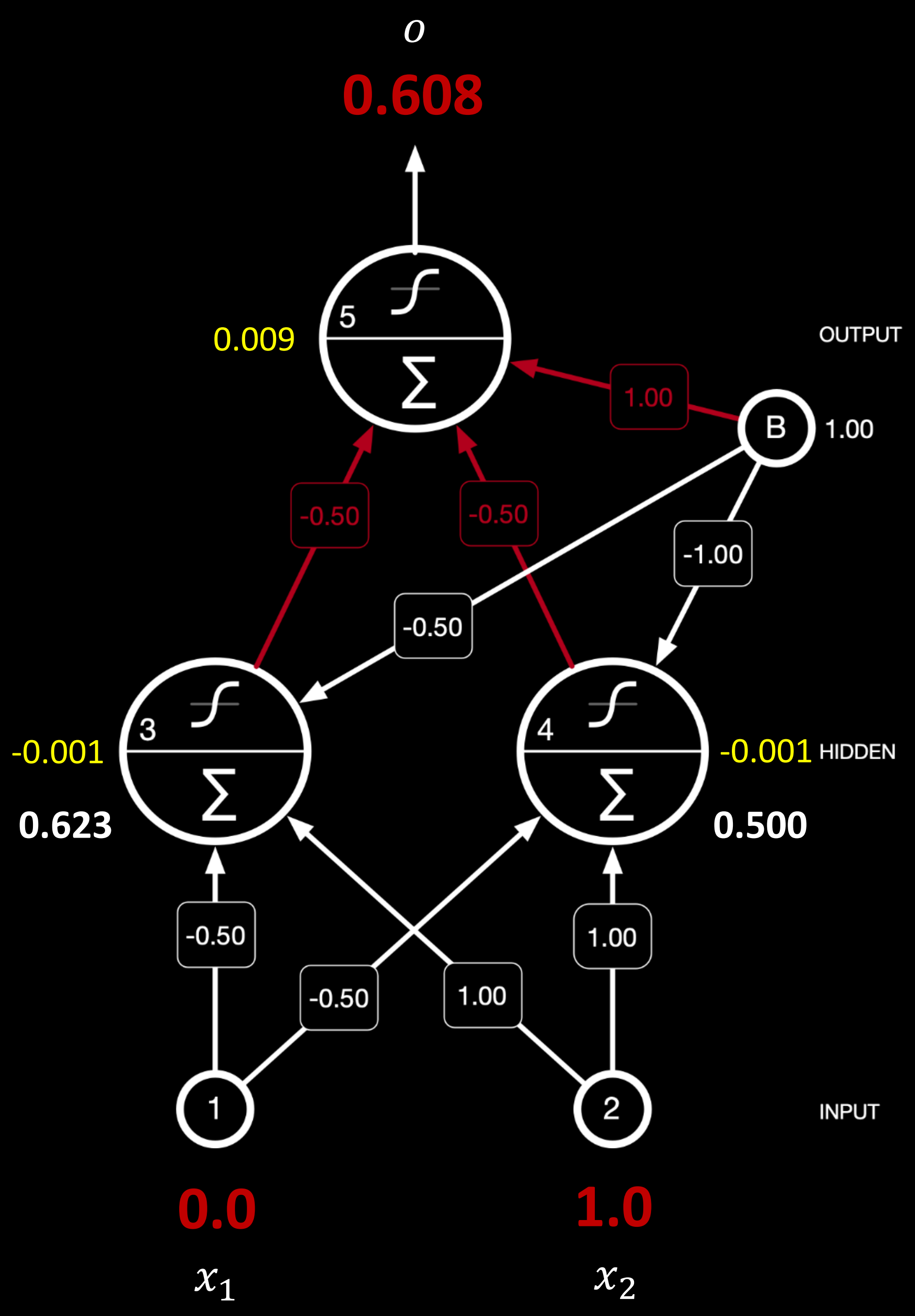


The **rate of change** of the error which is the important feedback through the network:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij}(t) = \eta\, o_i(t)\, \delta_i(t) \quad \text{where } \eta = 0.1$$

$$\Delta w_{35}(t) = \eta\, o_3(t)\, \delta_5(t) \quad \Delta w_{35} = \mathbf{0.001}$$
$$\Delta w_{45}(t) = \eta\, o_4(t)\, \delta_5(t) \quad \Delta w_{45} = \mathbf{0.0}$$
$$\Delta w_{B5}(t) = \eta\, o_B(t)\, \delta_5(t) \quad \Delta w_{B5} = \mathbf{0.001}$$



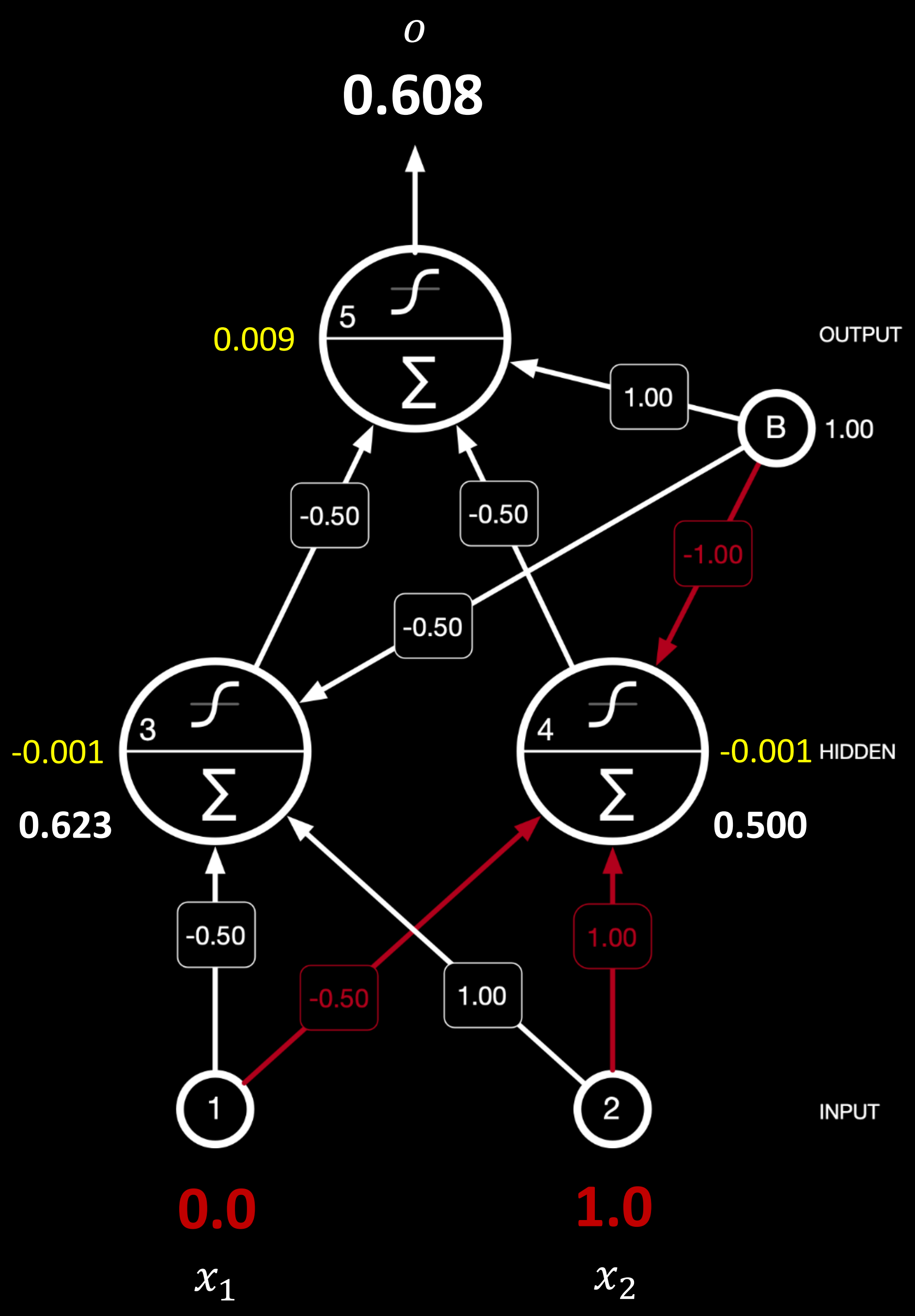
The **rate of change** of the error which is the important feedback through the network:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij}(t) = \eta\, o_i(t)\, \delta_i(t) \quad \text{where } \eta = 0.1$$

$$\Delta w_{35}(t) = \eta\, o_3(t)\, \delta_5(t) \quad \Delta w_{35} = 0.001$$
$$\Delta w_{45}(t) = \eta\, o_4(t)\, \delta_5(t) \quad \Delta w_{45} = 0.0$$
$$\Delta w_{B5}(t) = \eta\, o_B(t)\, \delta_5(t) \quad \Delta w_{B5} = 0.001$$

$$\Delta w_{14}(t) = \eta\, o_1(t)\, \delta_4(t) \quad \Delta w_{14} = \mathbf{0.0}$$
$$\Delta w_{24}(t) = \eta\, o_2(t)\, \delta_4(t) \quad \Delta w_{24} = \mathbf{0.0}$$
$$\Delta w_{B4}(t) = \eta\, o_B(t)\, \delta_4(t) \quad \Delta w_{B4} = \mathbf{0.0}$$



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

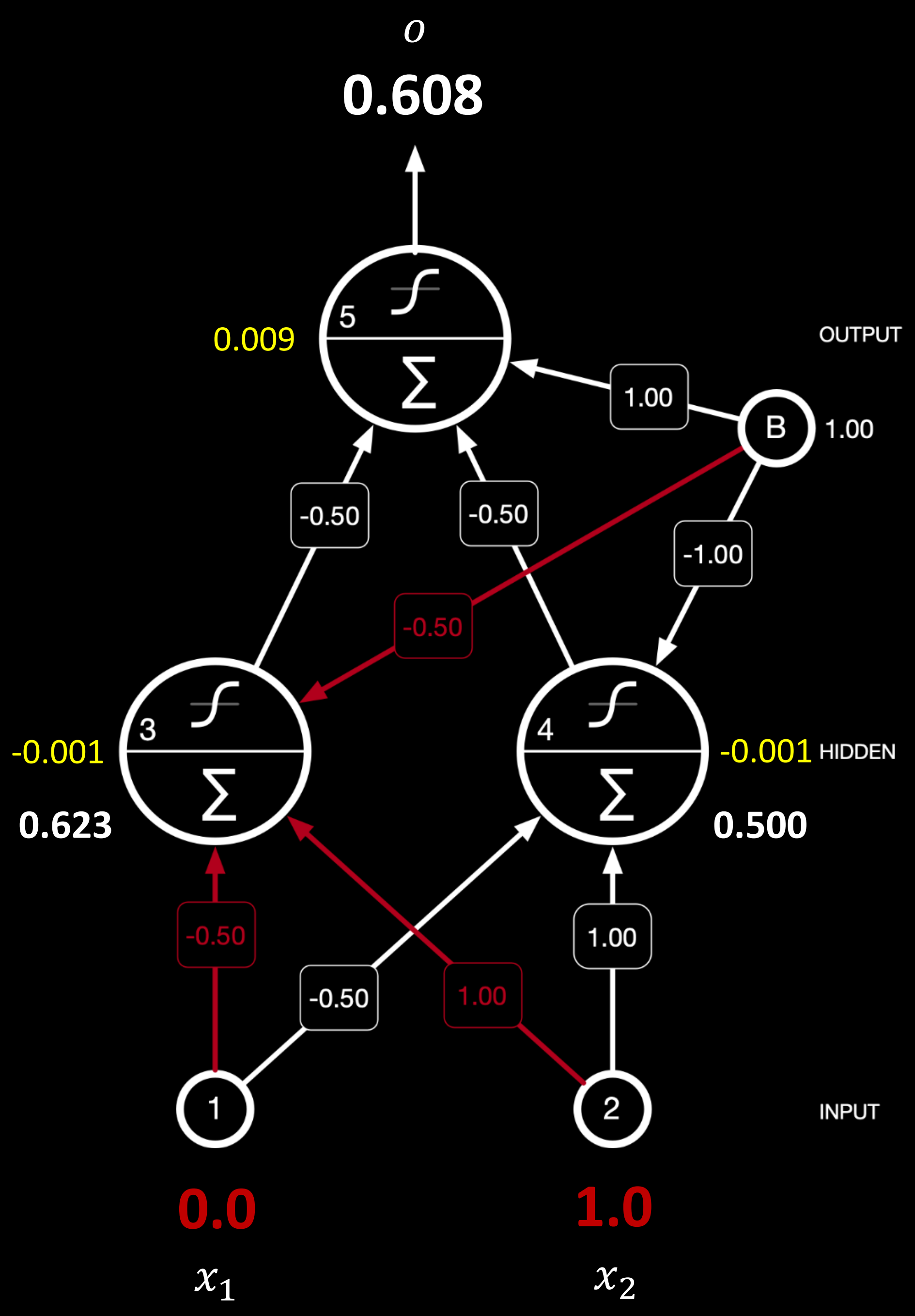
$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$\Delta w_{ij}(t) = \eta\, o_i(t)\, \delta_i(t)$ where $\eta = 0.1$

$$\Delta w_{35}(t) = \eta\, o_3(t)\, \delta_5(t) \quad \Delta w_{35} = 0.001$$
$$\Delta w_{45}(t) = \eta\, o_4(t)\, \delta_5(t) \quad \Delta w_{45} = 0.0$$
$$\Delta w_{B5}(t) = \eta\, o_B(t)\, \delta_5(t) \quad \Delta w_{B5} = 0.001$$

$$\Delta w_{14}(t) = \eta\, o_1(t)\, \delta_4(t) \quad \Delta w_{14} = 0.0$$
$$\Delta w_{24}(t) = \eta\, o_2(t)\, \delta_4(t) \quad \Delta w_{24} = 0.0$$
$$\Delta w_{B4}(t) = \eta\, o_B(t)\, \delta_4(t) \quad \Delta w_{B4} = 0.0$$

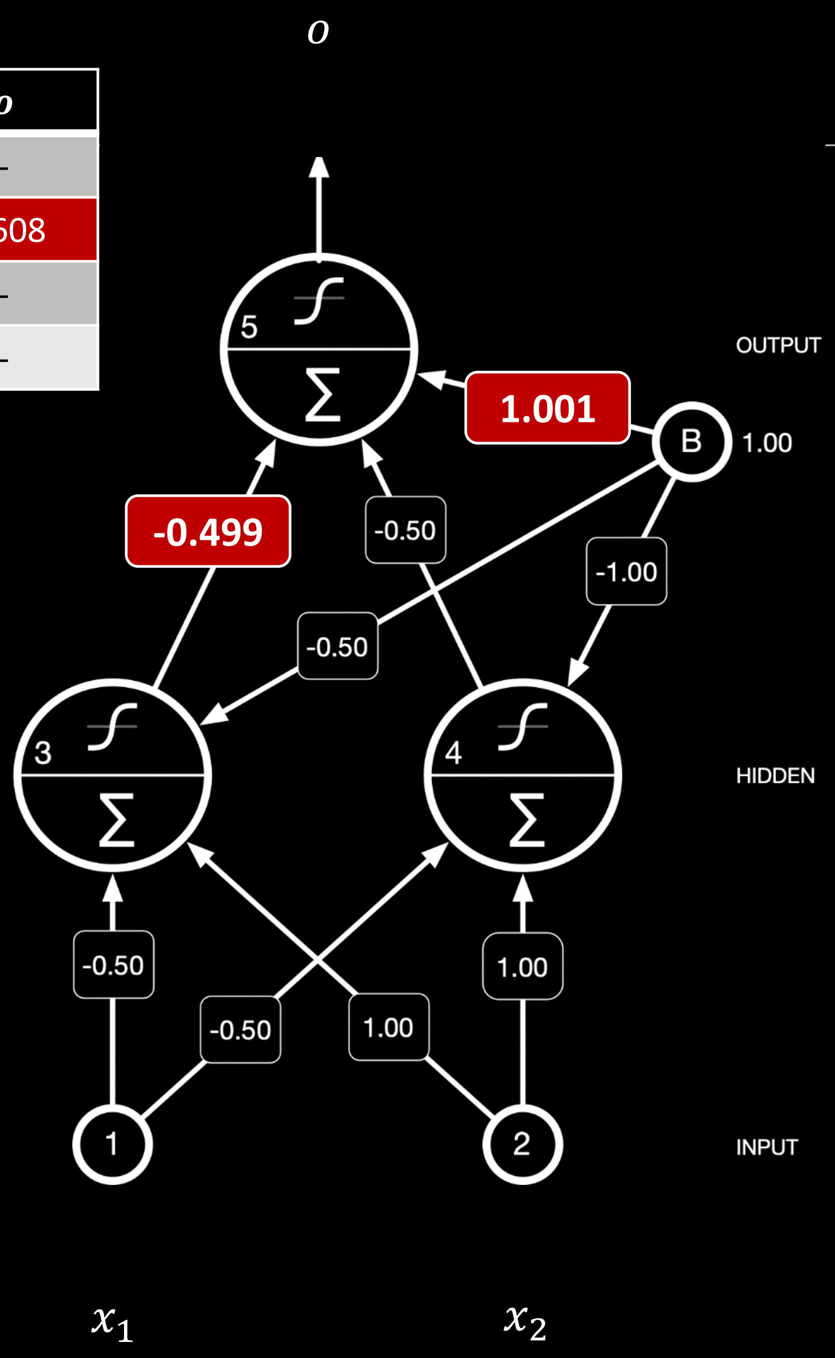
$$\Delta w_{13}(t) = \eta\, o_1(t)\, \delta_3(t) \quad \Delta w_{13} = 0.0$$
$$\Delta w_{23}(t) = \eta\, o_2(t)\, \delta_3(t) \quad \Delta w_{23} = 0.0$$
$$\Delta w_{B3}(t) = \eta\, o_B(t)\, \delta_3(t) \quad \Delta w_{B3} = 0.0$$



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

XOR FUNCTION

x_1	x_2	d	o
0	0	0	-
0	1	1	0.608
1	0	1	-
1	1	0	-



READINGS

Chapter 18.7

QUESTIONS ?

ARTIFICIAL INTELLIGENCE

COMP 131

FABRIZIO SANTINI

VERSION 3.2