Biologically Inspired Computation

Dr Marta Vallejo
m.vallejo@hw.ac.uk

Additional Material 2

Artificial Neural Networks: Initial History

- Evolution of N neural network: the “neuron”
- Neuron: the activation function
- Hebbian Learning
- The Perceptron
- Training, Learning Rule & Decision boundaries
- Widrow-Hoff Learning Rule
- Dark Era

Evolution of Neural Networks

- 1911 - Ramon y Cajal introduced the idea of neurons as structural constituents of the brain

- 1943 - McCulloch and Pitts apply Boolean algebra to nerve net behaviour

- 1948 - Donald Hebb postulates qualitative mechanism for learning at cellular level in brains

- 1957 - Rosenblatt develops '*perceptron*' neurocomputer

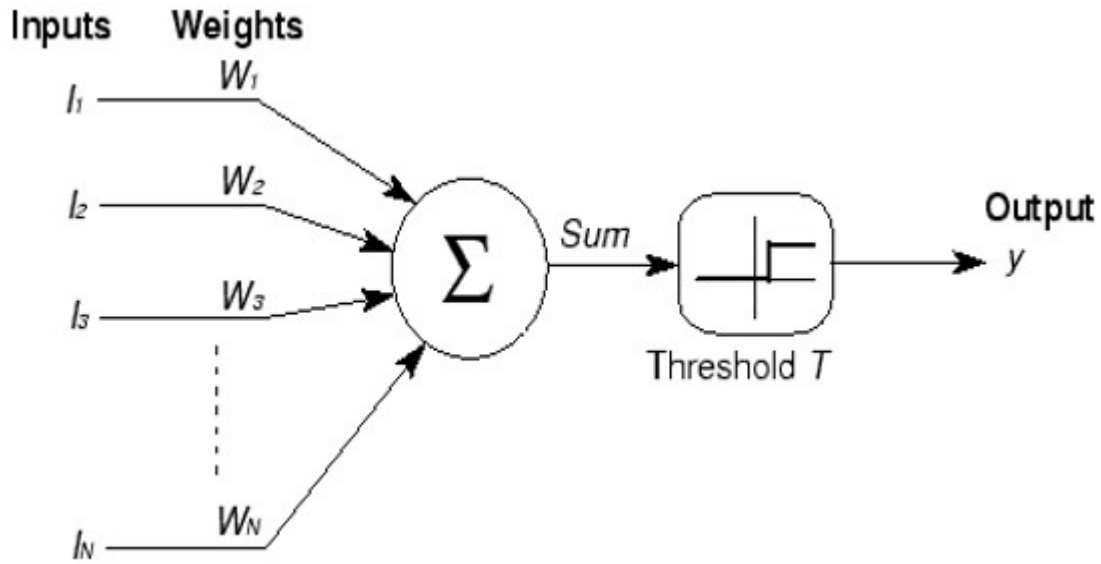
- Between 1960's & 1980's - Almost no research in ANN

- Middle 80's - John Hopfield revives ANN

- Today - ANN one of the most active current areas of research

First artificial neural network: the “neuron”

The first model of neural networks was proposed in 1943 by McCulloch and Pitts as a computational model of neural activity.



Neuron model solves simple logic and arithmetic functions, if they are **separable**.

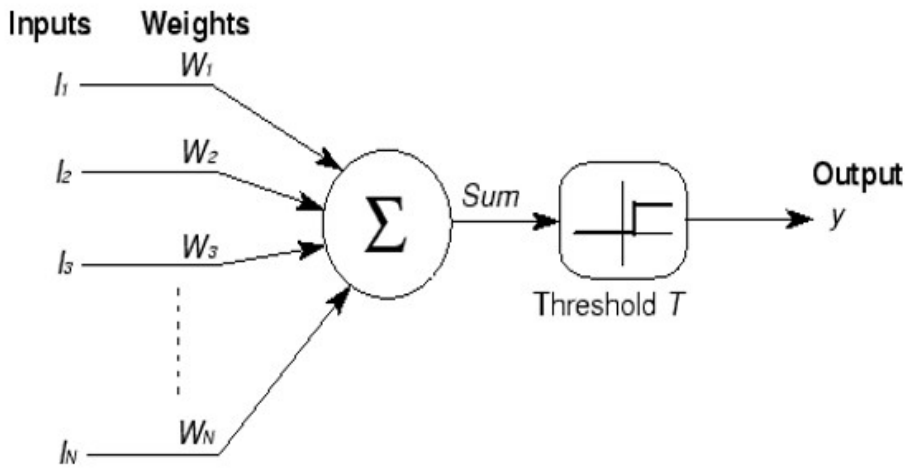
McCulloch and Pitts, “A logical calculus of the ideas immanent in nervous activity”. The bulletin of mathematical biophysics. December 1943, Volume 5, Issue 4, pp 115–133

The “neuron”

Separable logic gates: **AND, OR, NOT**

From logic we know that we can construct **any** logical function from these three basic logic gates.

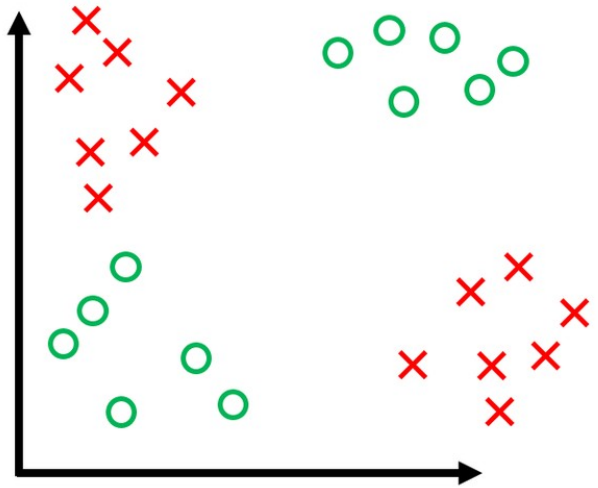
Cannot deal with XOR.



Truth table

input		output
0	0	0
0	1	1
1	0	1
1	1	0

$$y = A \otimes B$$



XOR is not a linear separable function

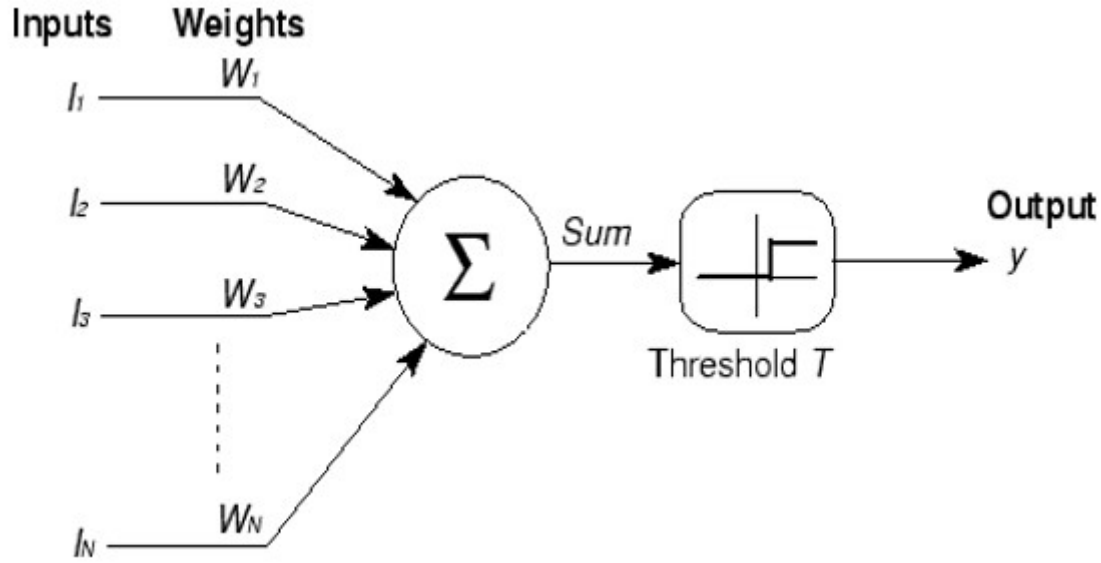
First artificial neural network: the “neuron”

Binary inputs (0) or (1), and one binary output (0) or (1).

Each input has associated with a unit weights [-1,+1].

Uses of a fixed threshold **T** that determines the output of the neuron.

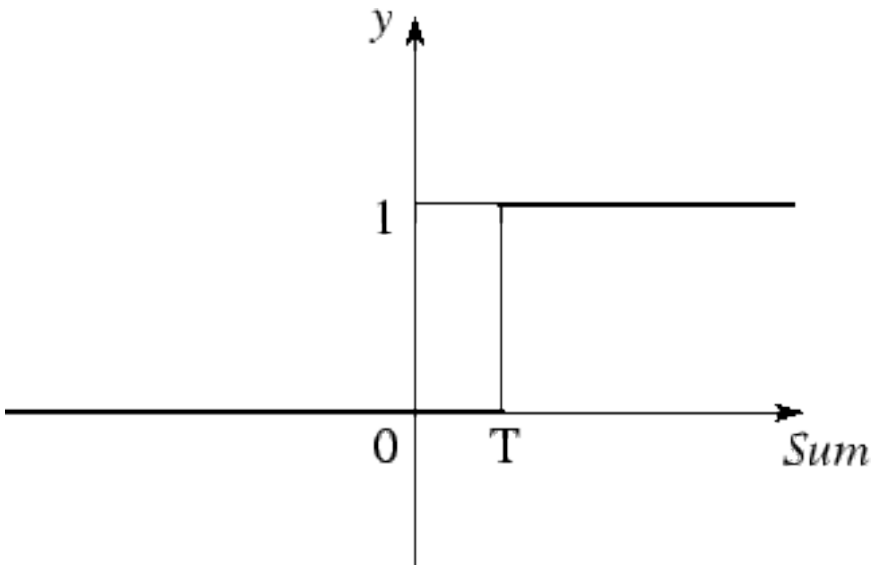
Absolute inhibition



$$\text{Sum} = \sum_{i=1}^{N} I_i W_i$$

$$y = f(\text{Sum})$$

“neuron”: The activation function



$$\text{Sum} = \sum_{i=1}^{N} I_i W_i \qquad y = f(\text{Sum})$$

$$y = \begin{cases} 1, & \text{if Sum} \geqslant T \\ 0, & \text{otherwise} \end{cases}$$

The binary step function classifies the set of inputs into two different classes.

The value of the weights form the **decision boundaries** between classes. Remember in 2D, the decision boundary is a line.

“neuron”: Example NOT gate

Goal: train a neuron to behave as a logic gate NOT of one input

As we know the behaviour of a logic gate NOT, our learning will be **supervised**

Our unknown parameters are the weights **w** and the threshold **T**

$w = ?$ $T = ?$

Truth table

input	output
1	0
0	1

Training is done by a **trial and error** method

“neuron”: Example NOT gate

Truth table

input	output
0	1
1	0

$$\text{Sum} = \sum_{i=1}^{N} I_i W_i \qquad y = \begin{cases} 1, & \text{if Sum} \geqslant T \\ 0, & \text{otherwise} \end{cases}$$

Our unknown parameters are **w** and **T:**

Attempt 1: w=1 and T=1

Propagate input = 0

$$\text{Sum} = I * W = 0 * 1 = 0 \qquad \text{is } 0 \geqslant 1 \qquad \text{no, then } y = 0$$

Is 0 our desired output for input 0?

$$\text{is } 0 \geqslant 1 \qquad \text{no, then } y = 0$$

“neuron”: Example NOT gate

Truth table

input	output
0	1
1	0

$$\text{Sum} = \sum_{i=1}^{N} I_i W_i \qquad y = \begin{cases} 1, & \text{if Sum} \geqslant T \\ 0, & \text{otherwise} \end{cases}$$

Our unknown parameters are **w** and **T:**

Attempt 1: w=1 and T=1

Propagate input = 0

$$\text{Sum} = I * W = 0 * 1 = 0 \qquad \text{is } 0 \geqslant 1 \qquad \text{no, then } y = 0$$

Is 0 our desired output for input 0?

NO, then the output is an **error**

“neuron”: Example NOT gate

Truth table

input	output
0	1
1	0

$$\text{Sum} = \sum_{i=1}^{N} I_i W_i \qquad y = \begin{cases} 1, & \text{if Sum} \geqslant T \\ 0, & \text{otherwise} \end{cases}$$

Our unknown parameters are **w** and **T:**

Attempt 1: w=1 and T=1

Propagate input = 0

$\text{Sum} = I * W = 0 * 1 = 0$ *is* $0 \geqslant 1$ *no, then* $y = 0$ **Error**

Propagate input = 1

$\text{Sum} = I * W = 1 * 1 = 1$ *is* $1 \geqslant 1$ *yes, then* $y = 1$

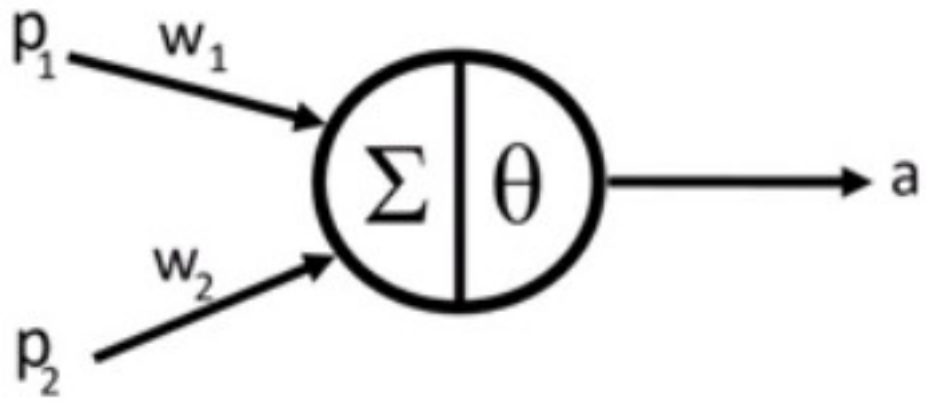
Wrong parameter values

Is 1 our desired output for input 1? No, **Error**

“neuron”: Example NOT gate

Can you try with values **w=-1** and **T=0?**

And find the values (w, T) for the OR gate?



Hebbian Learning

Synaptic plasticity: the changing in shaping and connections in the brain during learning . One of the first who study the plastic nature of the brain was the Canadian psychologist Donald Hebb

He postulated:

1.- When two neurons fire at the same time, the connections between them are strengthen, then become more likely that fire again in the future

2.- When firing uncoordinated , the connections between them are weaken, then more likely they behave individually

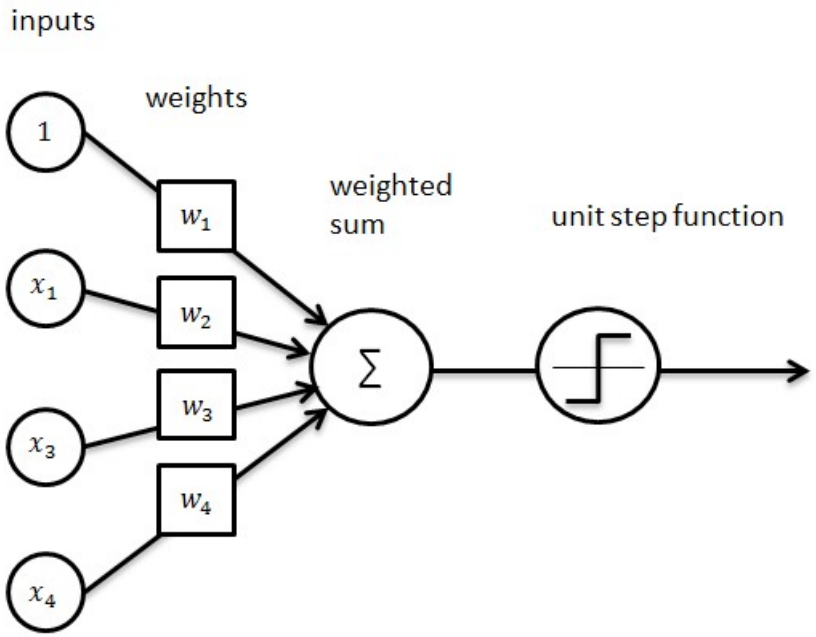
"Cells that fire together, wire together"

Hebb, Donald (1949) "The organization of behavior", New York, Wiley.

The Perceptron

The following differences from the McCullough-Pitts neuron:

- The weights and thresholds can be determined analytically or by a learning algorithm.
- There is no absolute inhibitory synapse.
- Although the neurons were still two-state, T goes from [-1,1], not [0,1].



Rosenblatt, Frank. (1958), "The perceptron: A probabilistic model for information storage and organization in the brain". Psychological Review, v65, n6, pp:386-408

The Perceptron: Training

Independently of the level of complexity of the neural network, it is necessary to develop a systematic procedure for learning the appropriate weights.

The common procedure is using a set of **training data** (supervised learning).

Rosenblatt's **key contribution** was the introduction of a learning rule for training perceptron networks to solve pattern recognition problems

He proved that his learning rule always **converge** to the correct set of weights

General Training Algorithm

1.-Generate a training dataset (input/output):

$$X = [x_1, x_2, \ldots x_n] \qquad y_{target} = (given/known)$$

2.-Then, present the network with X and allow it to generate an output y

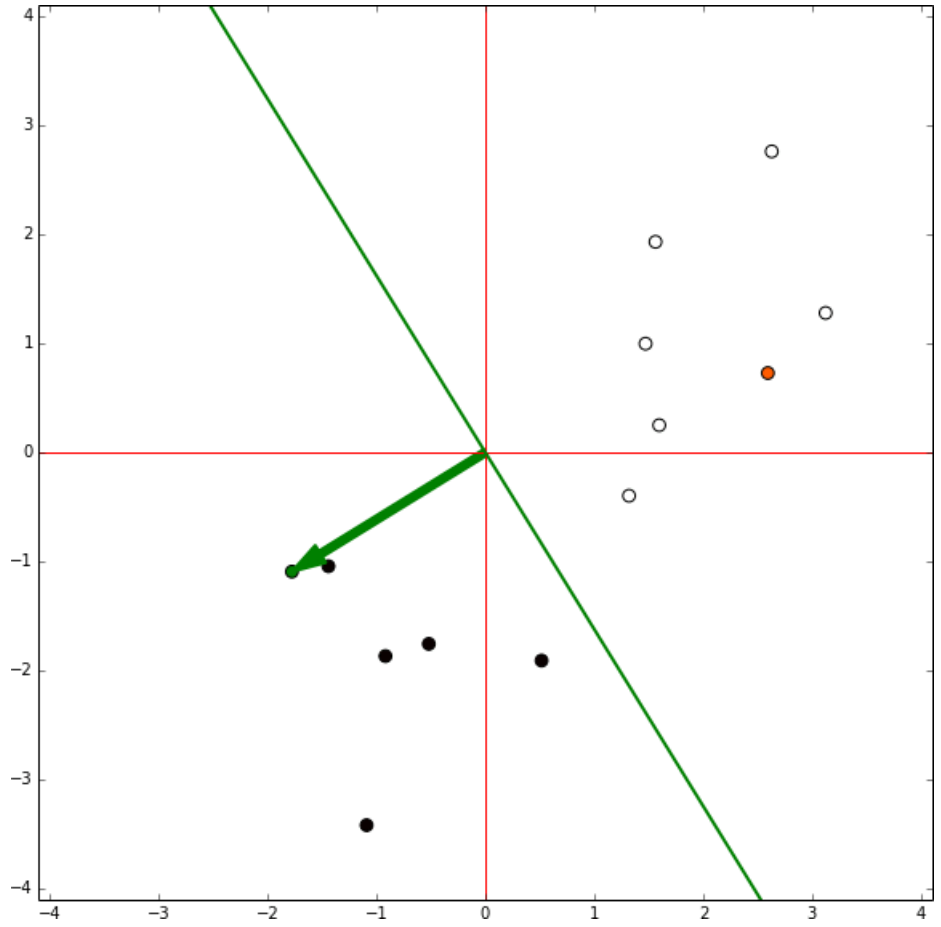
3.-Compare y with y_{target} to compute the error

4.-Adjust weights w, to reduce error

5.-Repeat 2-4 multiple times

The Perceptron: Training

A simple Perceptron uses **decision boundaries** (lines or hyperplanes), which are shifted around until each training pattern is correctly classified.



Perceptron Learning Rule

1.-Initialize weights at random

2.-For each training pair/pattern (X, y_{target})

- compute output y
- compute error $\delta = (y_{target} - y)$
- Use the error to update weights as follows:

$$w_{new} = w_{old} + \eta * \delta * x$$

where η is called the **learning rate** or **step size** and it determines how smoothly the learning process is.

3.-Repeat 2 until convergence (i.e. error δ is zero)

Perceptron: Decision boundaries

If **T** is the threshold value and the value of the neuron is:

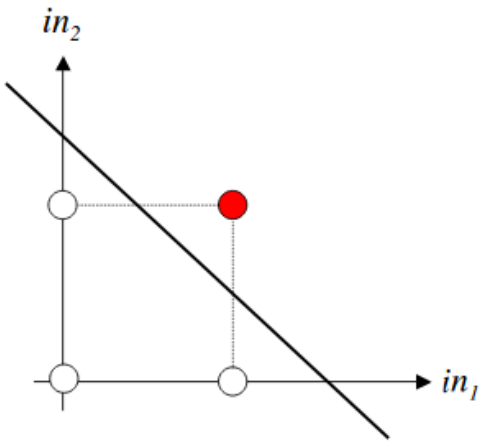
$$\text{Sum} = \sum_{i=1}^{N} I_i W_i$$

For two inputs, the decision boundary (between y = 0 and y = 1) is at

$$I_1 W_1 + I_2 W_2 - T = 0$$

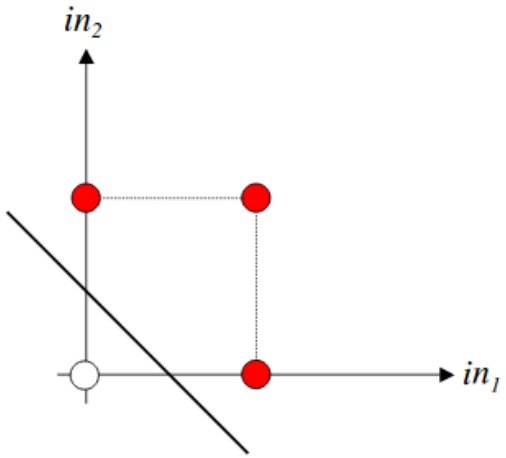
AND

$w_1 = 1,\ w_2 = 1,\ \theta = 1.5$



OR

$w_1 = 1,\ w_2 = 1,\ \theta = 0.5$



Widrow-Hoff Learning Rule

Also called LMS (Least Mean Square) learning rule

Supervised learning method which is independent of the activation function of the neuron

Goal: Minimise the squared error between the desired output value d_i and the neuron active value $w_i^t X$

$$r = d_i - w_i^t X$$

To update the weights

$$w_{new} = w_{old} + c * r * x$$

Considered a special case of the delta learning rule when

$$f\left(w_i^t X\right) = w_i^t X$$

Widrow and Hoff (1960), "Adaptive Switching Circuits", IRE WESCON convention record, pp: 96-104.

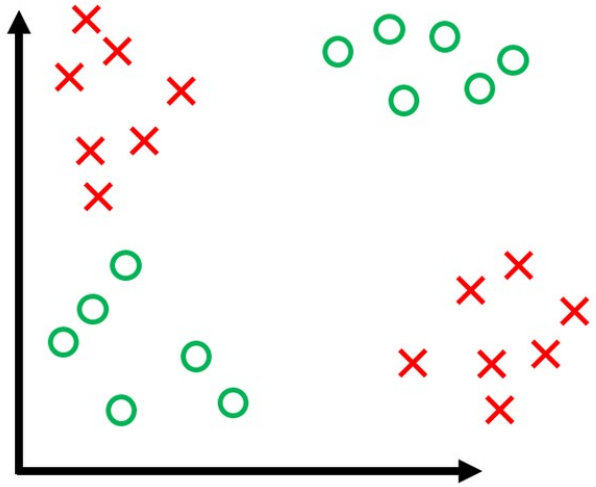
“Dark Era”

Marvin Minsky and Seymour Papert (1969) “Perceptrons, an introduction to Computational Geometry”, MIT, Cambridge, MA.

This book marked the beginning of the “dark era” in ANN research – the field stagnated for more than 10 years.

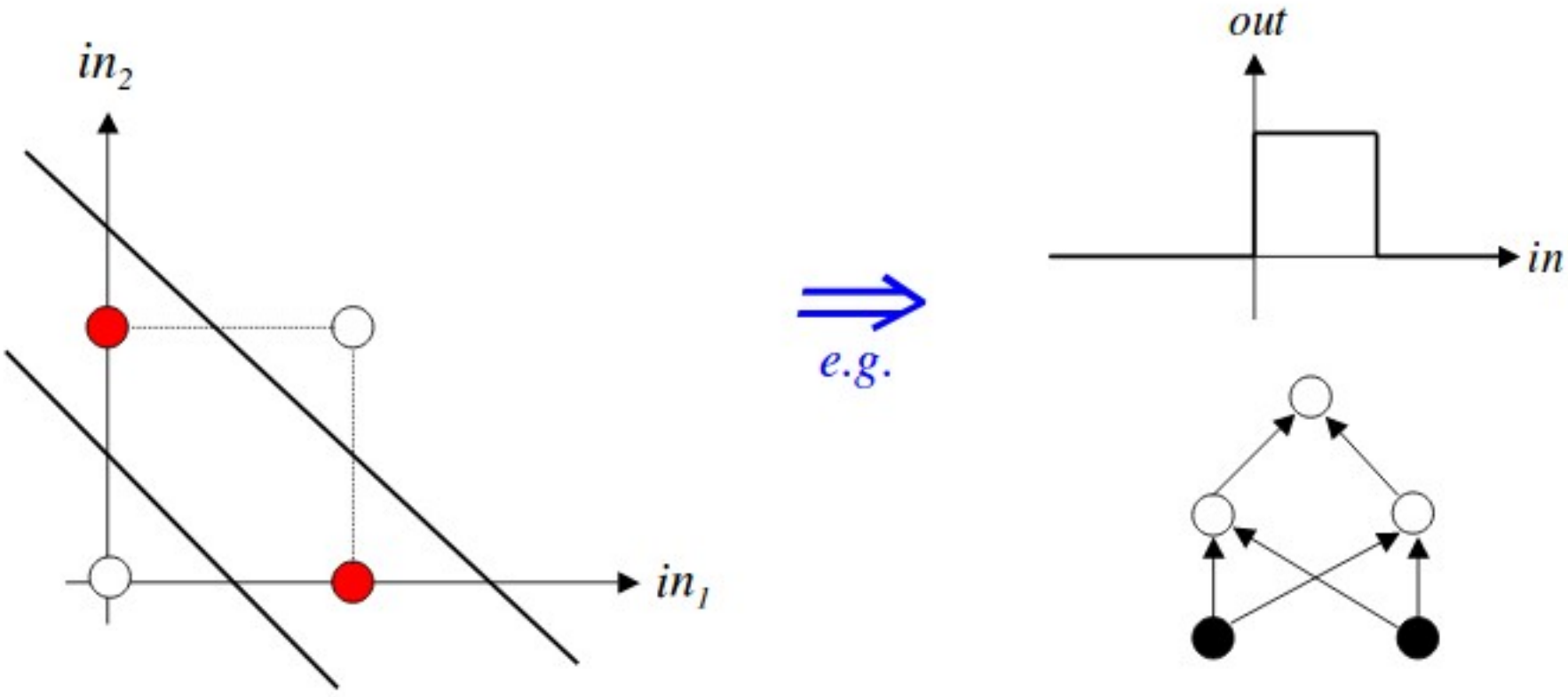
Highlighted the theoretical limitation of the Perceptron at that time: linear separability problem, e.g the inability to solve the XOR problem.

A few hardy pioneers continued.



Perceptron: Decision boundaries

But, what is the decision boundary appropriate for XOR?



We need to generate two decision boundaries, but how?

Reborn

1974: Werbos came to introduce a so-called backpropagation algorithm for the three-layered perceptron network.

1982: Hopfield brought out his idea of a recurrent neural network. Unlike the neurons in MLP, the Hopfield network consists of only one layer whose neurons are fully connected with each other.

1982: A totally unique kind of network model is the Self-Organising Map (SOM) that was introduced by Kohonen.

1986: The application area of the MLP networks remained rather limited until the breakthrough when a general back propagation algorithm for a multi-layered perceptron was introduced by Rummelhart and Mclelland

Since then, research on artificial neural networks has remained active, leading to many new network types