

A brief introduction to Weightless Neural Systems

I. Aleksander

Imperial College London - UK

M. De Gregorio

Ist. di Cibernetica "E. Caianiello" - CNR - IT

F.M.G. França, P.M.V. Lima

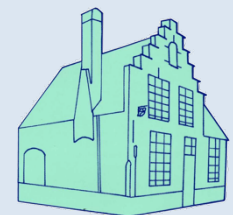
Universidade Federal do Rio de Janeiro - BR

H. Morton

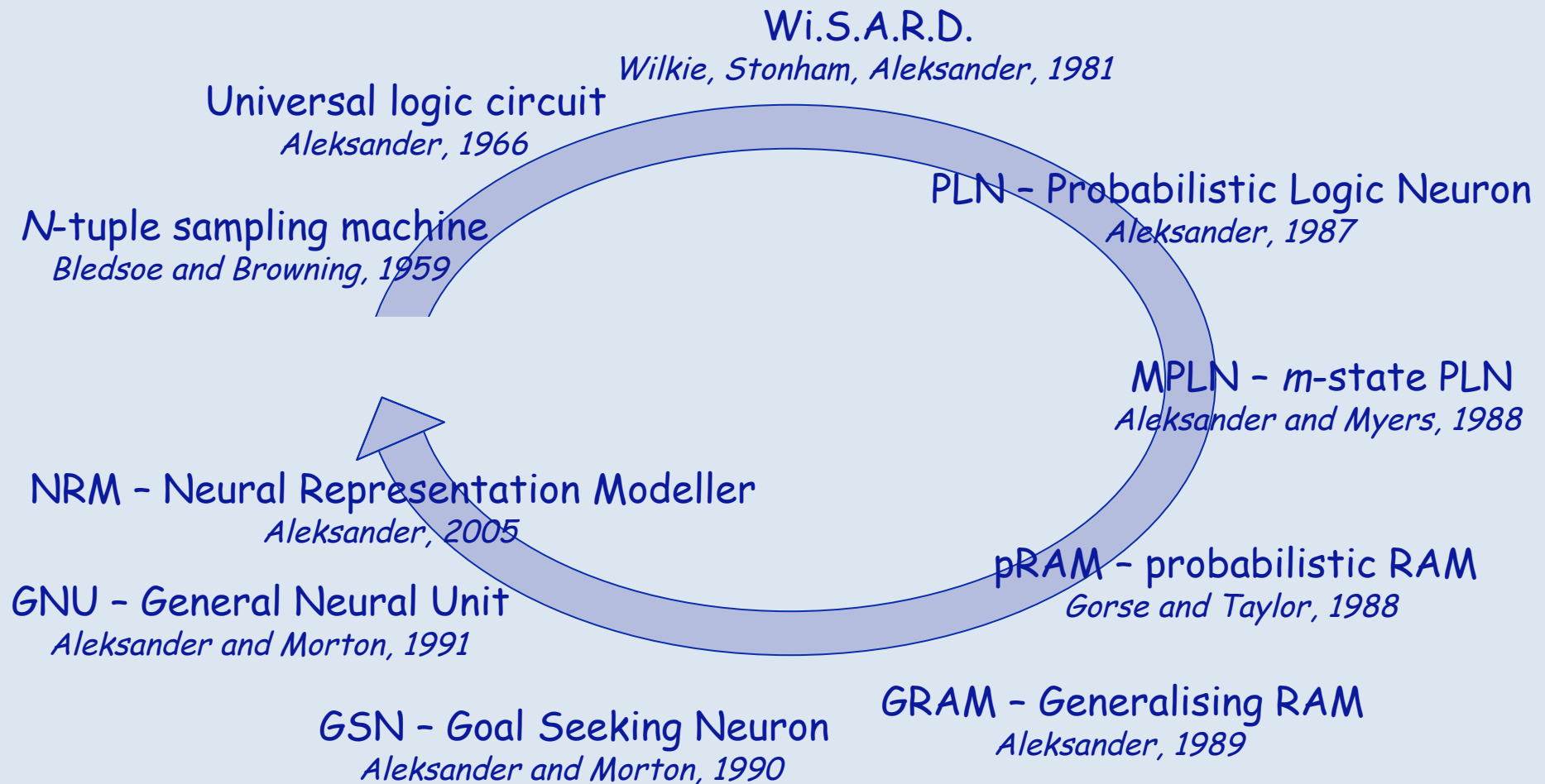
Brunel University Uxbridge, Middlesex - UK

ESANN "2009

Bruges - April 22-23-24, 2009



Chronological excursus

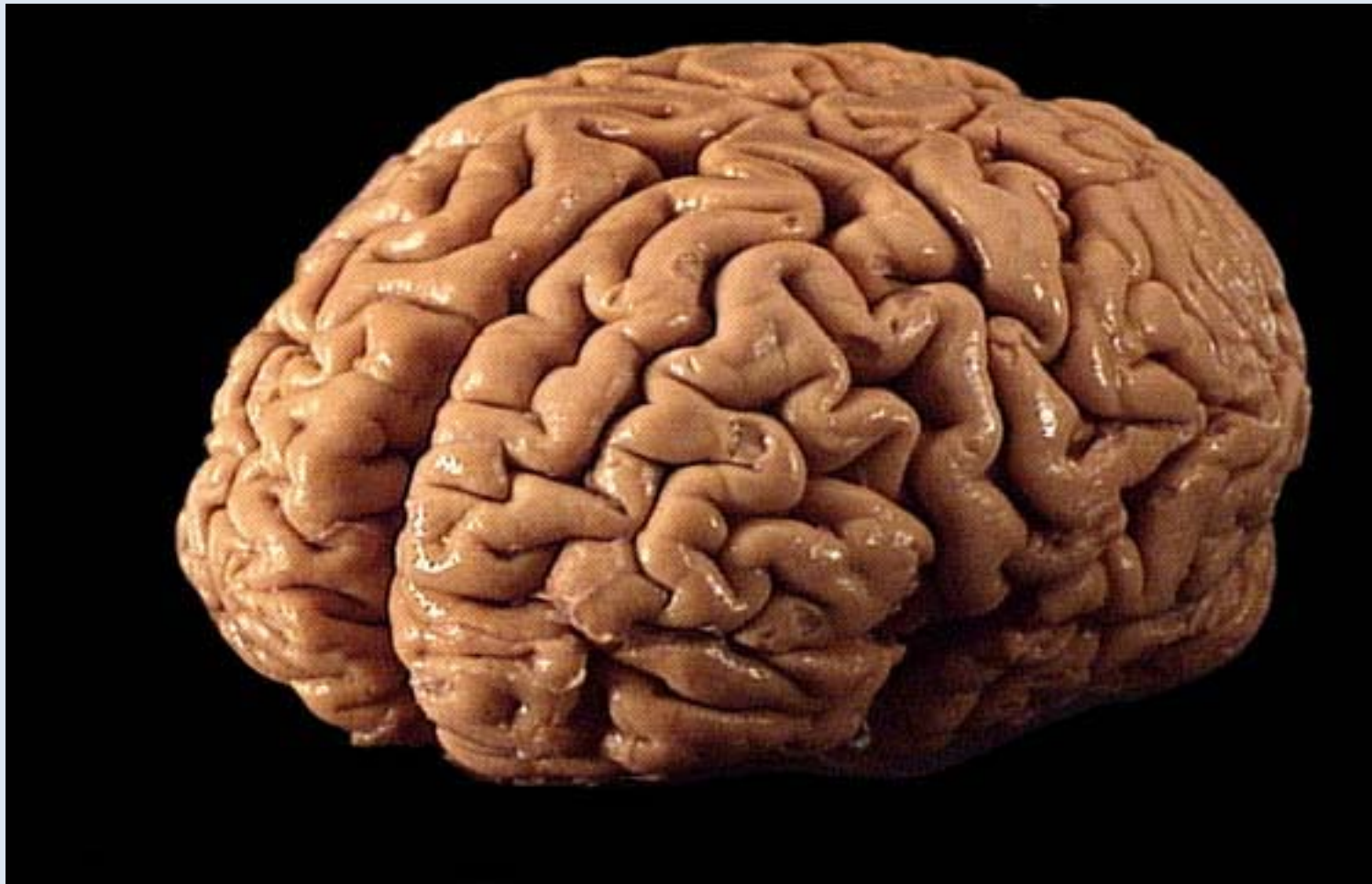


Main characteristics of WNS

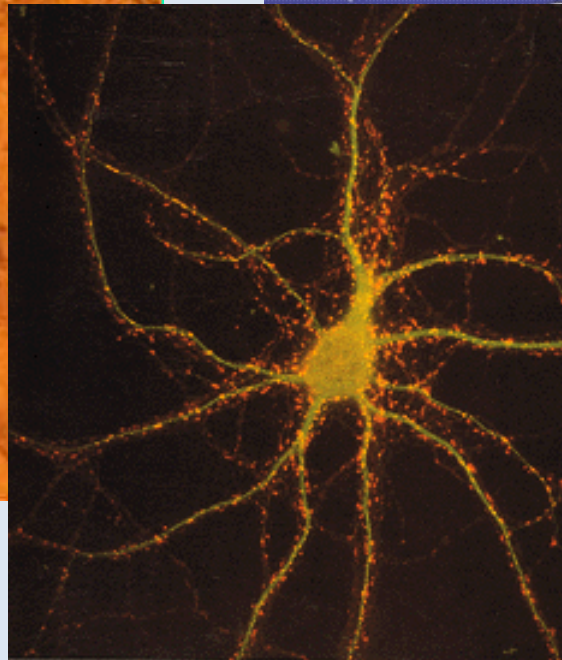
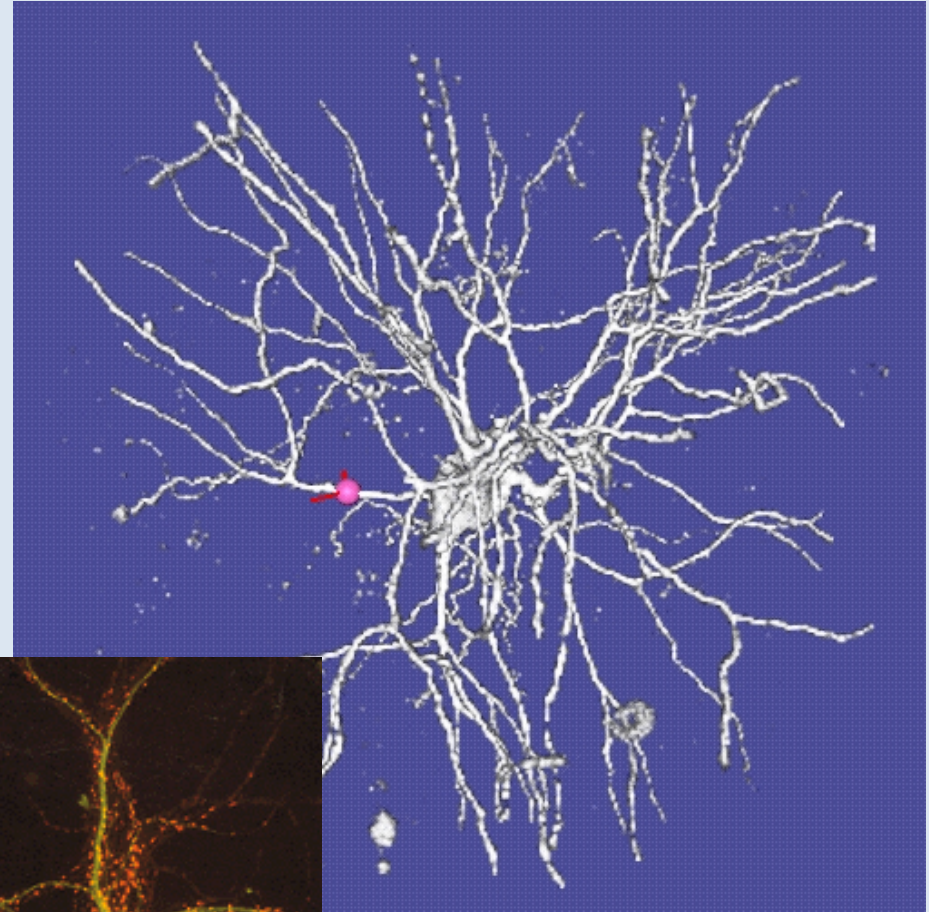
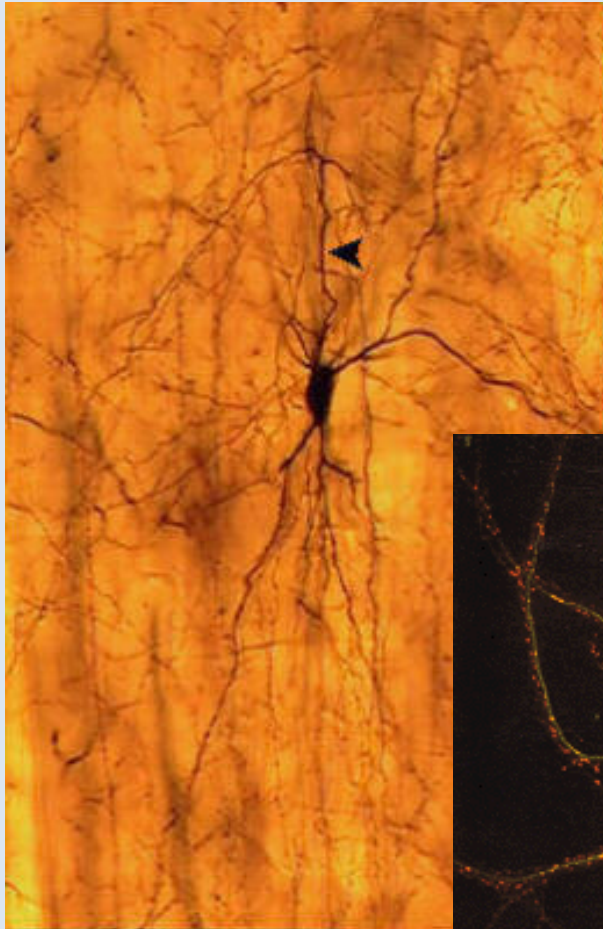
- ✓ Binary inputs and outputs and no weight between nodes
- ✓ Neuron functions are stored into look-up tables (RAM)
- ✓ Learning consists of changing the contents of look-up table entries
- ✓ Both supervised and unsupervised learning techniques
- ✓ Mutual independence between nodes (high speed learning process)
- ✓ Easy parallel implementation



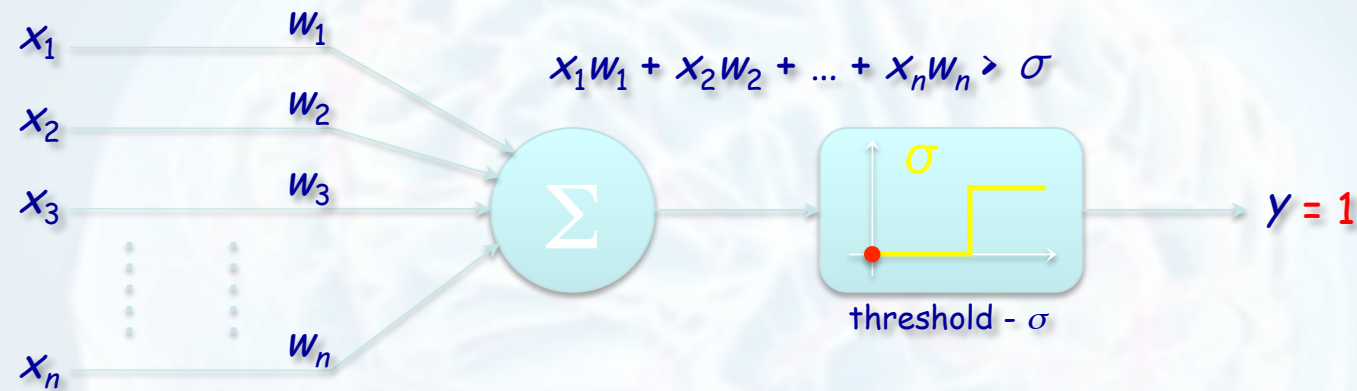
O Cérebro Humano



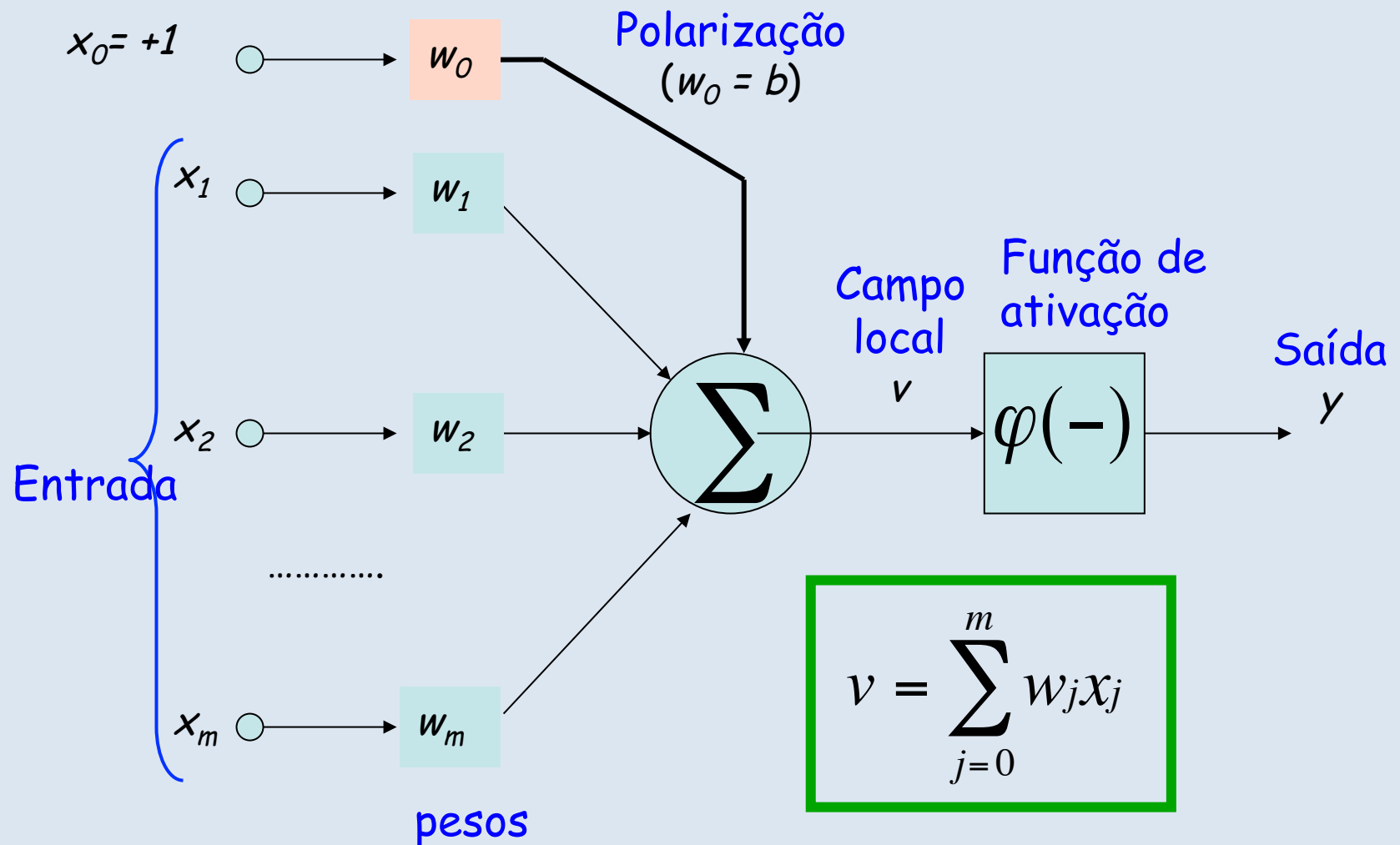
Neurônios



The McCulloch and Pitts model



Neurônio (McCulloch & Pitts, 1943)

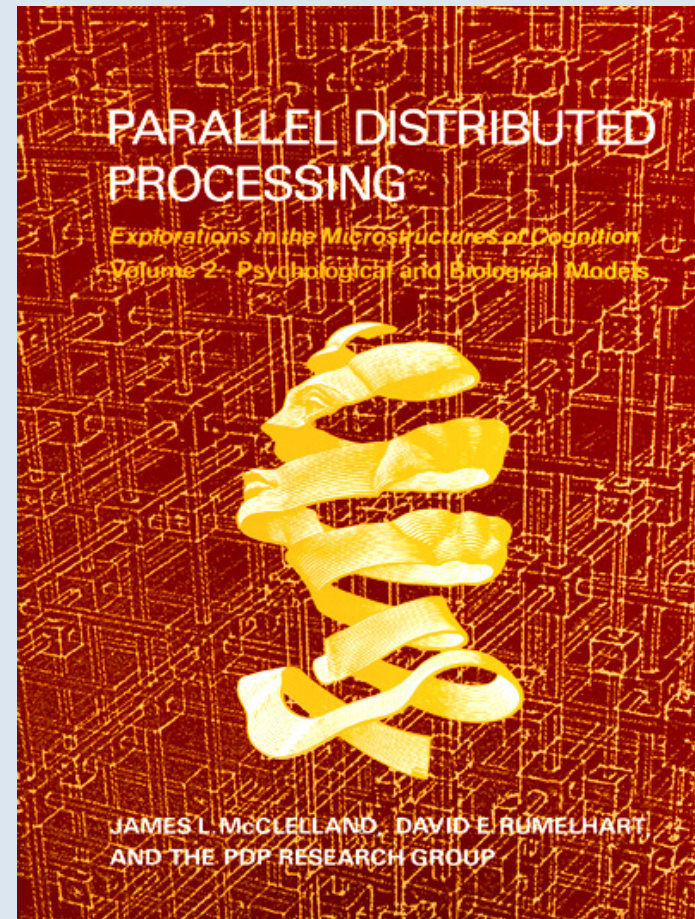
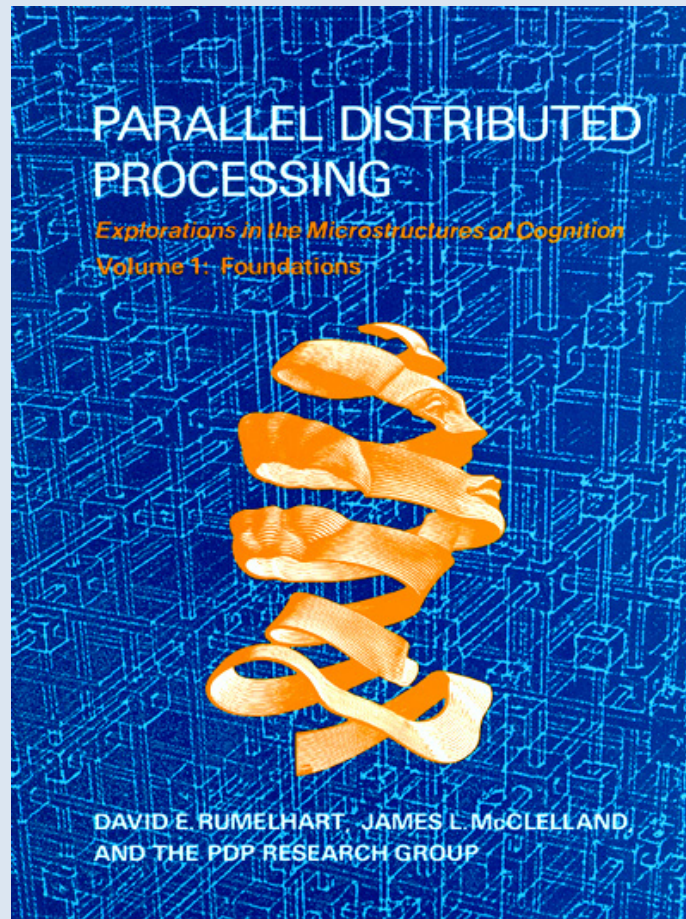


Aprendizagem humana

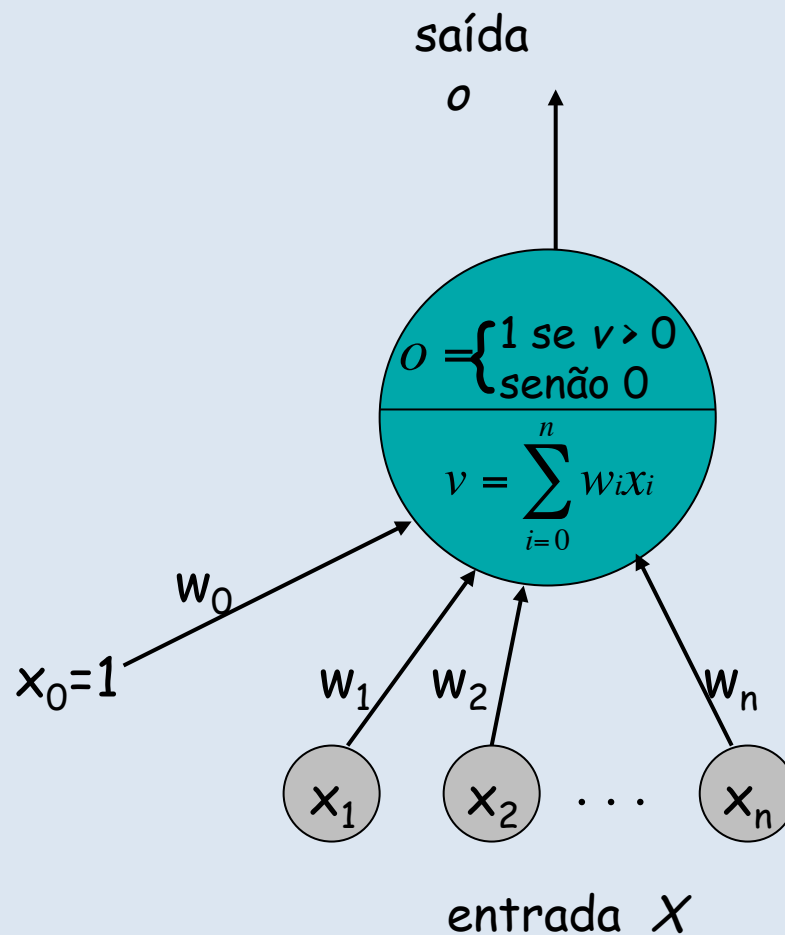
- Número de neurônios: $\sim 10^{10}$
- Conexões por neurônio: $\sim 10^4$ to 10^5
- Tempo de chaveamento: ~ 0.001 segundo
- Tempo de reconhecimento: ~ 0.1 segundo

100 passos de inferência não parece muito

A "Bíblia" (1986)

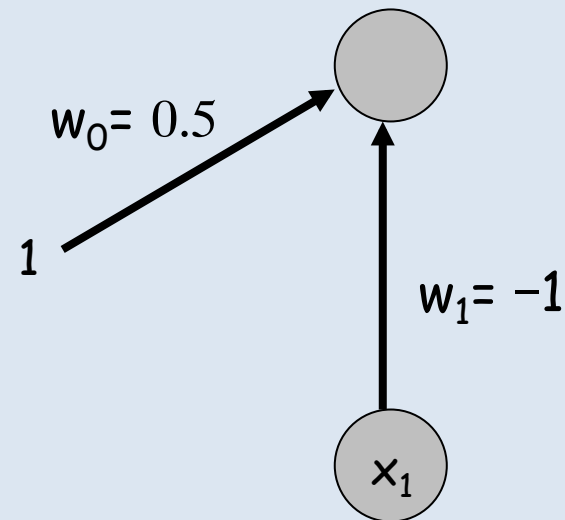


Perceptron (Frank Rosenblatt, 1957)



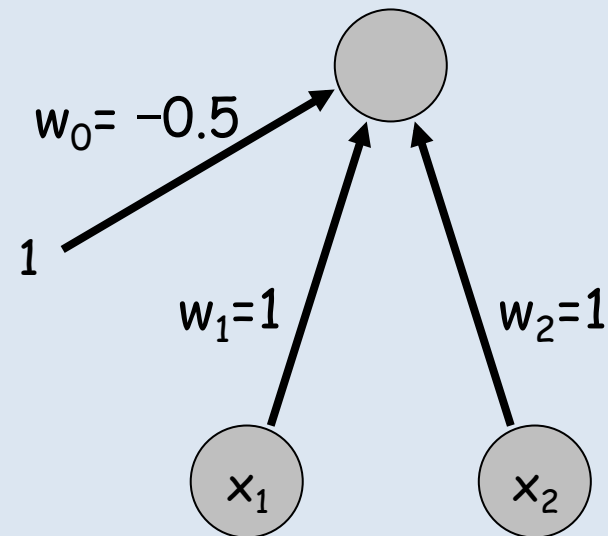
Inversor

| input x_1 | output |
|----------------|--------|
| 0 | 1 |
| 1 | 0 |



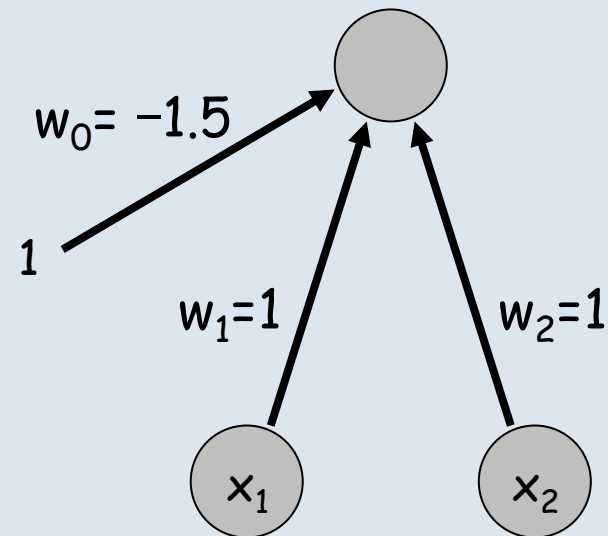
"OU" Booleano

| input x1 | input x2 | ouput |
|-------------|-------------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



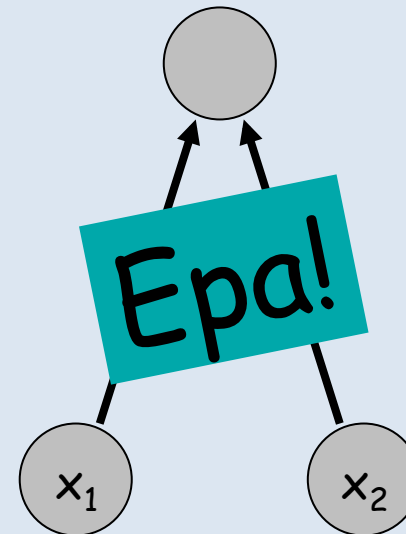
"E" Booleano

| input x1 | input x2 | ouput |
|-------------|-------------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

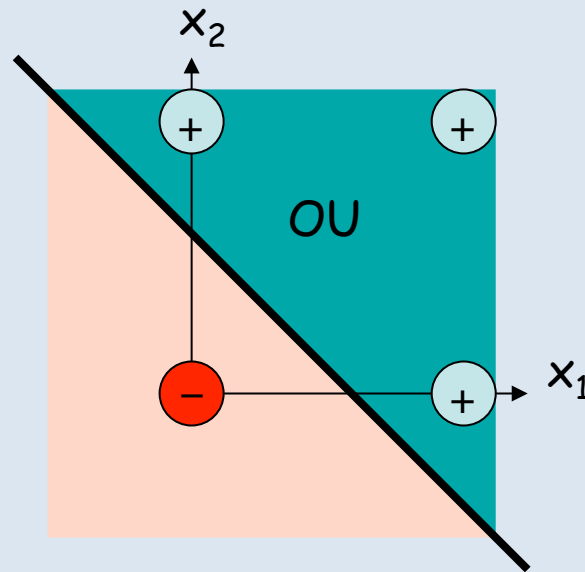


"OU-exclusivo" Booleano

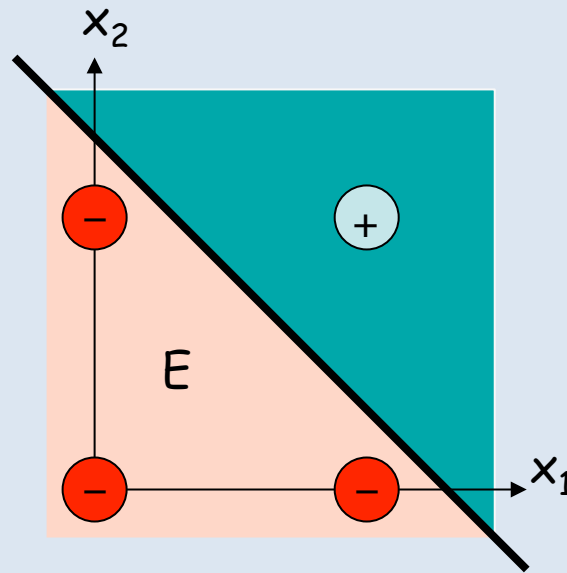
| input x1 | input x2 | ouput |
|-------------|-------------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



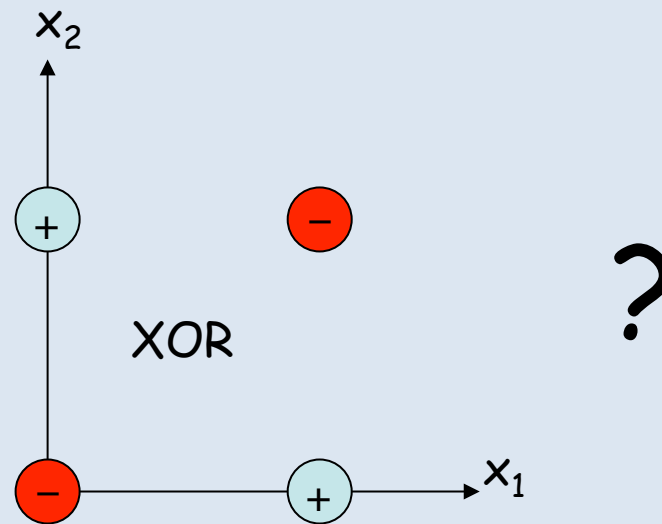
Separabilidade Linear



Separabilidade Linear

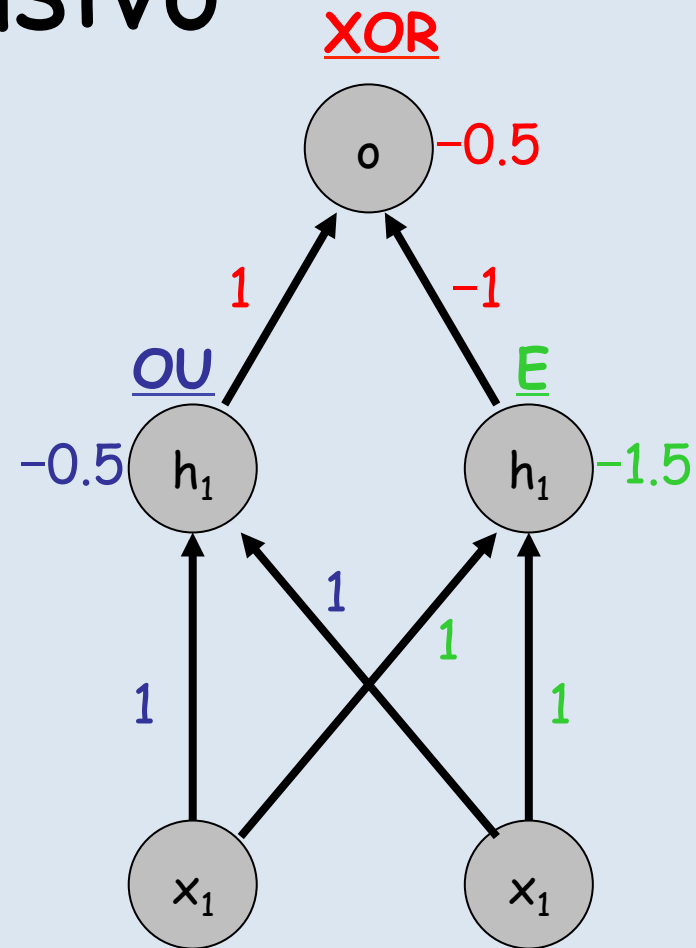


Separabilidade Linear

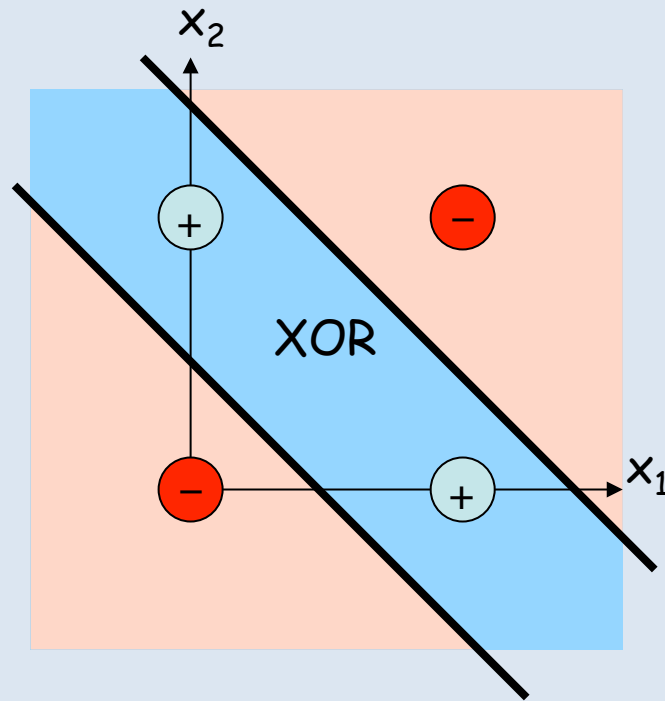


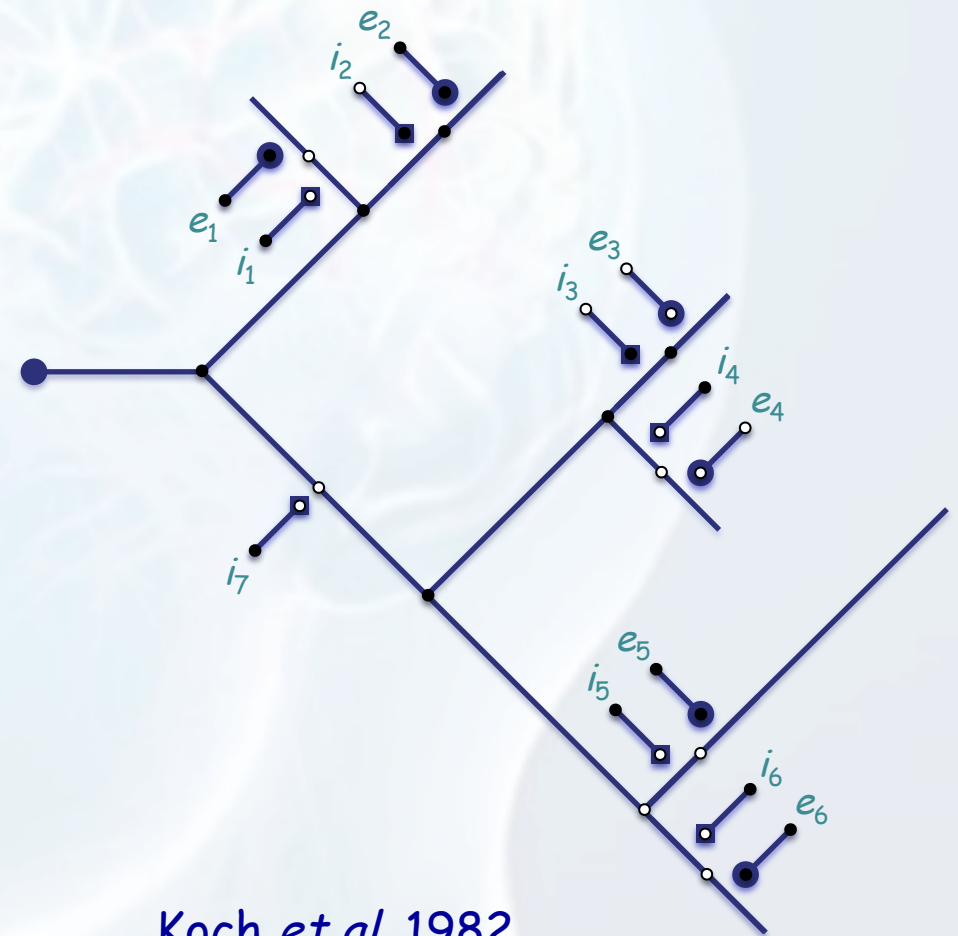
OU-exclusive

| input x1 | input x2 | ouput |
|-------------|-------------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



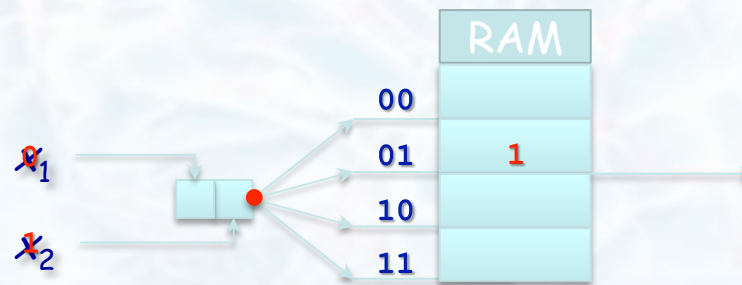
Separabilidade Linear



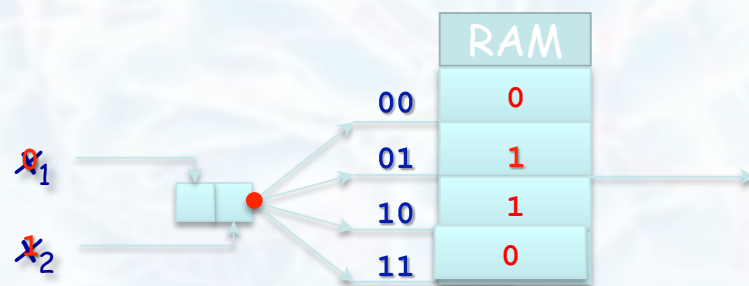


Koch *et al.* 1982

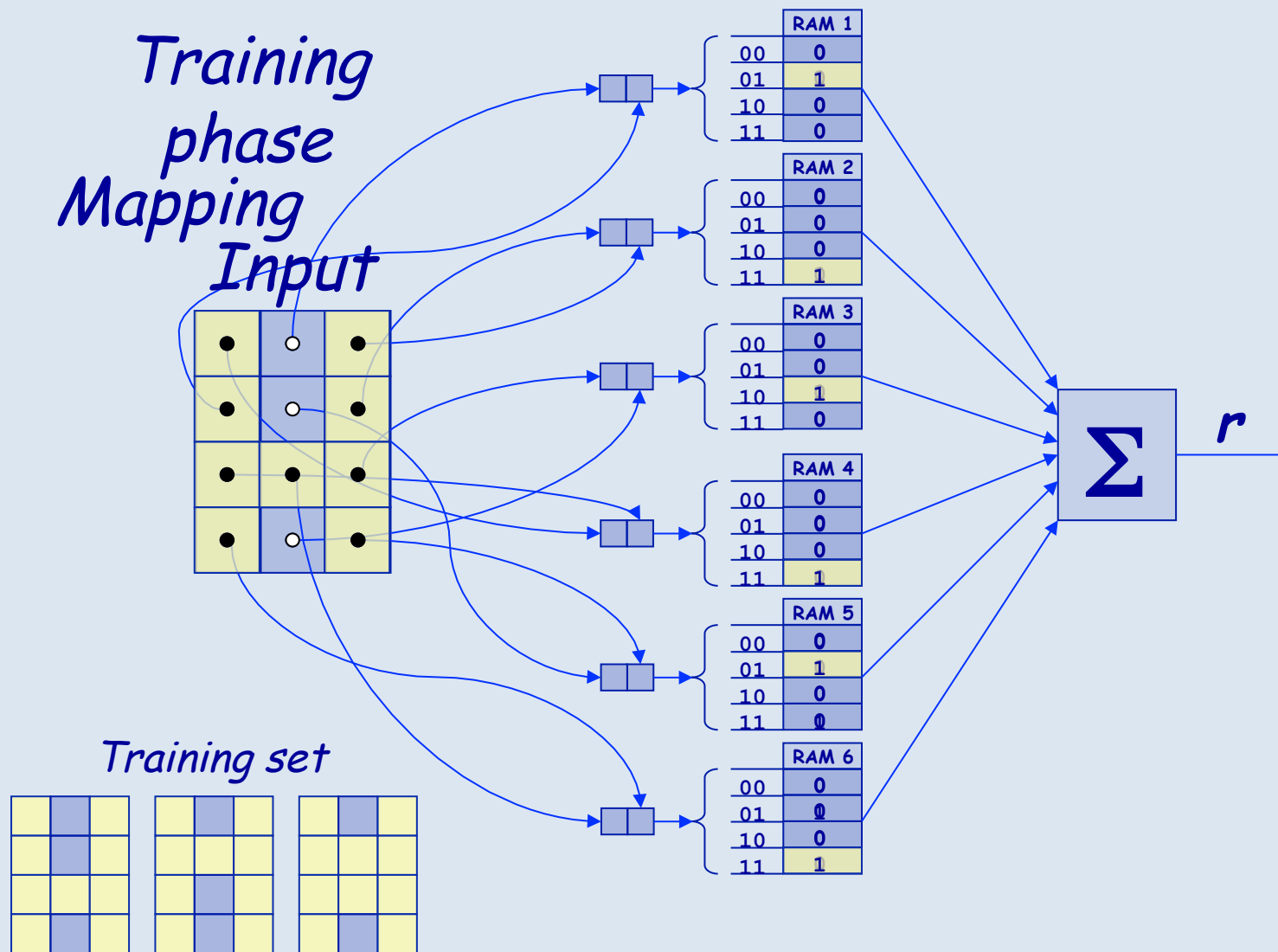
The RAM-node



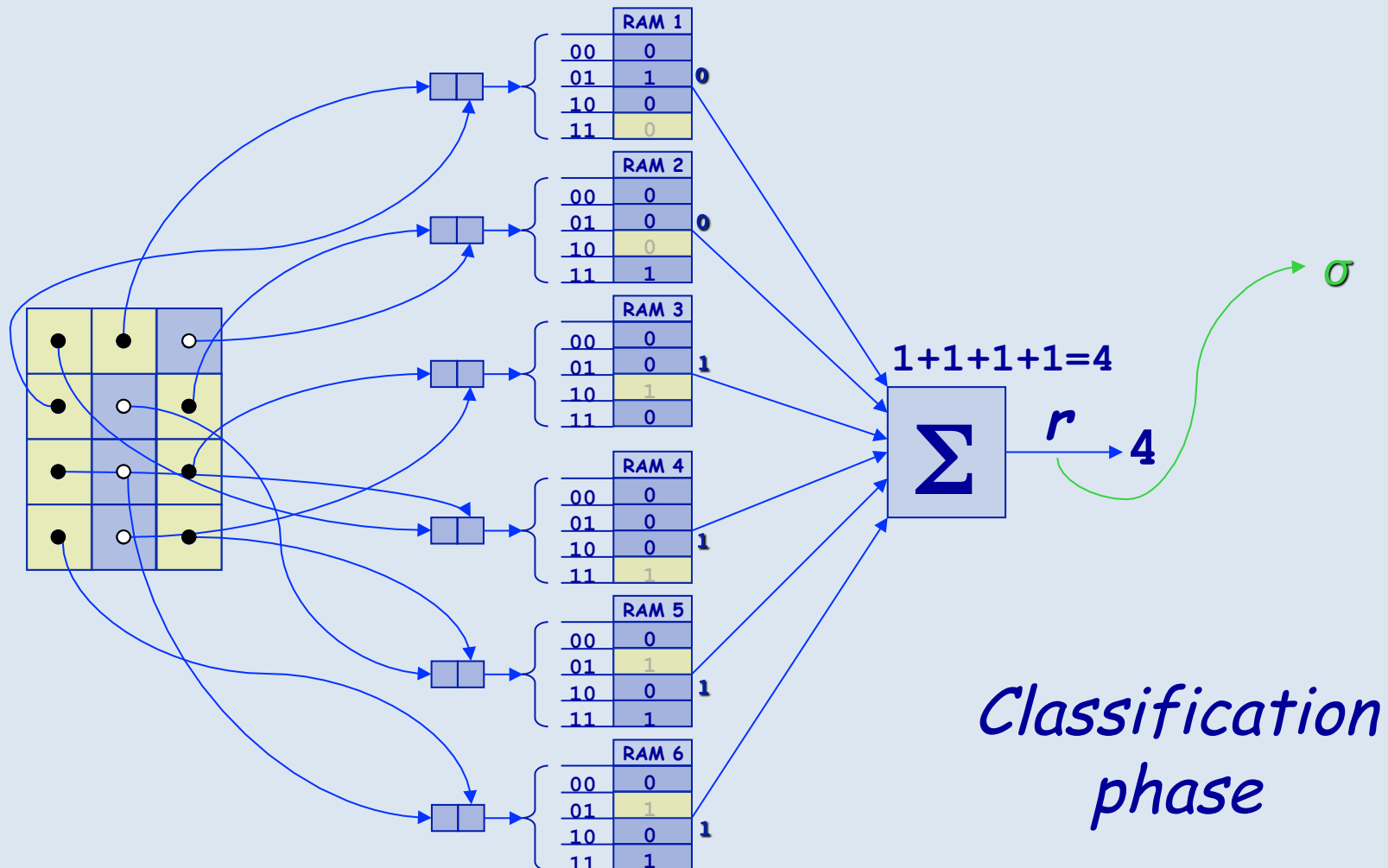
The RAM-node



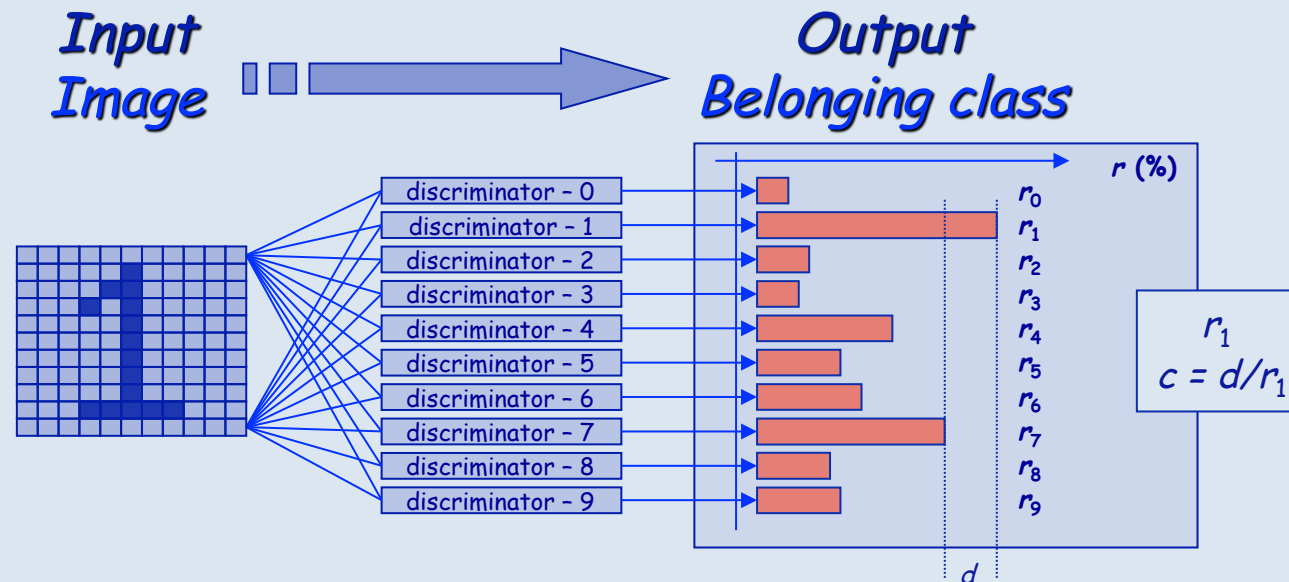
RAM discriminator



RAM discriminator



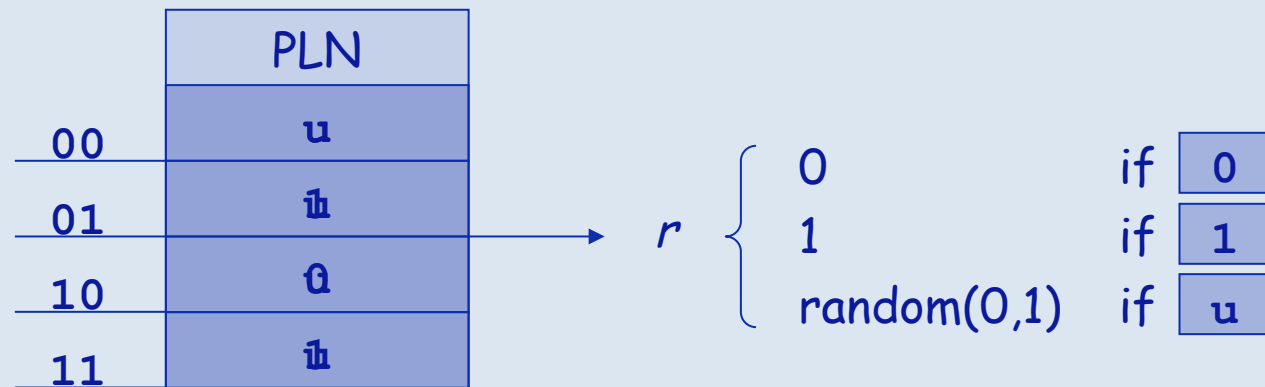
Multidiscriminator



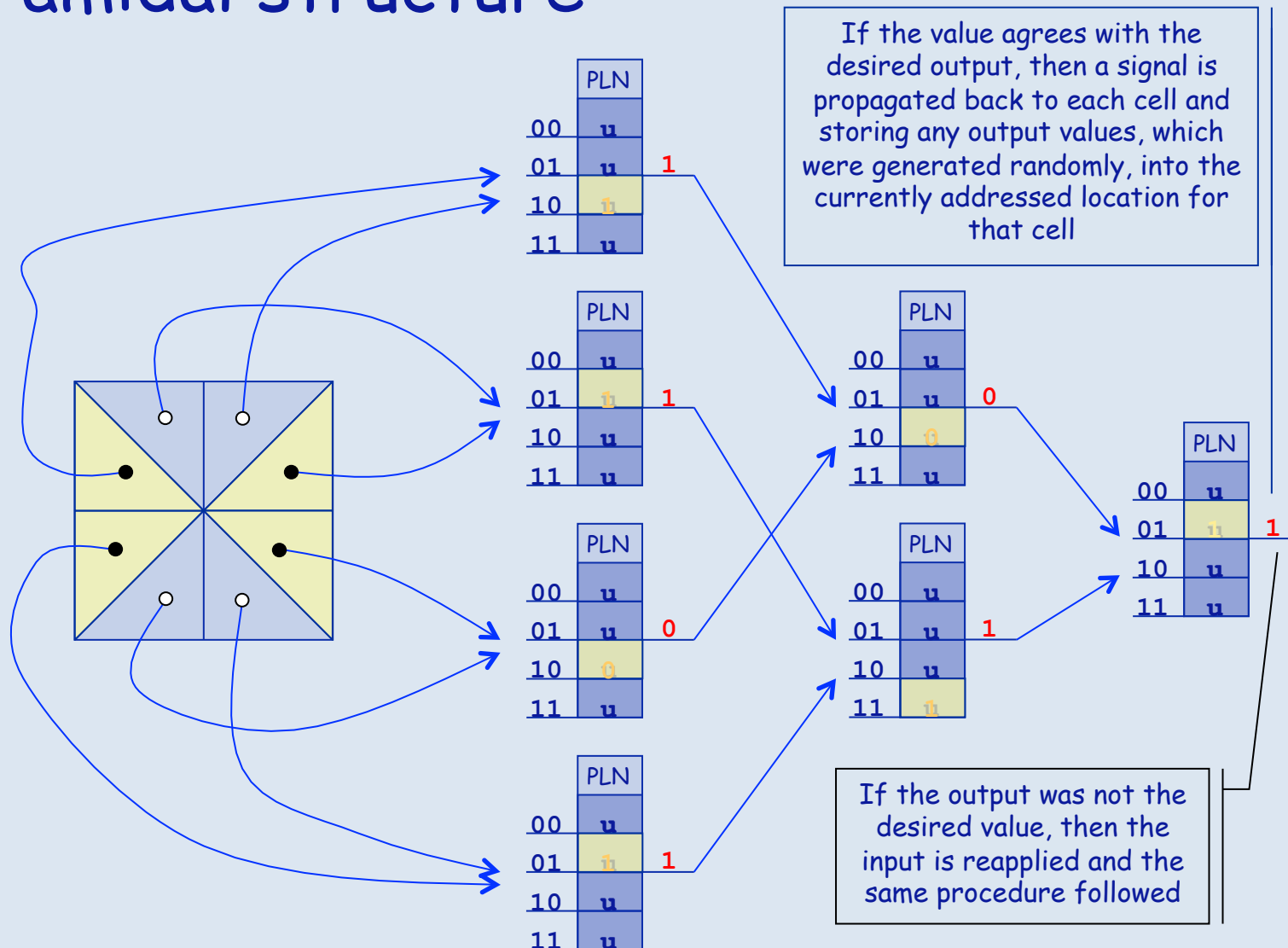
Wi.S.A.R.D.
*Wilkie Stonham and
Aleksander's Recognition Device*



PLN - Probabilistic Logic Neuron



Pyramidal structure



MPLN and pRAM

| | MPLN |
|----|--------------------------|
| 00 | $[p_1, p_2, \dots, p_n]$ |
| 01 | $[p_1, p_2, \dots, p_n]$ |
| 10 | $[p_1, p_2, \dots, p_n]$ |
| 11 | $[p_1, p_2, \dots, p_n]$ |

→ r

discrete

| | pRAM |
|----|----------|
| 00 | $[0, 1]$ |
| 01 | $[0, 1]$ |
| 10 | $[0, 1]$ |
| 11 | $[0, 1]$ |

→ r

continuous



GSN - Goal Seeking Neuron

- ✓ GSN was designed to maximise the storage efficiency of values in its memory, by storing new data without disrupting previously learned information
- ✓ Three modes of operation:
 - ***Validating*** (can the pattern be learned?)
 - ***Learning***
 - ***Recall***
- ✓ The concept of an ***Addressable Set*** is introduced
- ✓ These networks require only ***one pass*** through the data to train



GSN - Goal Seeking Neuron

Validation

The purpose of the validation phase is to check if a new pattern can be learnt without corrupting the previously stored knowledge:

value **u** on the output: the net can learn any desired output

value **0** or **1** on the output: the net can learn only the proposed pattern if the output of the net is the desired one

otherwise: there would be a disruption of information previously stored

Learning (if the validation phase is successful)

Output **0** (**1**) and at least one **0** (**1**) in the addressed content:

one of these location is selected randomly and its *input address* becomes, effectively, the desired outputs for cells in the previous layer

Otherwise:

an undefined location is selected at random and the desired output stored in that location

Recall

The aim of the recall phase is that of producing the most occurring value (**0** or **1**) in the addressed contents. If there are equal numbers of **0**'s and **1**'s in the addressed contents, then the output is u

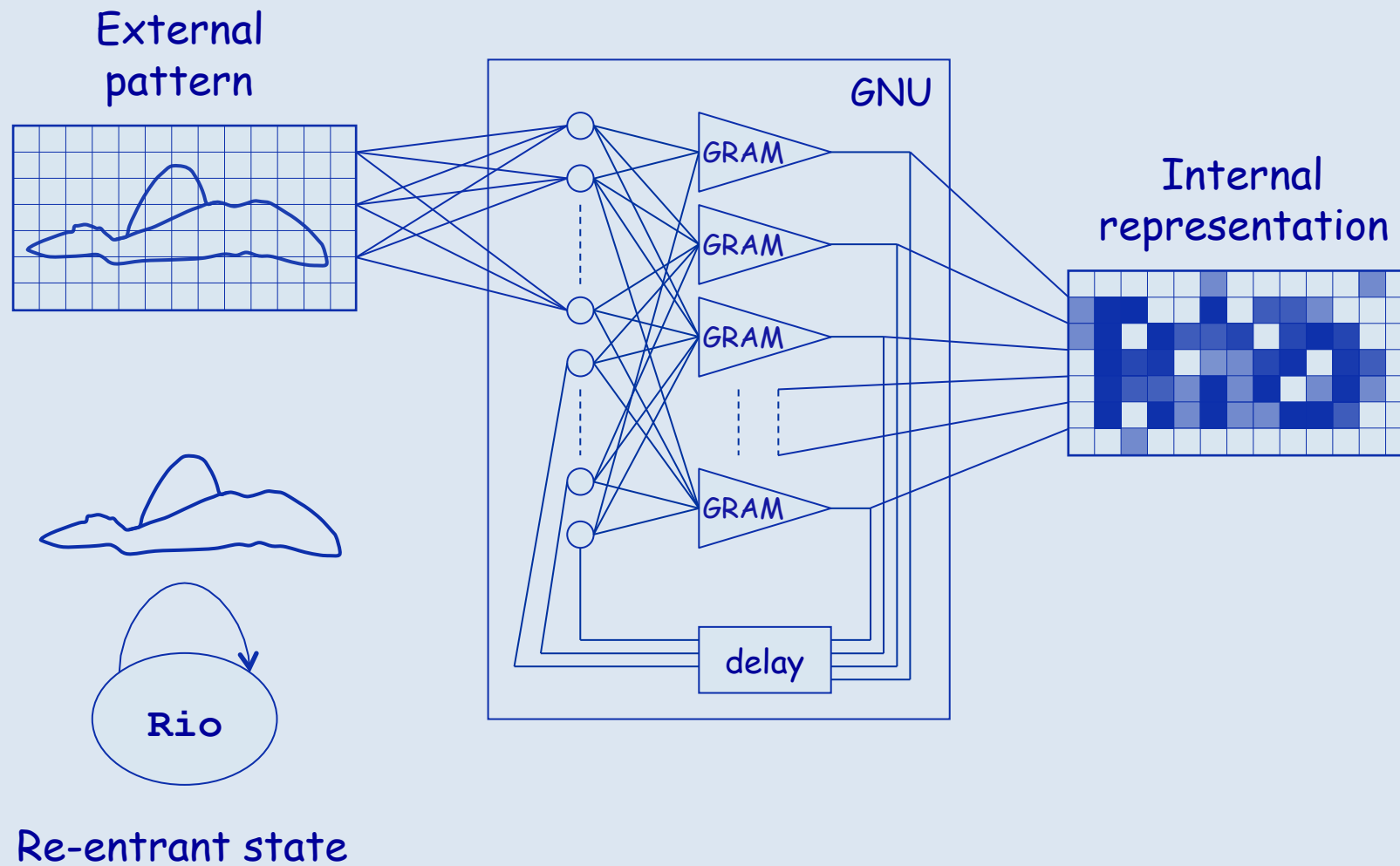


GRAM - Generalising RAM

- ✓ PLN suffers from hesitation when the training set is small (this is because too many entries are left in the u -state)
- ✓ PLNs only generalise at the network level: individual nodes do not have generalisation properties
- ✓ GRAMs were introduced by Aleksander in order to increase the generalization of WNNs at the node level by including a *spreading phase* in the learning algorithm, just after a training pattern is stored
- ✓ The *spreading algorithm* generalises information from the presented training set, since it attributes the class of the nearest stored pattern to those locations for which no information was given
- ✓ Since individual nodes with such learning scheme do generalise they are known as *Generalising Random Access Memories (GRAM)*



GNU - General Neural Unit



Some commercial and software developments

- ✓ The **WISARD** system was commercialized in 1986 and marketed by Computer Recognition systems on the UK. Systems were purchased by the UK police for face recognition and fingerprint processing
- ✓ **MAGNUS**: a system that allowed arbitrary structures of interconnected GNUs in order to model brain structures
- ✓ Barry Dunmall of Imperial College also commercialized this system, which became known as the **Neural Representation Modeller (NRM)**

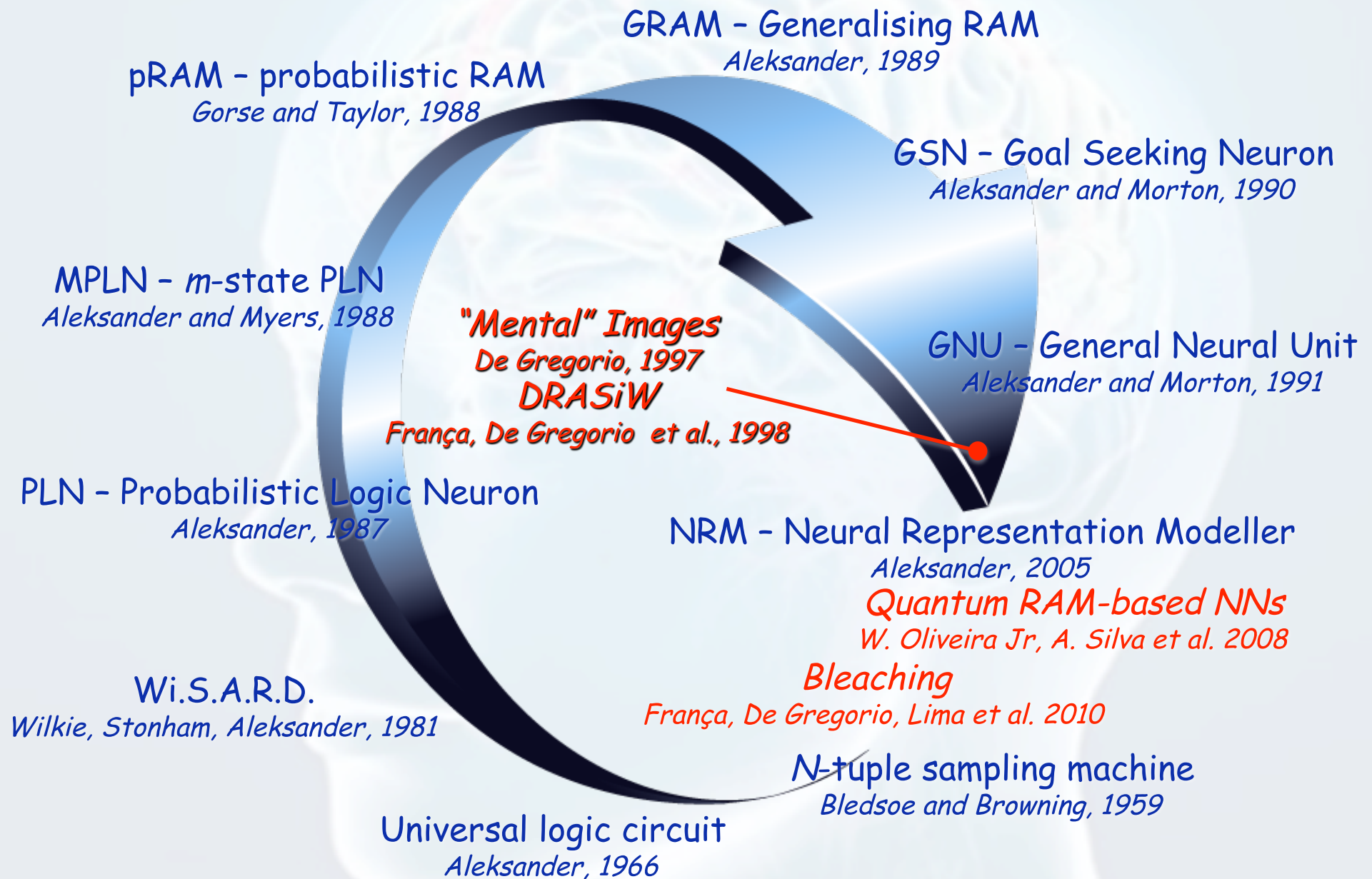


Conclusions

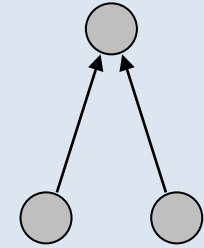
- ✓ Although WNNs are not in the mainstream of ANN research, the agility and scalability of these models implemented in the existing computing environments is very attractive
- ✓ Recent publications report that the role of WNNs were crucial in scientific investigations of very interesting subjects, such as artificial consciousness as well as in recent development of commercial solutions to important and difficult problems, such as automated video surveillance, robotics, acceleration of 3D video animation and automated text categorization and clustering



Chronological excursus



Perceptron: treinamento



novo peso

peso

incremento

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

incremento

passo

alvo

saída do perceptron

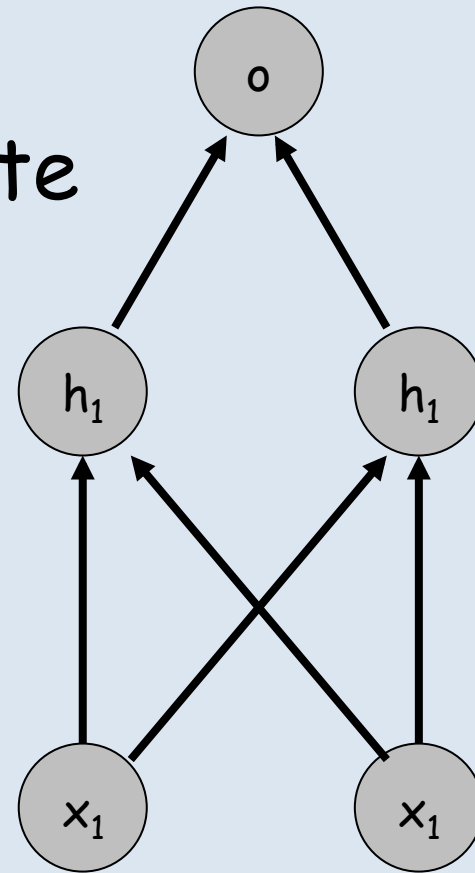
entrada

Converge, se...

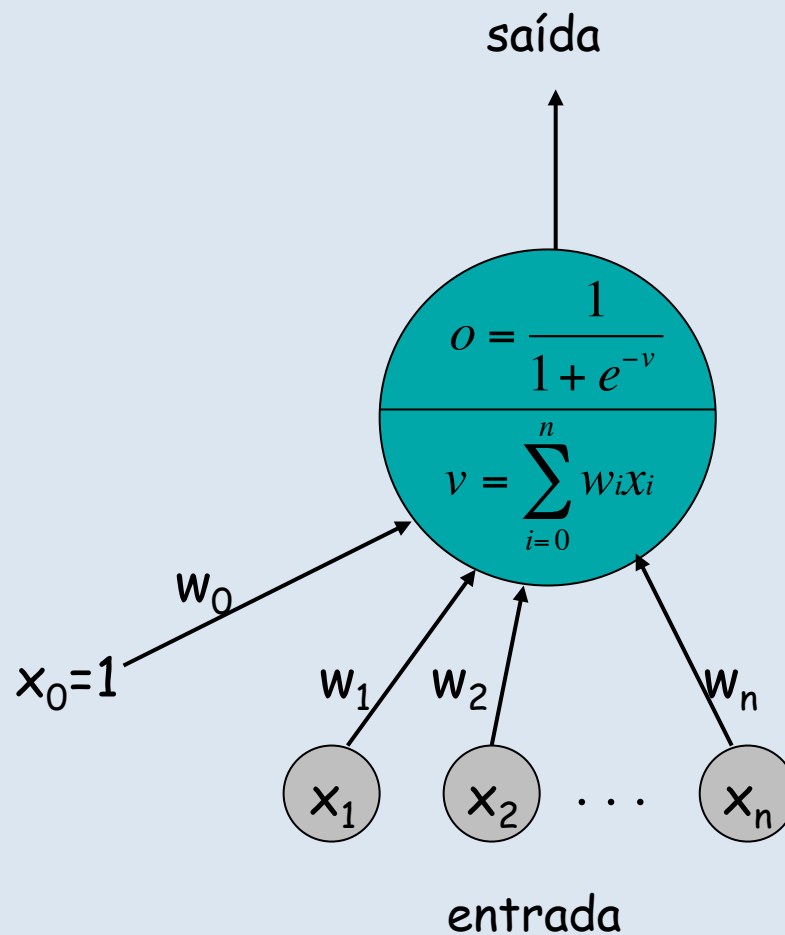
- ... dados de treinamento são linearmente separáveis
- ... passo η suficientemente pequeno
- ... sem neurônios "escondidos"

Como treinar *Perceptrons* Multi-Camadas?

- Gradiente descendente



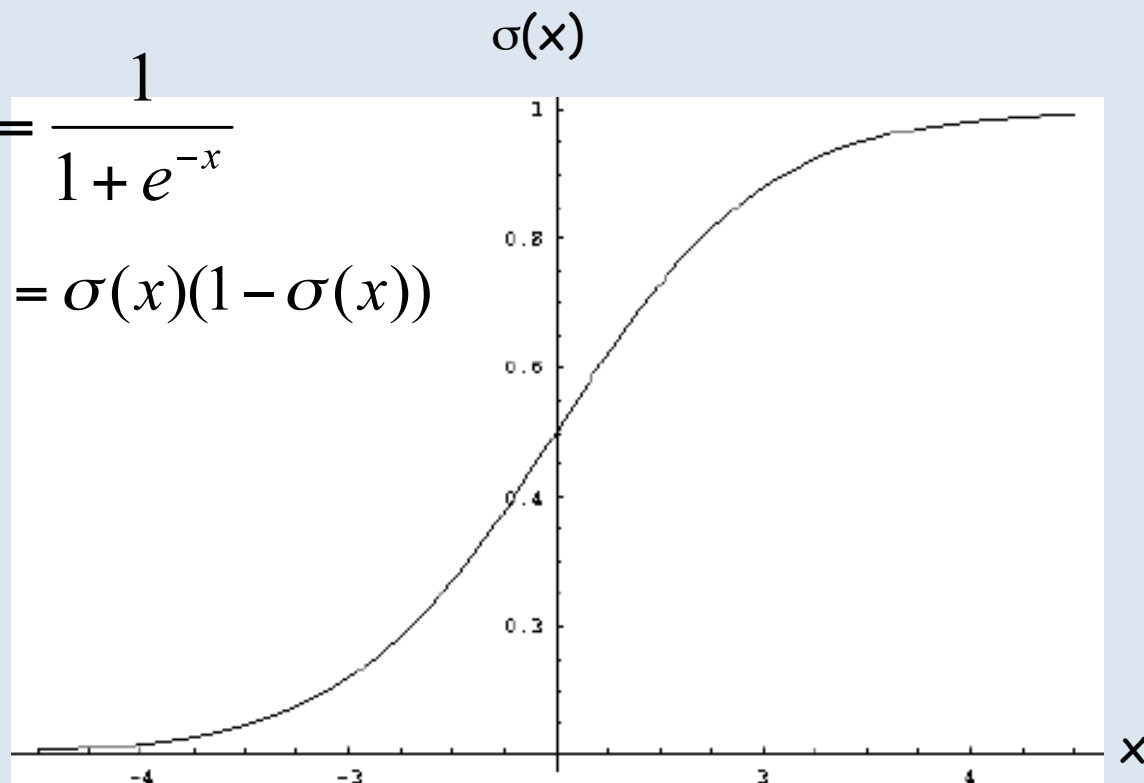
Função Logística (sigmóide)



Função Logística (sigmóide)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



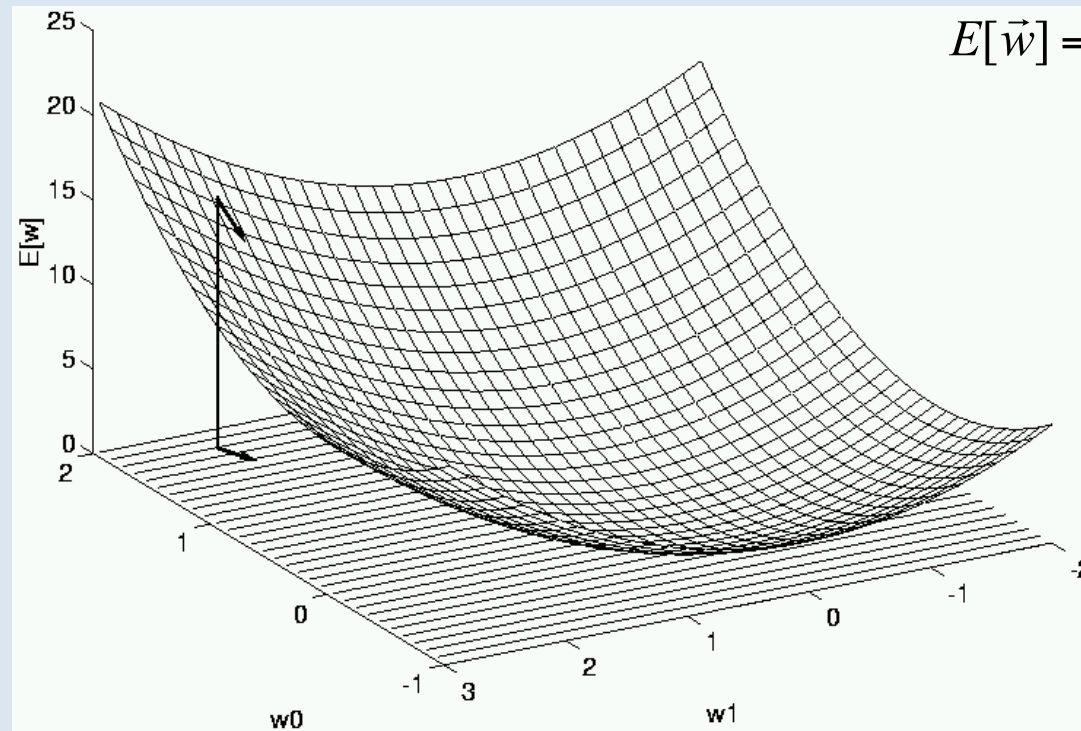
Gradiente Descendente

- Aprender pesos w_i que minimizem o erro quadrático

$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

D = dados de treinamento

Gradiente Descendente



$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Gradiente:

$$\nabla E[\vec{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Regra de treinamento:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradiente Descendente (única camada)

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \sigma(\vec{w} \cdot \vec{x}_d)) \\&= \sum_d (t_d - o_d) \left(-\frac{\partial}{\partial w_i} \sigma(\vec{w} \cdot \vec{x}_d) \right) \\&= -\sum_d (t_d - o_d) \sigma(\vec{w} \cdot \vec{x}_d) (1 - \sigma(\vec{w} \cdot \vec{x}_d)) x_{i,d}\end{aligned}$$

Aprendizagem não-incremental (*batch*)

- Inicialize cada w_i com pequeno valor aleatório
- Repetir até terminar:

$$\Delta w_i = 0$$

Para cada exemplo de treino d faça

$$o_d \leftarrow \sigma(\sum_i w_i x_{i,d})$$


$$\Delta w_i \leftarrow \Delta w_i + \eta (t_d - o_d) o_d (1 - o_d) x_{i,d}$$


$$w_i \leftarrow w_i + \Delta w_i$$

Aprendizagem incremental (online)

- Inicialize cada w_i com pequeno valor aleatório
- Repetir até terminar:

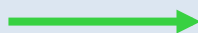
Para cada exemplo de treino d faça


$$\Delta w_i = 0$$


$$o_d \leftarrow \sum_i w_i x_{i,d}$$

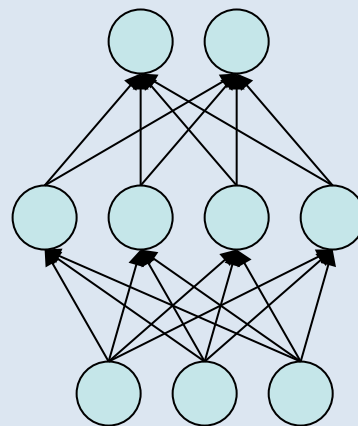
$$\Delta w_i \leftarrow \Delta w_i + \eta (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

$$w_i \leftarrow w_i + \Delta w_i$$



Algoritmo de Retropropagação (*Backpropagation*)

- Generalização para múltiplas camadas e múltiplos neurônios de saída



Algoritmo de Retropropagação (*Backpropagation*)

- Inic. pesos com pequenos valores aleatórios
- Para cada exemplo de treinamento faça
 - Para cada nó escondido h : $o_h = \sigma(\sum_i w_{hi} x_i)$
 - Para cada nó de saída k : $o_k = \sigma(\sum_h w_{kh} x_h)$
 - Para cada nó de saída k : $\delta_k = o_k (1 - o_k) (t_k - o_k)$
 - Para cada nó escondido h : $\delta_h = o_h (1 - o_h) \sum_k w_{hk} \delta_k$
 - Atualizar cada peso w_{ij} :

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \quad \text{com} \quad \Delta w_{ij} = \eta \delta_j x_{ij}$$

Algoritmo de Retropropagação (*Backpropagation*)

