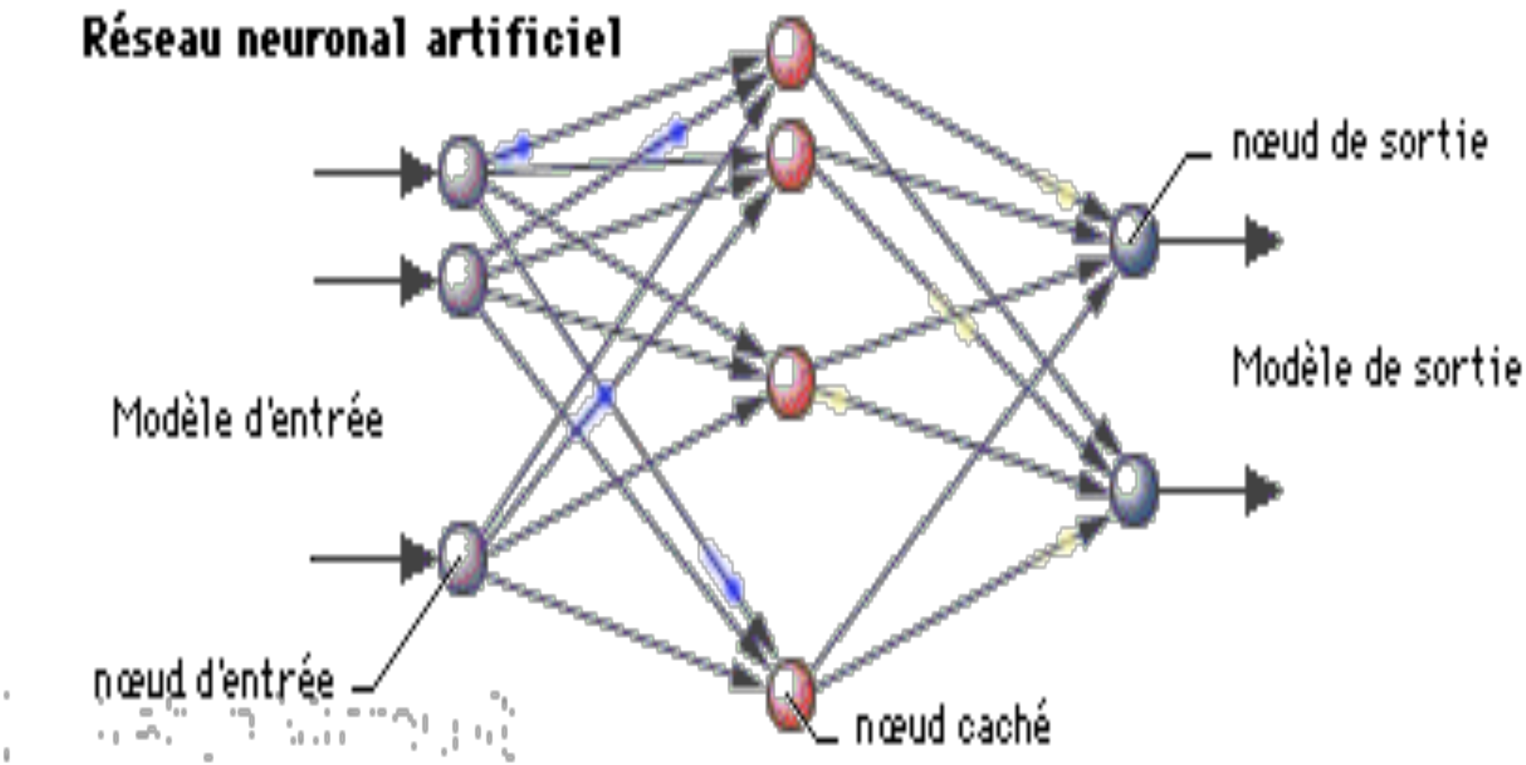
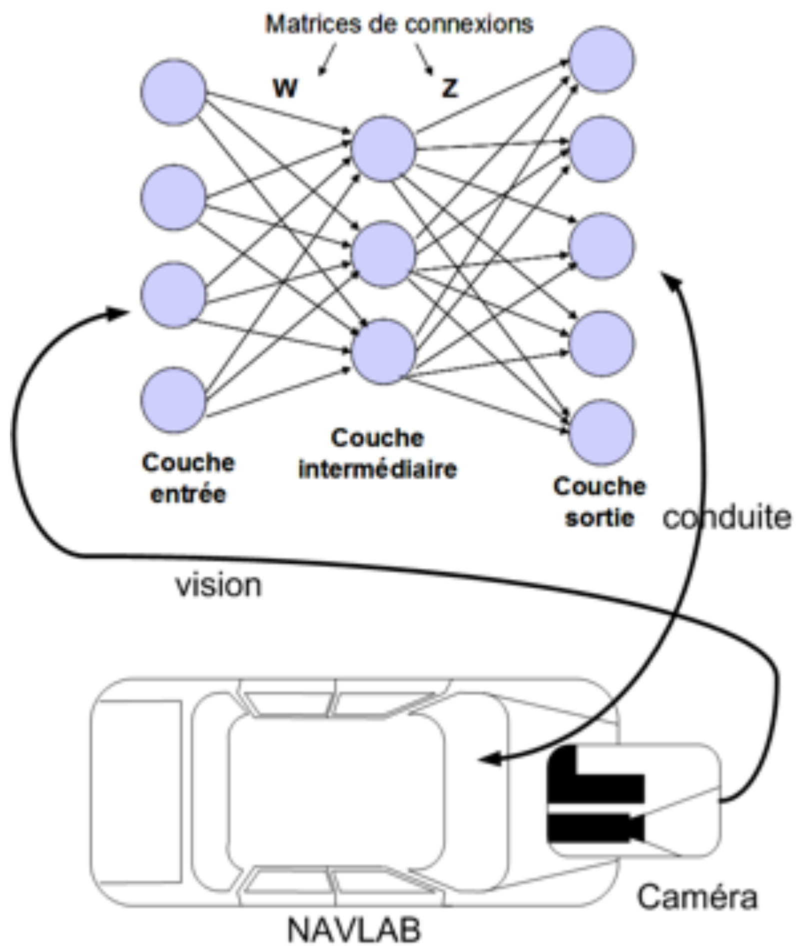
Neural Networks





Matrices de connexions
W
Z
Couche entrée
Couche intermédiaire
Couche sortie
conduite
vision
NAVLAB
Caméra

Plan

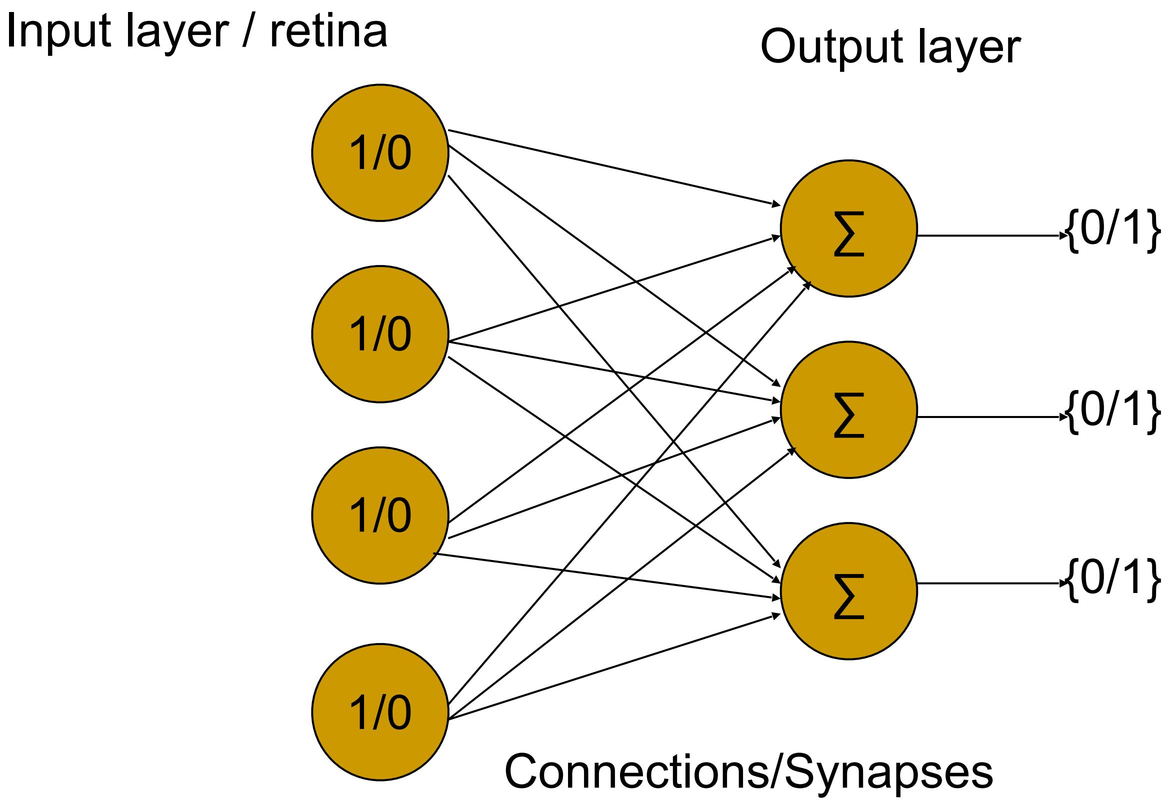
- Perceptron
  - Linear discriminant
- Associative memories
  - Hopfield networks
  - Chaotic networks
- Multilayer perceptron
  - Backpropagation

Perceptron

- Historically, the first neural net
- Inspired by human brain
- Proposed
  - By Rosenblatt
  - Between 1957 et 1961
- The brain was appearing as the best computer
- Goal: associated input patterns to recognition outputs
- Akin to a linear discriminant

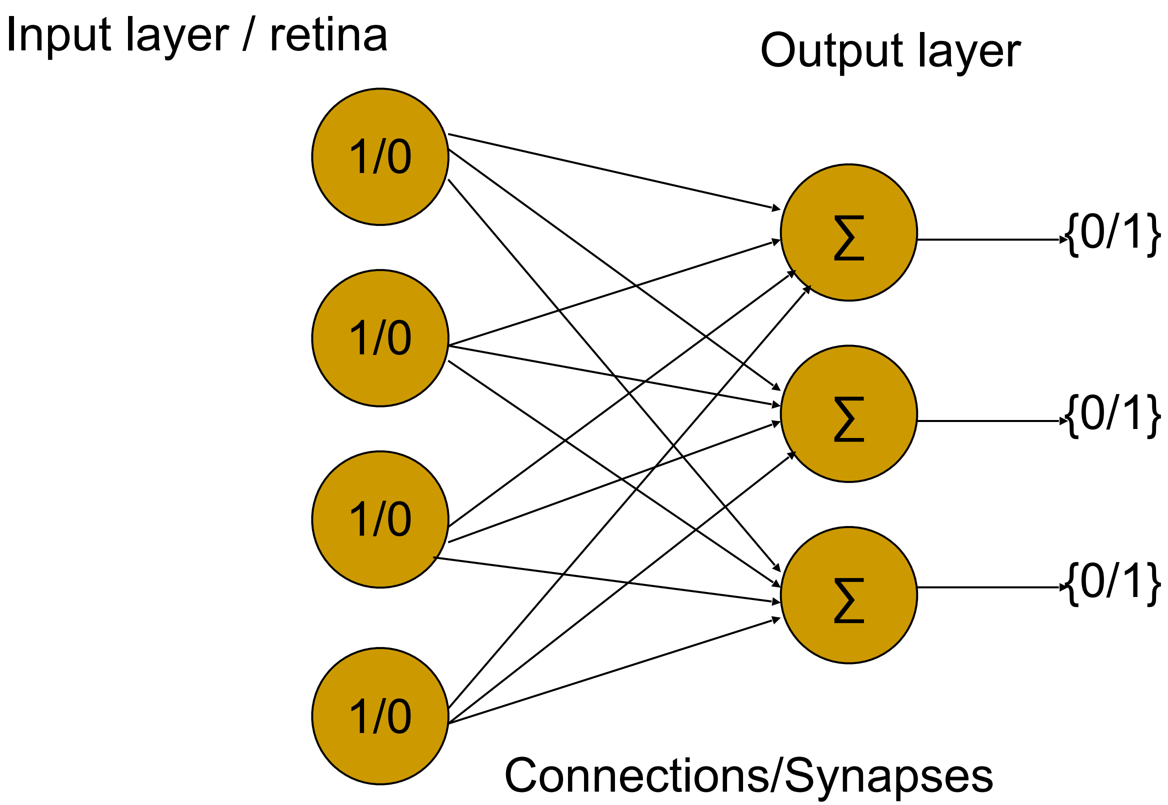
Perceptron

- Constitution



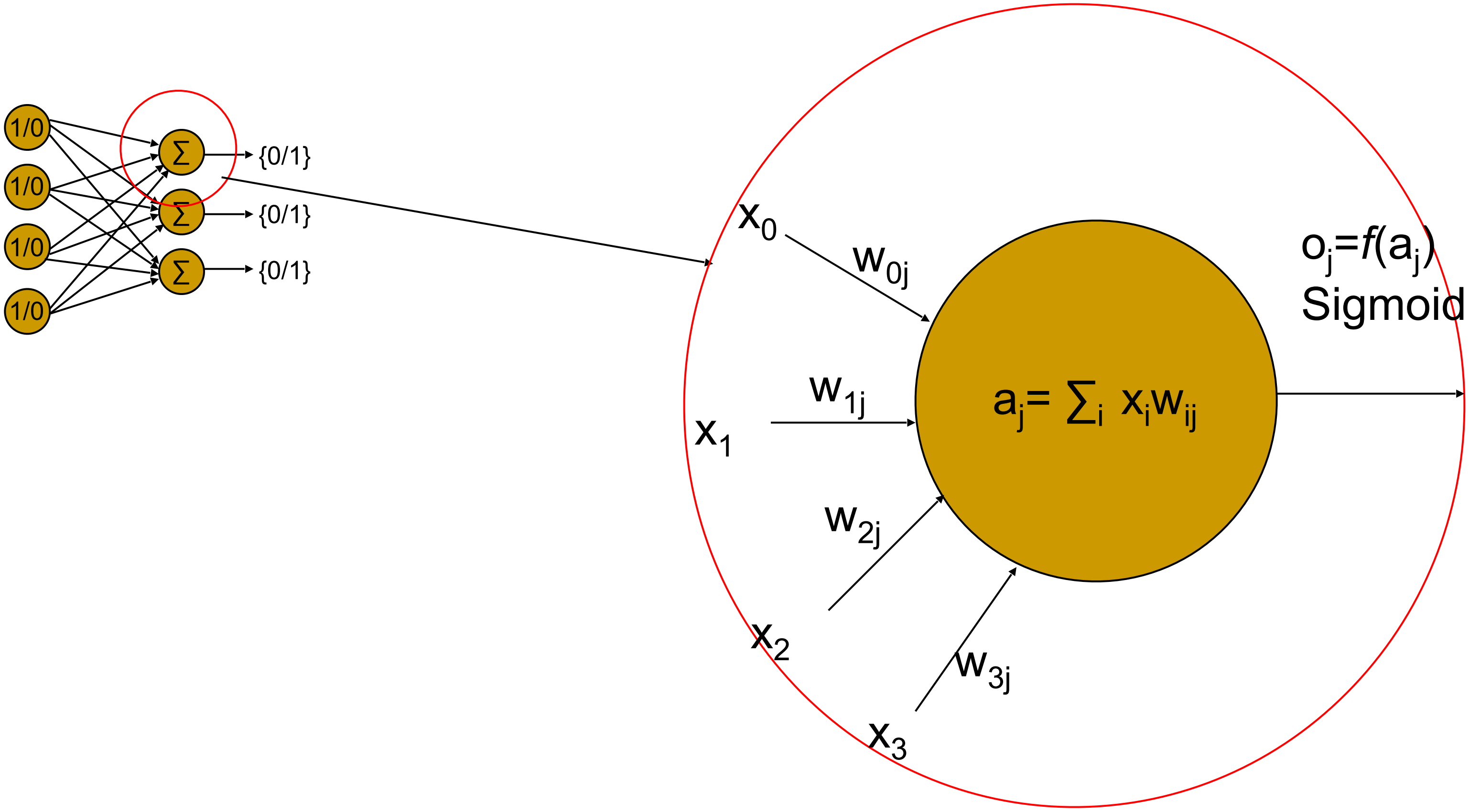
Perceptron

- Constitution



Perceptron

- Constitution



Perceptron

- Constitution

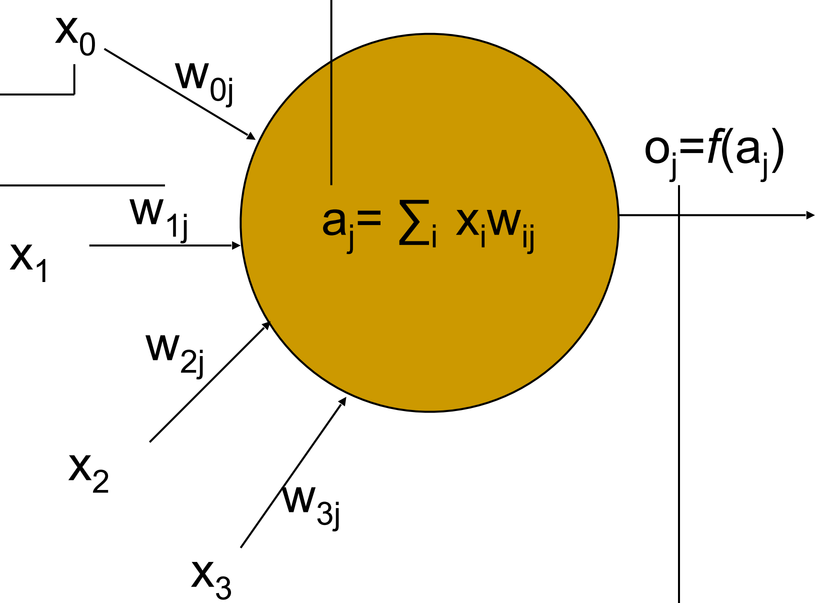
a_j : activation of the j output neuron

x_i : activation of the i input neuron

$w_{i,j}$: connection parameter between input neuron i and output neuron j

o_j : decision rule

$o_j = 0$ for $a_j <= \theta_j$, 1 for $a_j > \theta_j$



Perceptron

- Need an associated learning
  - Learning is supervised
    - Based on a couple input pattern and desired output
  - If the activation of output neuron is OK => nothing happens
  - Otherwise – inspired by neurophysiological data
    - If it is activated : decrease the value of the connection
    - If it is unactivated : increase the value of the connection
  - Iterated until the output neurons reach the desired value

Perceptron

- Supervised learning
  - How to decrease or increase the connections ?
  - Learning rule of Widrow-Hoff
  - Closed to Hebbian learning

$$\boxed{w_{i,j}^{(t+1)}} = \boxed{w_{i,j}^{(t)}} + \boxed{n(t_j - o_j)x_i} = w_{i,j}^{(t)} + \Delta w_{i,j}$$

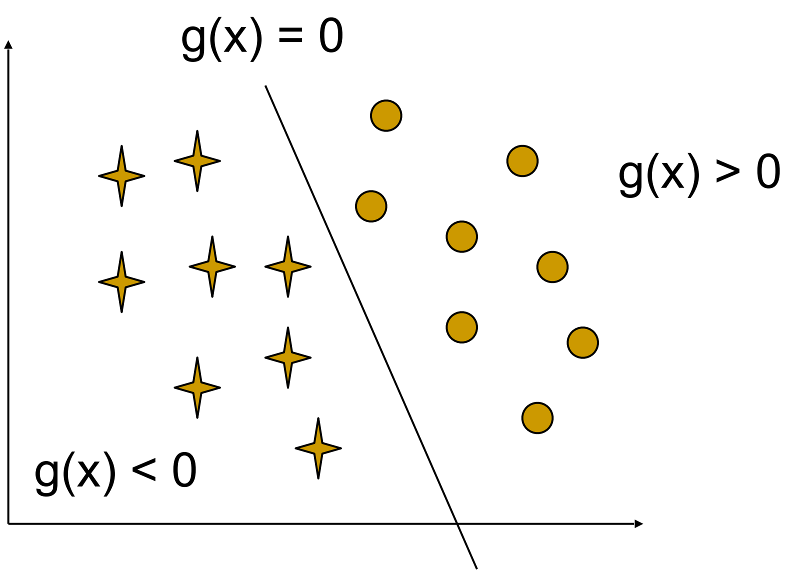
Desired value of output neuron j

Learning rate

Theory of linear discriminant

Compute:

$g(x) = W^T x + W_o$



And:

Choose:

class 1 if g(x) > 0

class 2 otherwise

But how to find W on the basis of the data ?

Gradient descent:

$$\Delta W_i = -\eta \frac{\partial E}{\partial W_i}, \forall i$$

In general a sigmoid is used for the statistical interpretation: (0,1)

$$Y = 1/1 + \exp\left[- g(x)\right]$$

Easy to derive = Y(1-Y)

Class 1 if Y > 0.5 and 2 otherwise

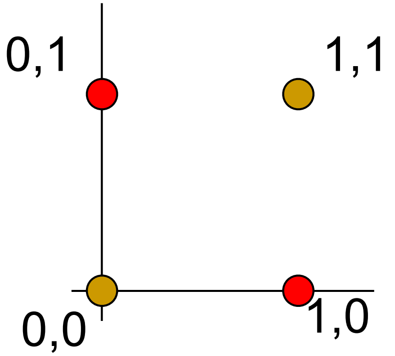
The error could be least square: $(Y - Y_d)^2$

Or maximum likelihood: $-\sum Y_d \log Y + (1 - Y_d)\log(1 - Y)$

But at the end, you got the learning rule: $\Delta W = \eta \sum (Yd - Y) X_j$

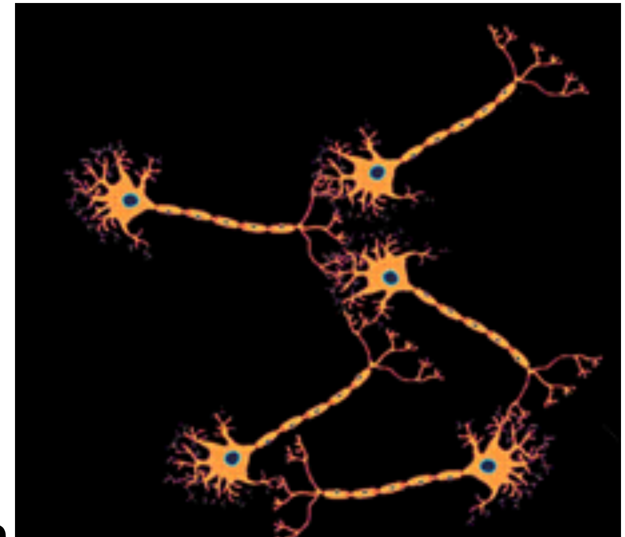
Perceptron limitations

- Limitations
  - Not always easy to learn
  - But above all, cannot separate not linearly separable data
- Why so ?
  - The XOR kills NN researches
    for 20 years
    (Minsky and Papert were responsable)



- Consequence
  - We had to wait for the magical hidden layer
  - And for backpropagation

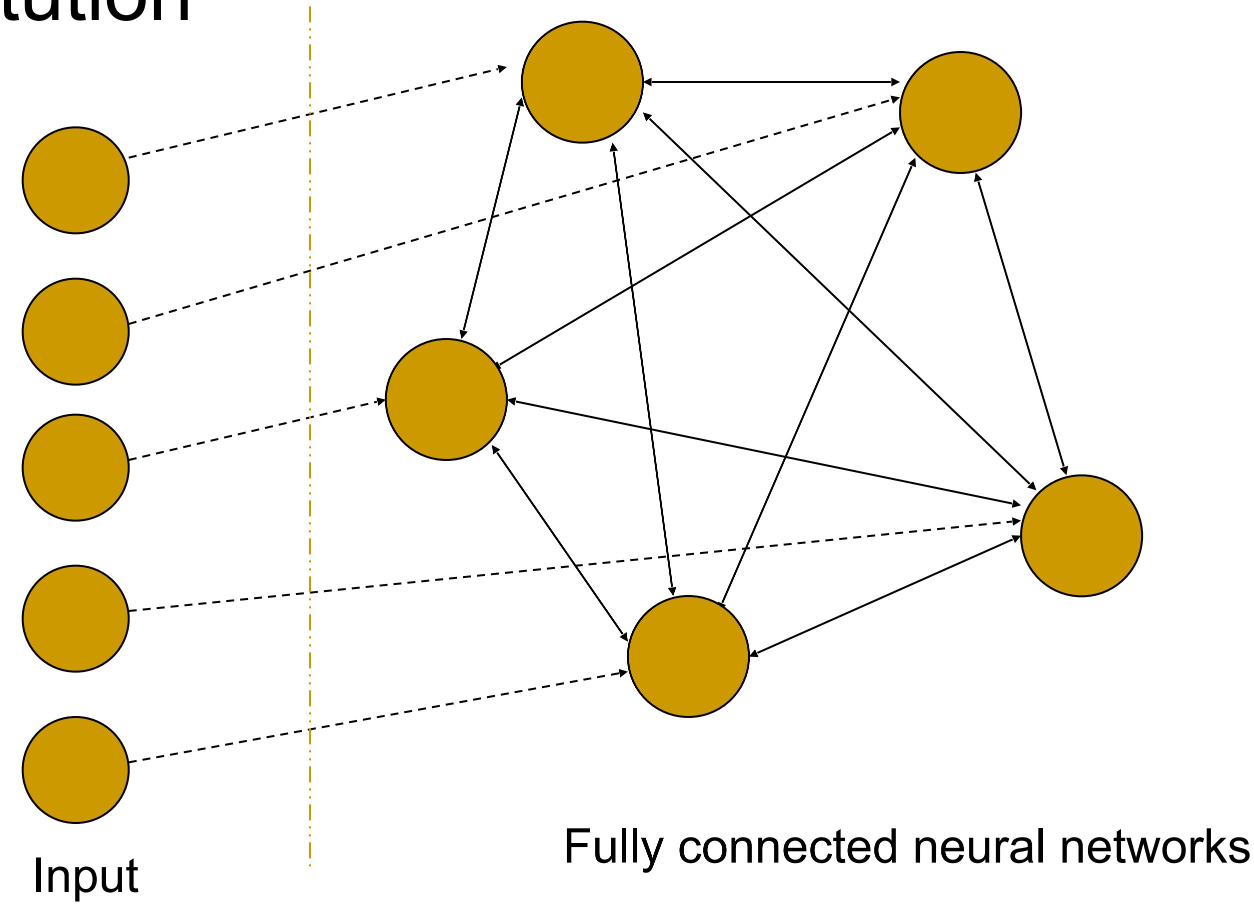
Associative memories



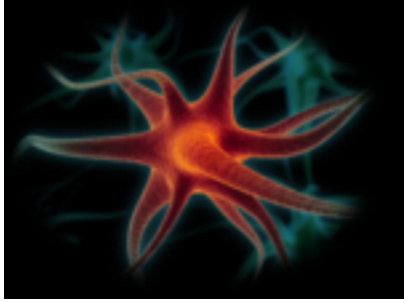
- Around 1970
- Two types
  - Hetero-associative
  - And auto-associative
- We will treat here only auto-associative
- Make an interesting connections between neurosciences and physics of complex systems
- John Hopfield

Auto-associative memories

- Constitution

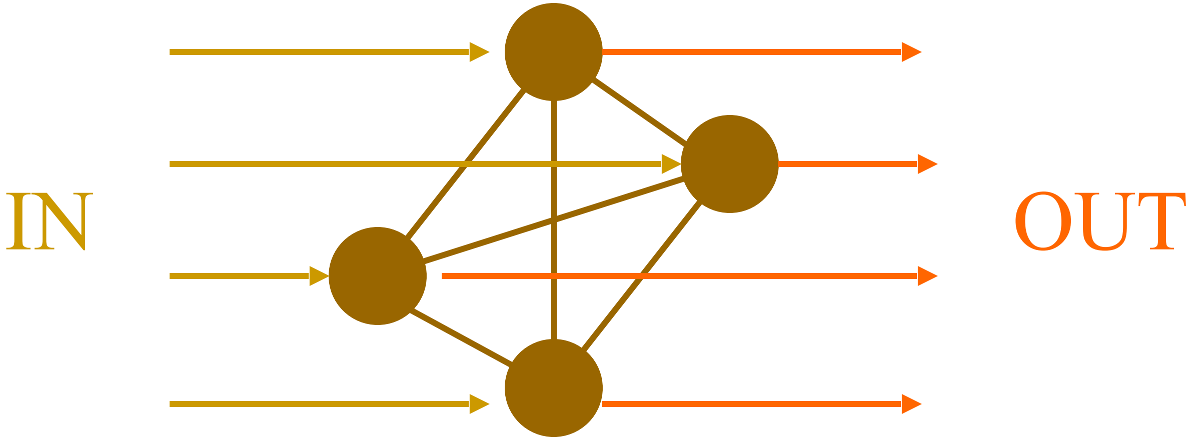


Associative memories



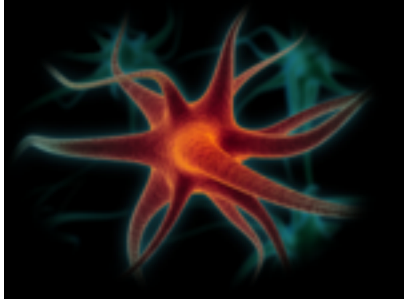
Hopfield -> DEMO

- Fully connected graphs
- Input layer = Output layer = Networks
- The connexions have to be symmetric



- It is again an hebbian learning rule

Associative memories

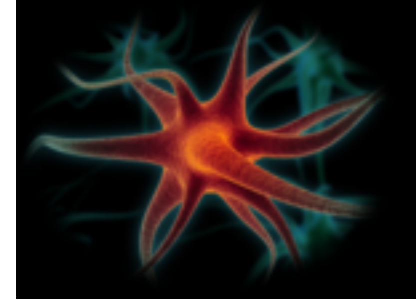


<u>Hopfield</u>

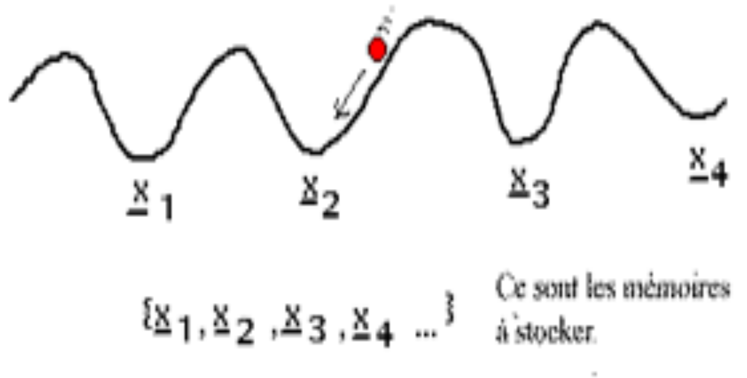
- The newtork becomes a dynamical machine
- It has been shown to converge into a fixed point
- This fixed point is a minimal of a Lyapunov energy

- These fixed point are used for storing «patterns »
- Discrete time and asynchronous updating
  - input in {-1,1}
  - $x_i \rightarrow \mathrm{sign}(\Sigma_j\, w_{ij} x_j)$

Mémoires associatives



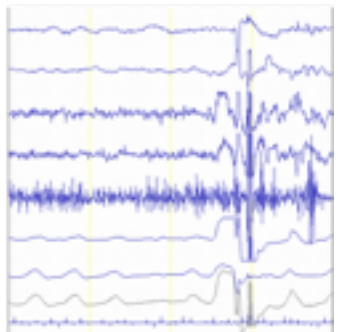
Hopfield

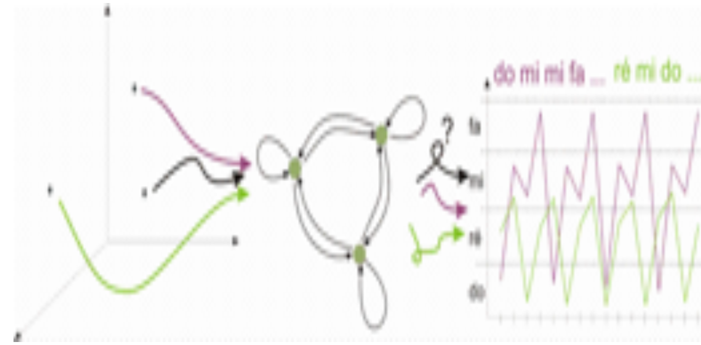


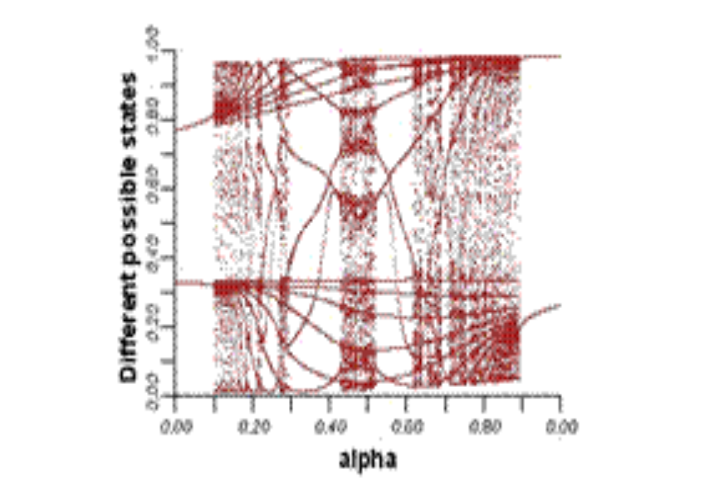
- The learning is done by Hebbian learning

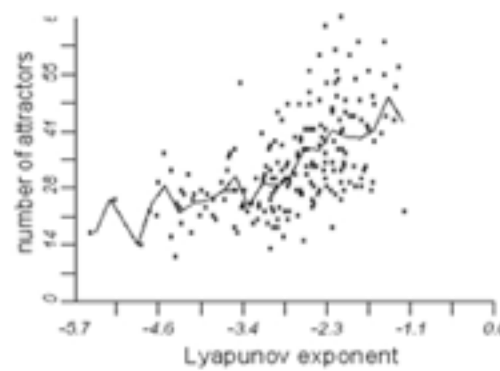
- Over all patterns to learn:
  - $$\Delta W_{ij} = \sum_{patterns} X_i^p X_j^p$$

My researches: Chaotic encoding of memories in brain



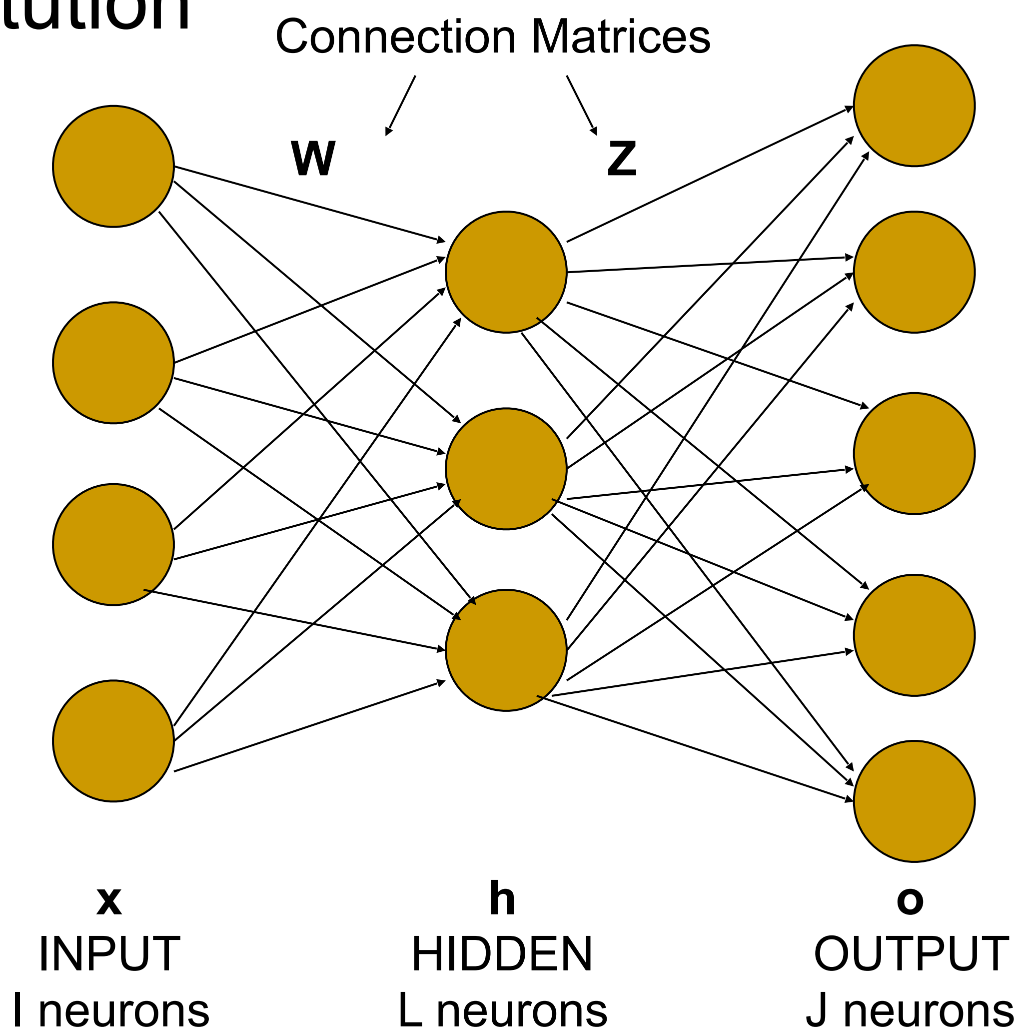






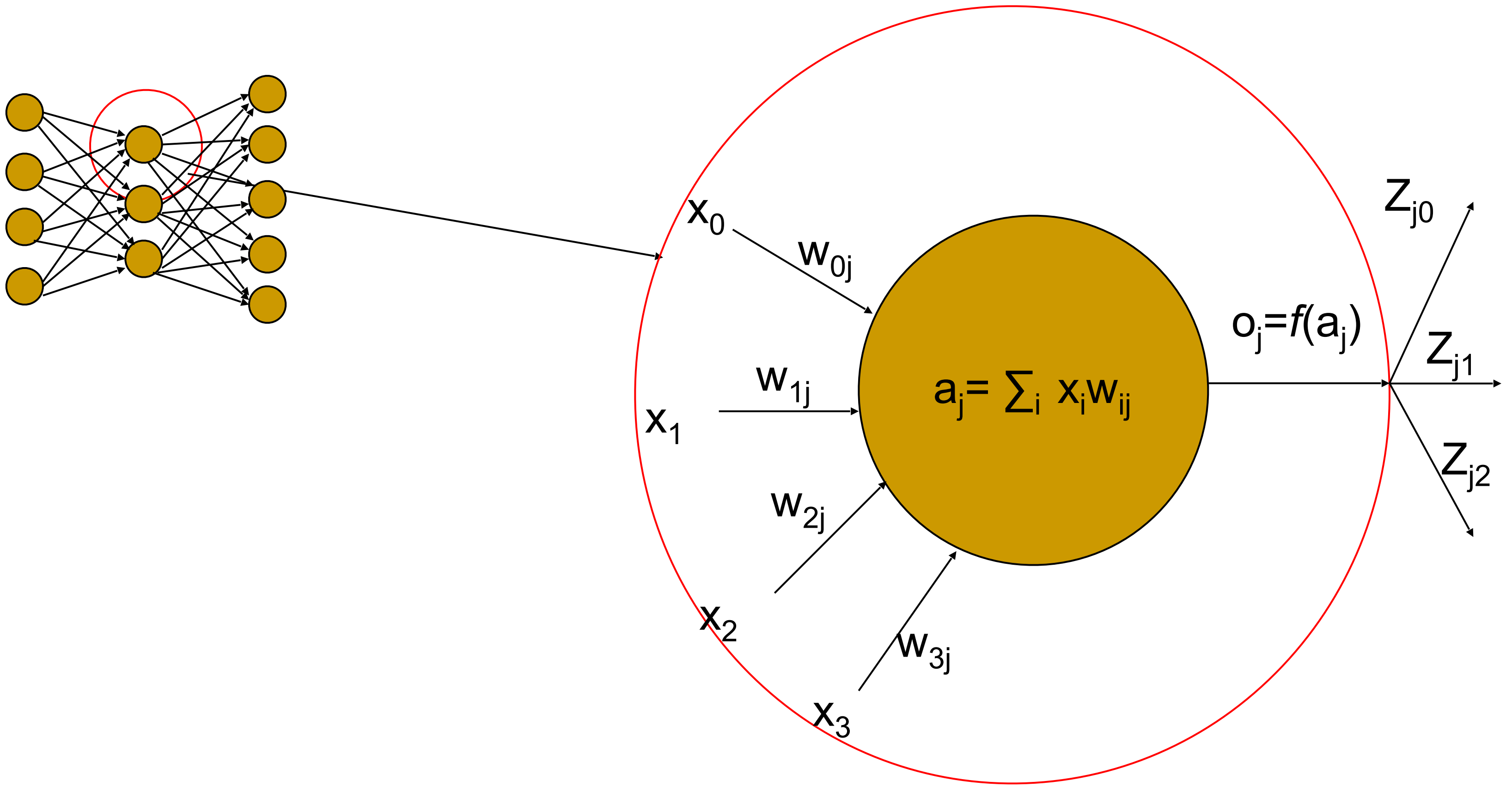
Multilayer perceptron

- Constitution



Multilayer Perceptron

- Constitution



Error backpropagation

- Learning algorithm
- How it proceeds :
  - Inject an input
  - Get the output
  - Compute the error with respect to the desired output
  - Propagate this error back from the output layer to the input layer of the network
  - Just a consequence of the chaining derivative of the gradient descent

Backpropagation

- Select a derivable transfert function
  - Classicaly used : The logistics

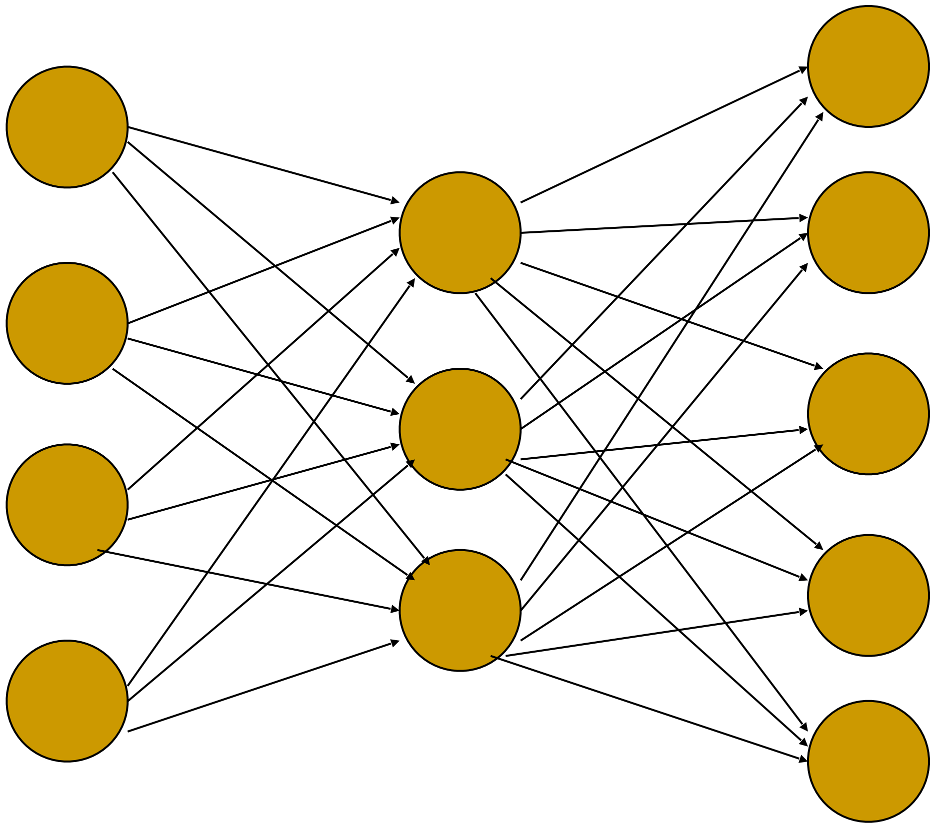
$$f(x) = \frac{1}{1+e^{-x}}$$

  - And its derivative

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

Backpropagation

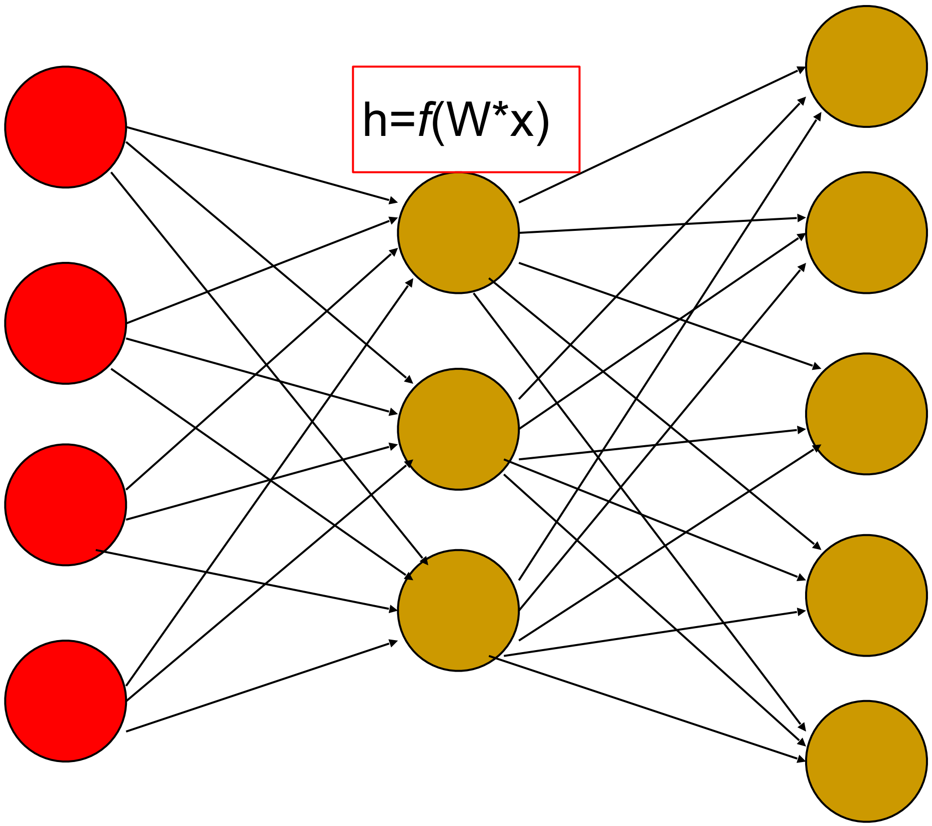
- The algorithm



1. Inject an entry

Backpropagation

- Algorithm

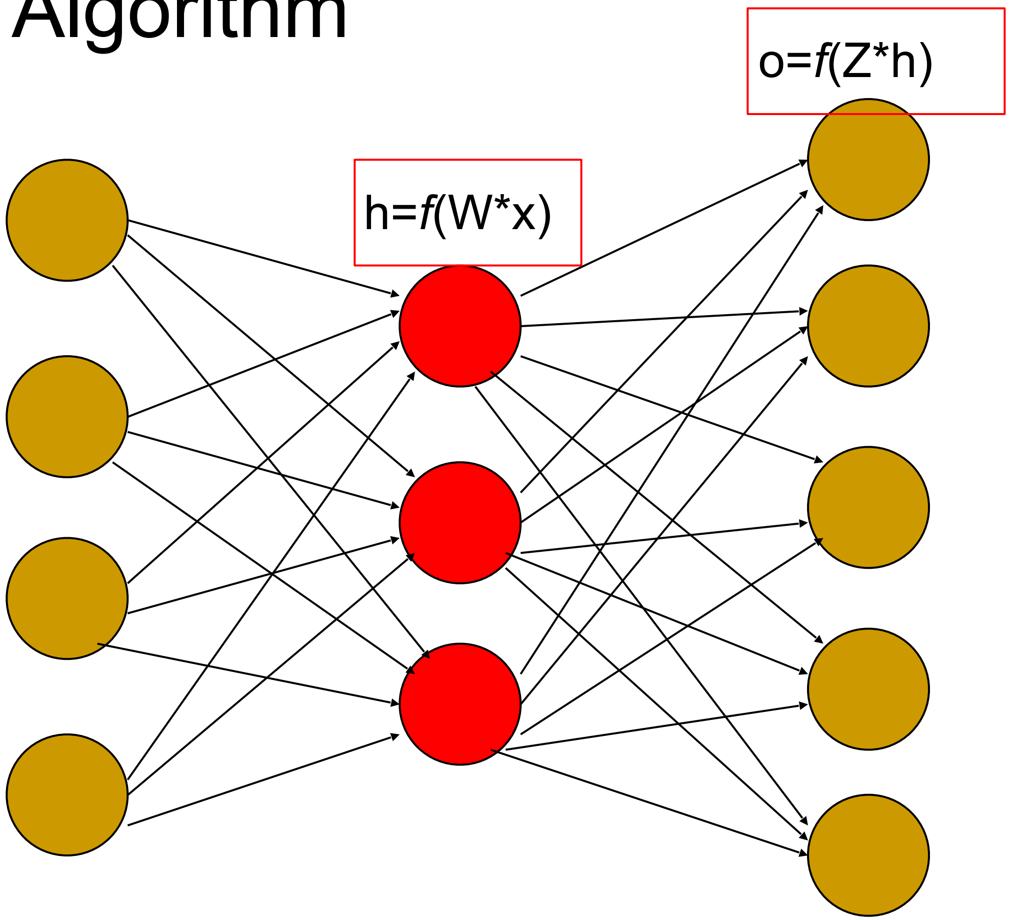


1. Inject an entry

2. Compute the intermediate h

Backpropagation

- Algorithm



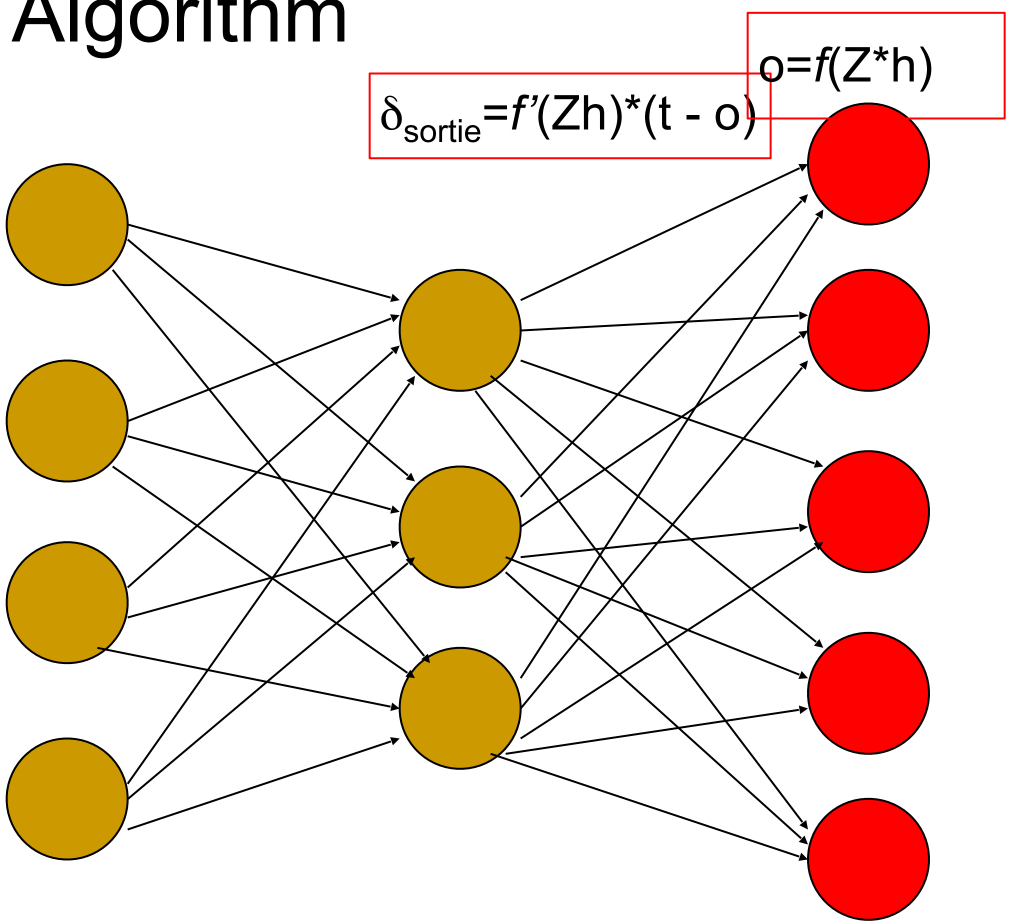
1. Inject an entry

2. Compute the intermediate h

3. Compute the output o

Backpropagation

- Algorithm

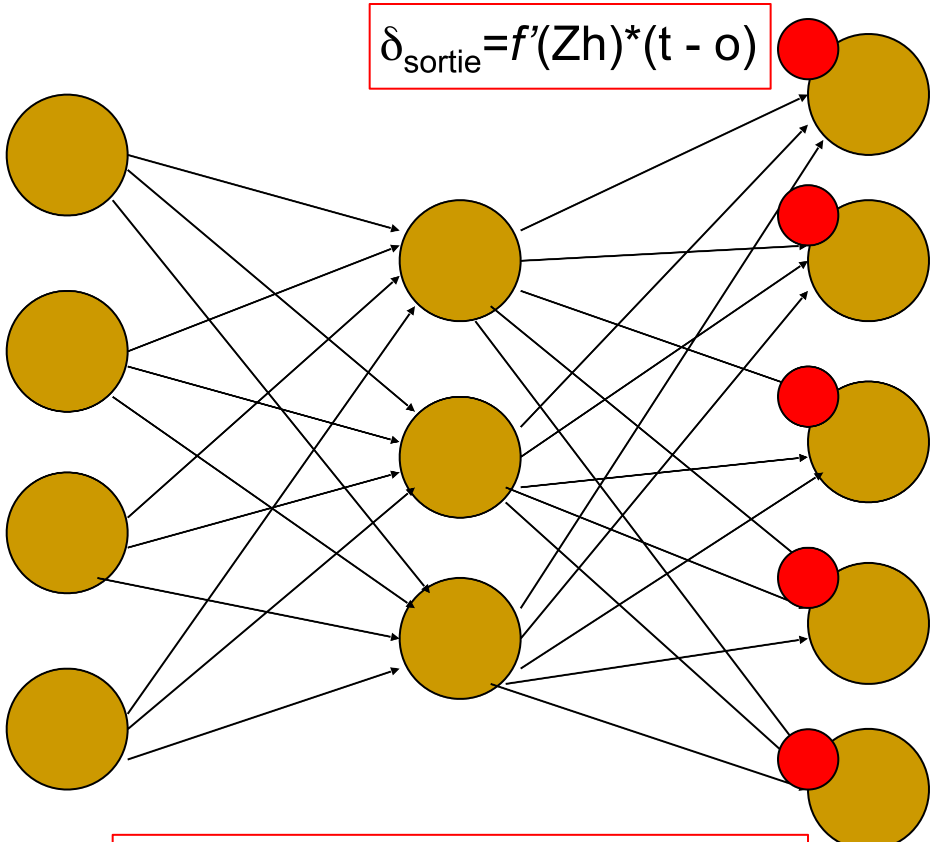


1. Inject an entry
2. Compute the intermediate h
3. Compute the output o
4. Compute the error output

Backpropagation

- Algorithm

$\boxed{\delta_{sortie}=f'(Zh)*(t - o)}$



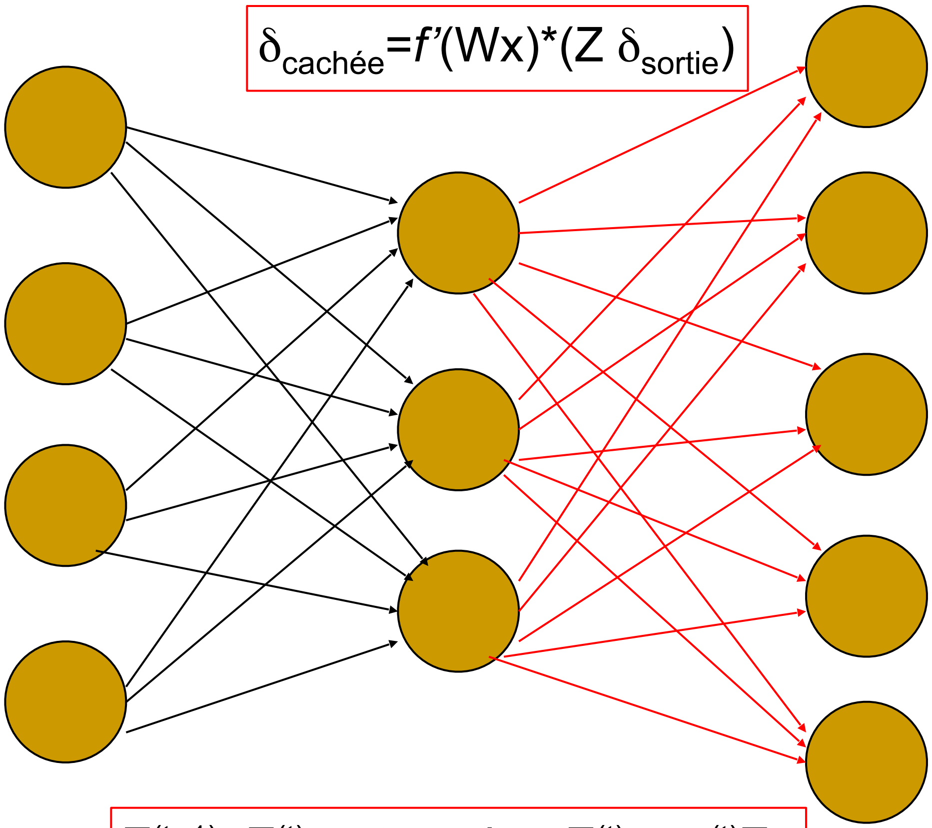
$\boxed{Z^{(t+1)}=Z^{(t)}+n\ \delta_{sortie}\ h = Z^{(t)} + \Delta^{(t)}Z}$

1. Inject an entry
2. Compute the intermediate h
3. Compute the output o
4. Compute the error output
5. Adjust Z on the basis of the error

Backpropagation

- Algorithm

$$\delta_{cachée}=f'(Wx)*(Z\ \delta_{sortie})$$



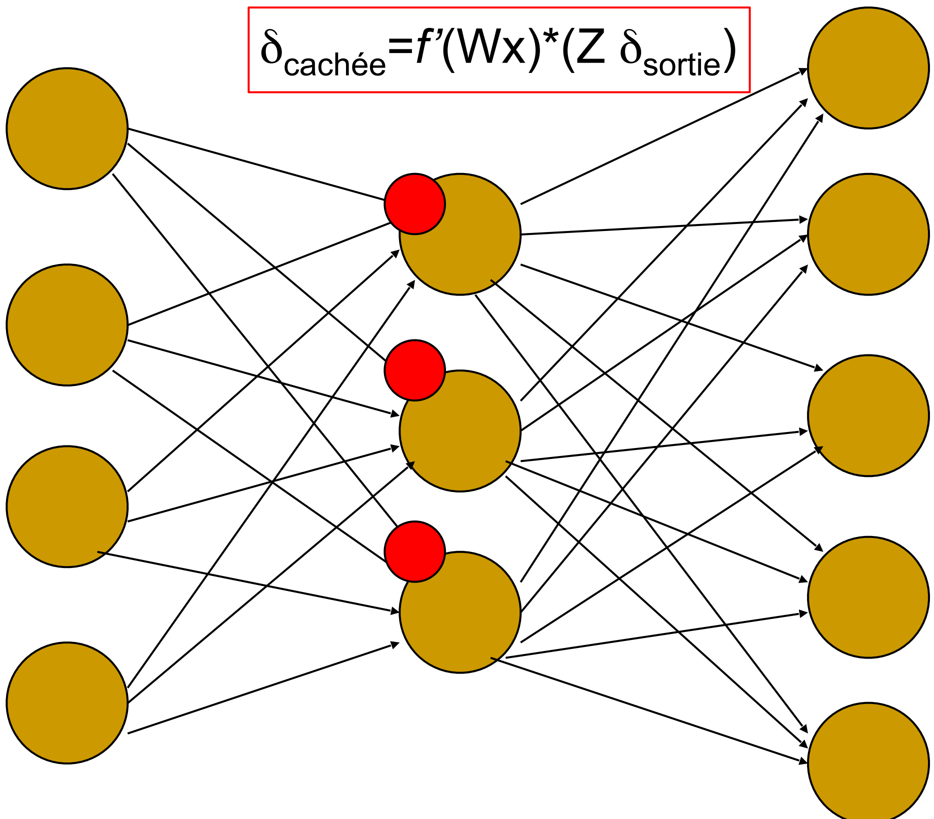
$$Z^{(t+1)}=Z^{(t)}+n\ \delta_{sortie}\, h = Z^{(t)} + \Delta^{(t)}Z$$

1. Inject an entry
2. Compute the intermediate h
3. Compute the output o
4. Compute the error output
5. Adjust Z on the basis of the error
6. Compute the error on the hidden layer

Backpropagation

- Algorithm

$$\delta_{cachée} = f'(Wx) * (Z\ \delta_{sortie})$$

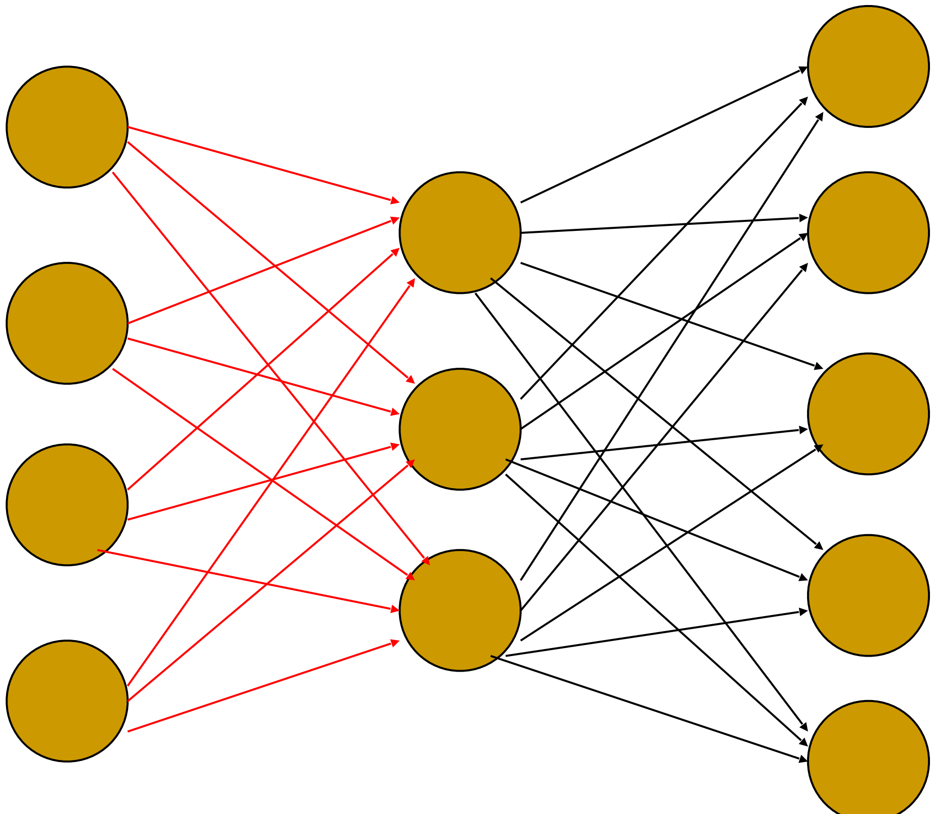


$$W^{(t+1)} = W^{(t)} + n\ \delta_{cachée}\ x = W^{(t)} + \Delta^{(t)} W$$

1. Inject an entry
2. Compute the intermediate h
3. Compute the output o
4. Compute the error output
5. Adjust Z on the basis of the error
6. Compute the error on the hidden layer
7. Adjust W on the basis of this error

Backpropagation

- Algorithm

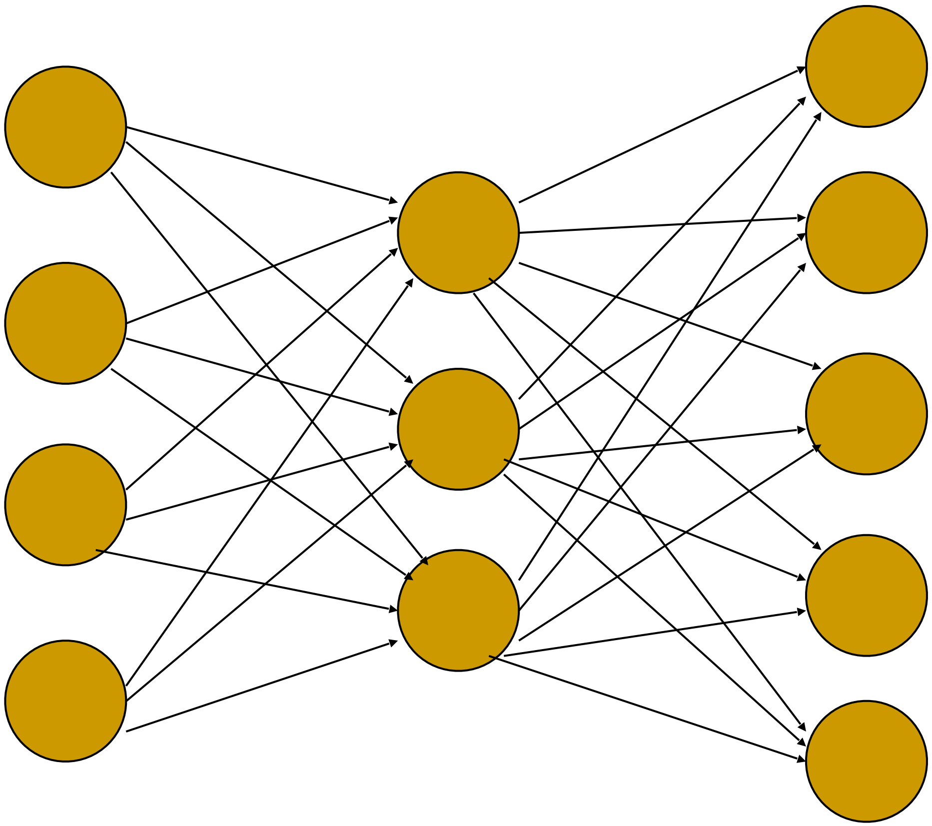


$$W^{(t+1)}=W^{(t)}+n\ \delta_{\text{cachée}}\ x = W^{(t)} + \Delta^{(t)}W$$

1. Inject an entry
2. Compute the intermediate h
3. Compute the output o
4. Compute the error output
5. Adjust Z on the basis of the error
6. Compute the error on the hidden layer
7. Adjust W on the basis of this error

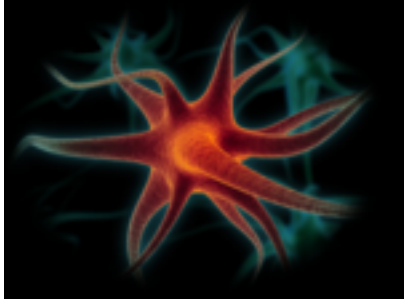
Backpropagation

- Algorithm

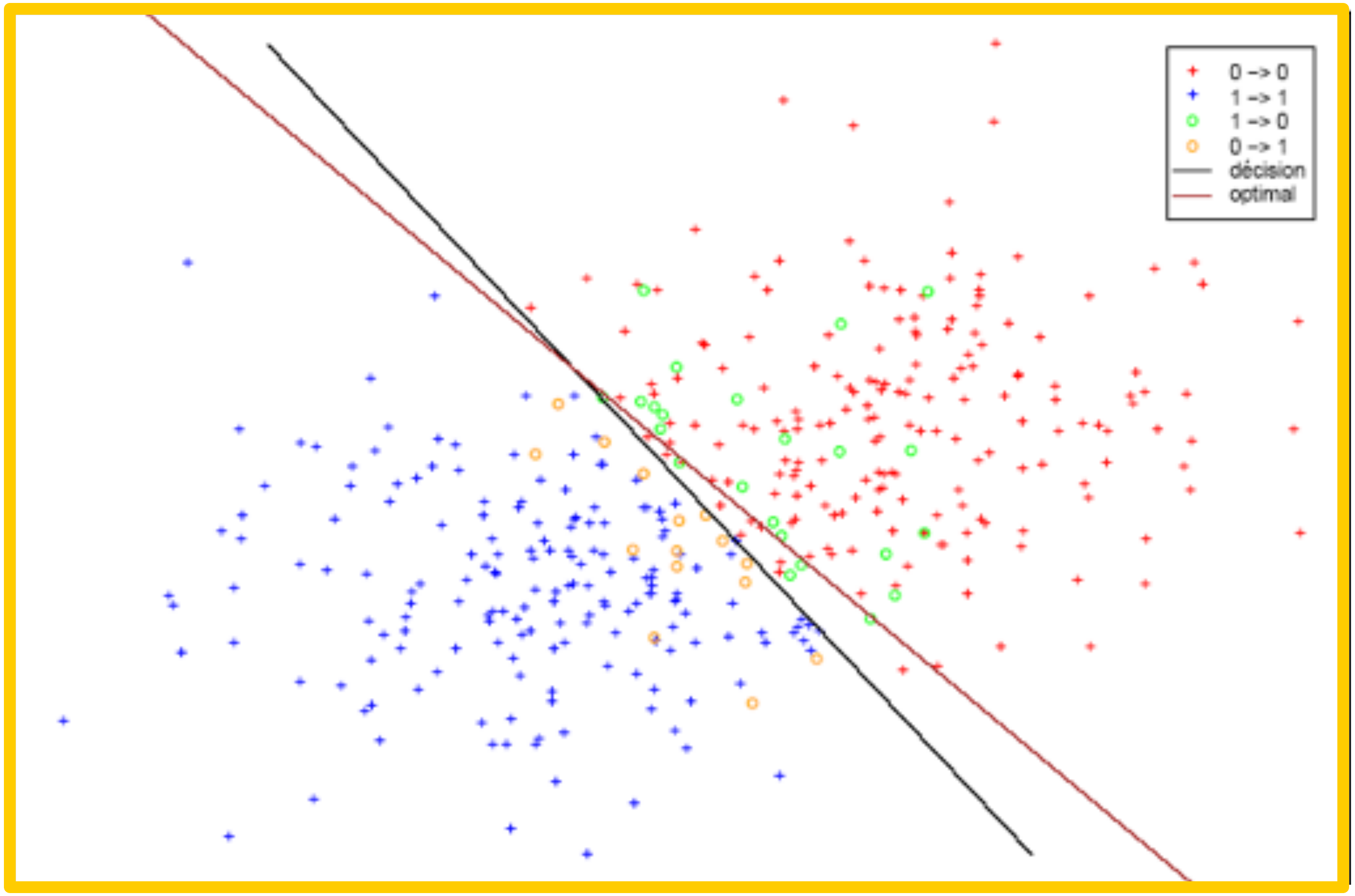


1. Inject an entry
2. Compute the intermediate h
3. Compute the output o
4. Compute the error output
5. Adjust Z on the basis of the error
6. Compute the error on the hidden layer
7. Adjust W on the basis of this error

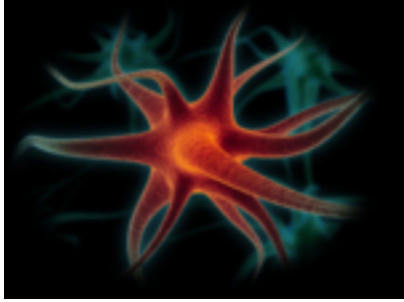
Neural network



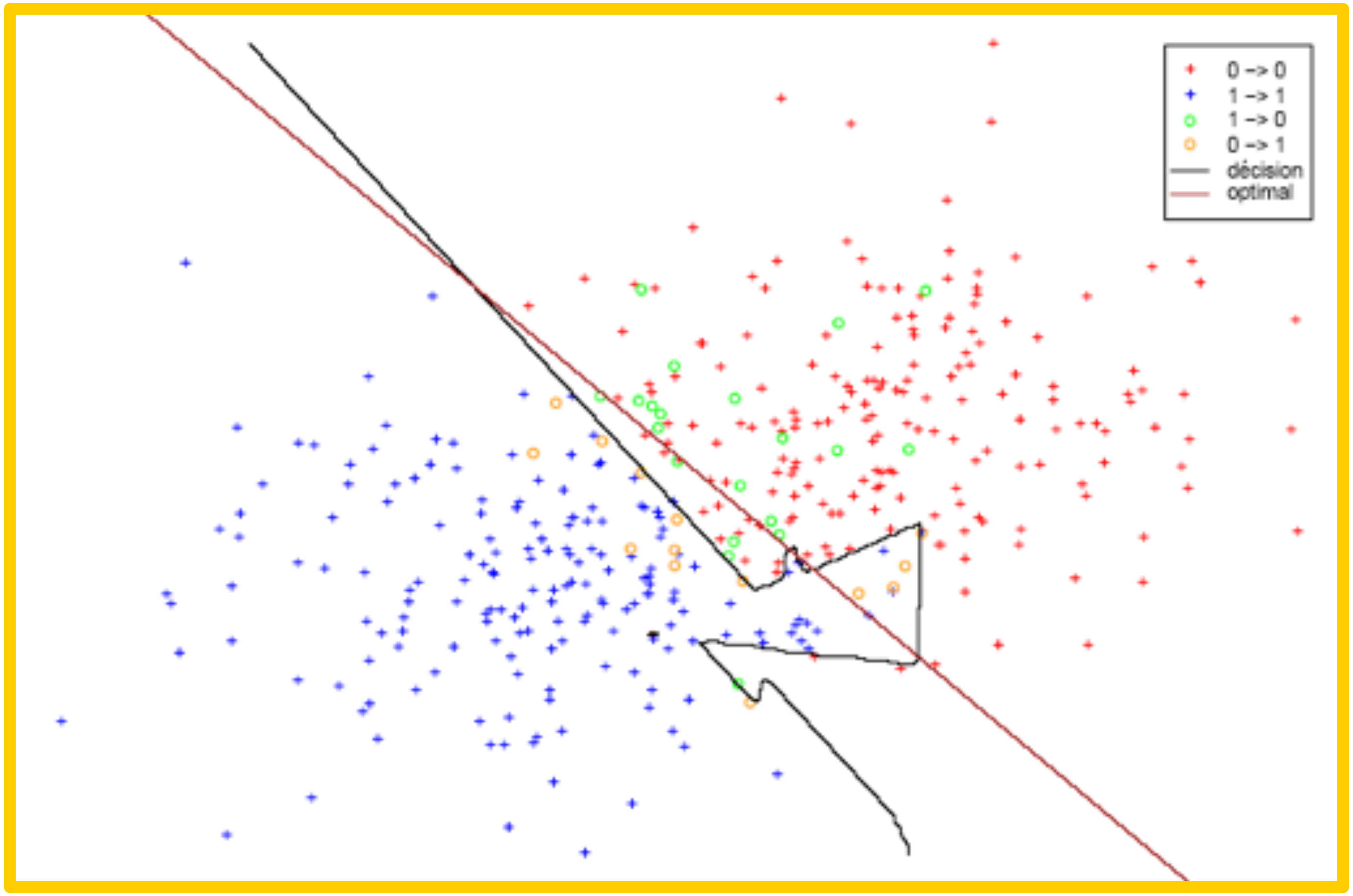
Simple linear discriminant



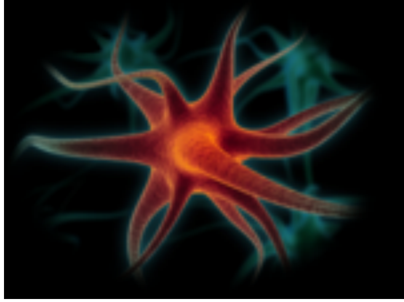
Neural networks



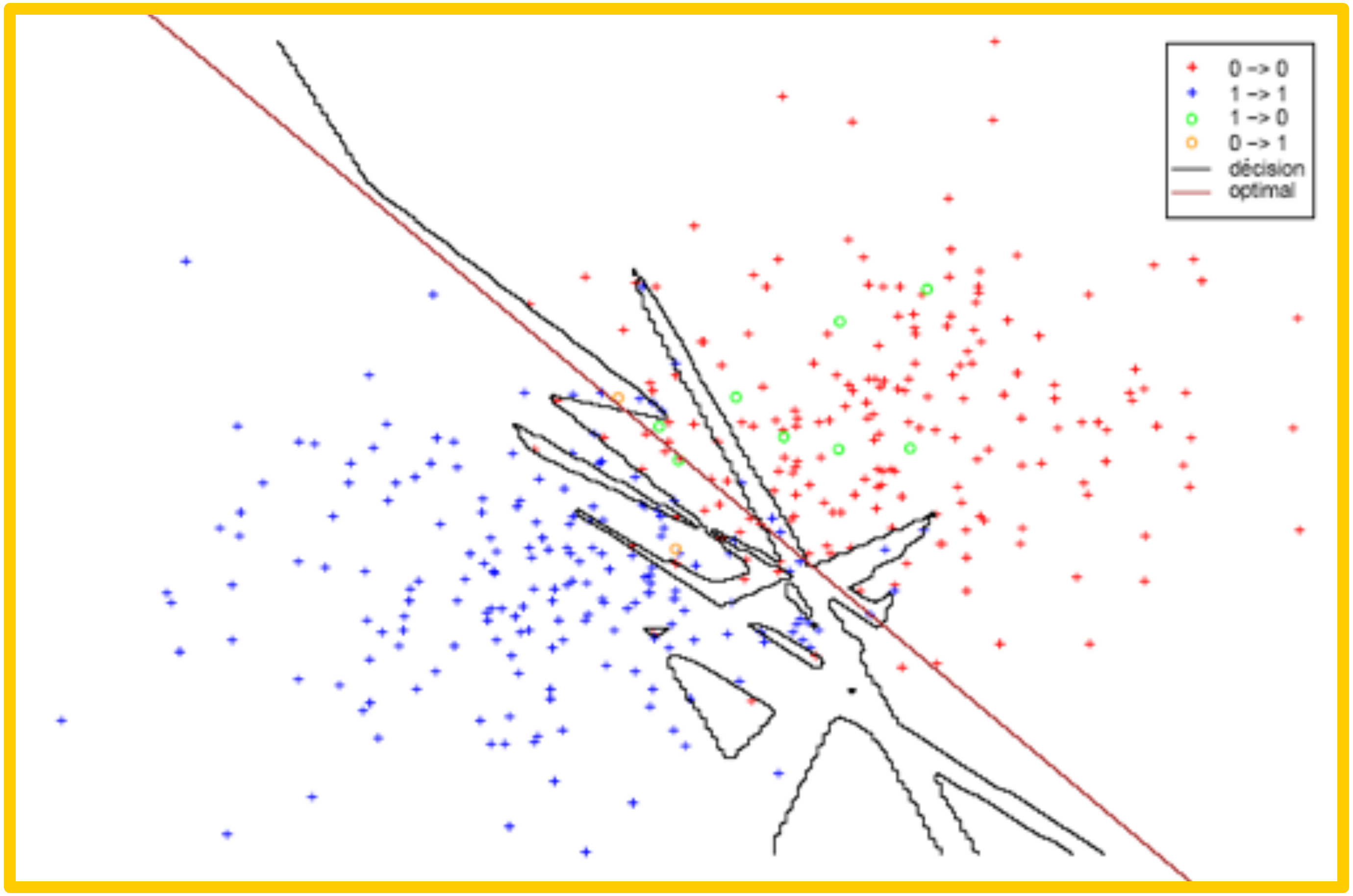
Few layers - Little learning

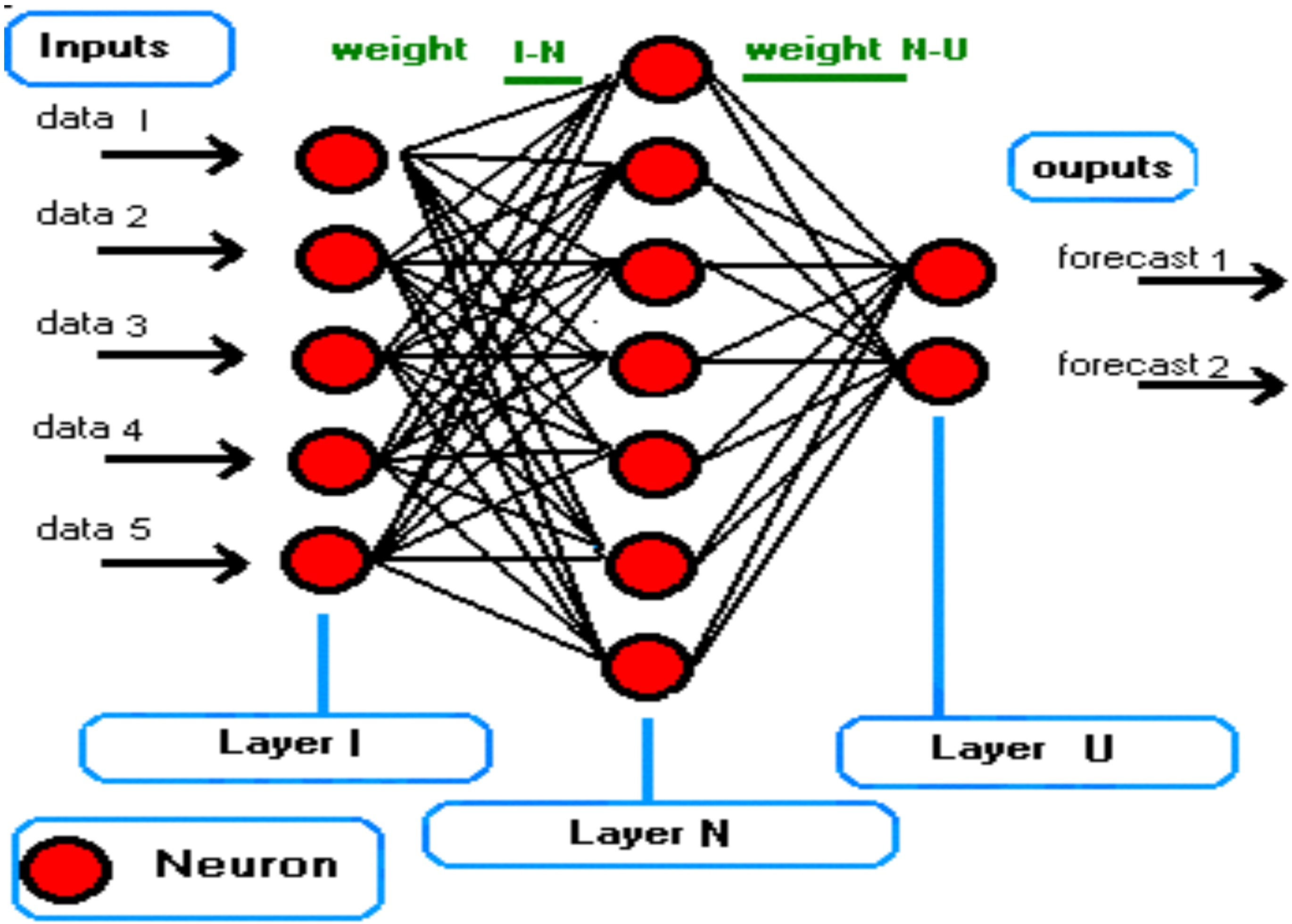


Neural networks

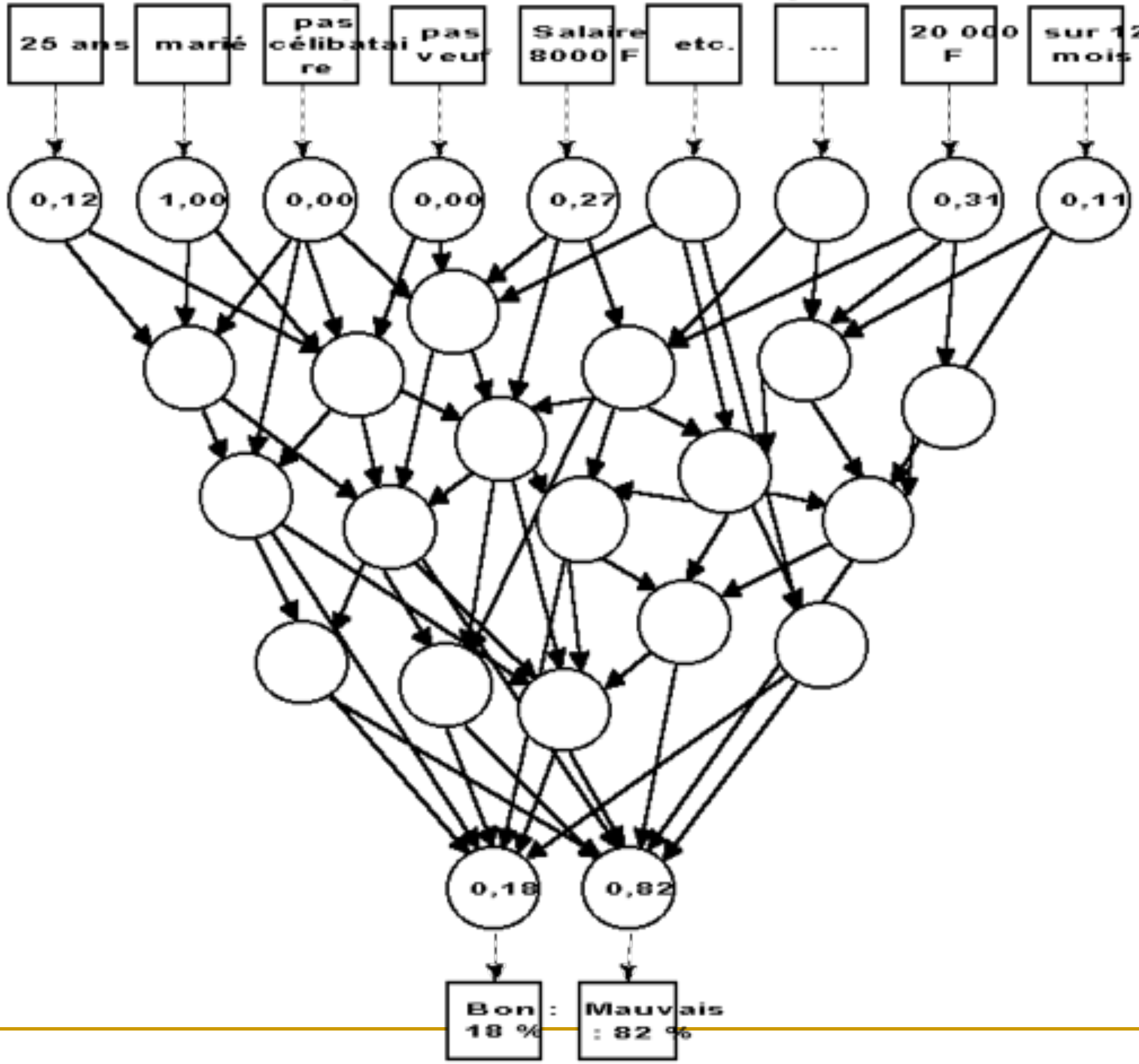


More layers - More learning



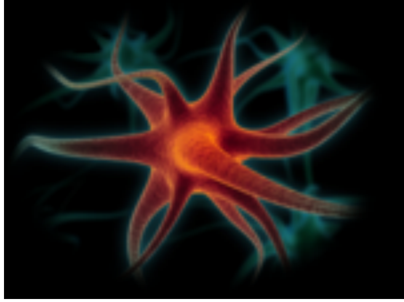

Inputs
weight
I-N
weight N-U
data I
data 2
data 3
data 4
data 5
ouputs
forecast 1
forecast 2
Layer I
Layer N
Layer U
Neuron

description de dossier de prêt



suggestion de décision

Neural networks



<u>Tricks</u>

- Favour simple NN (you can add the structure in the error)
- Few layers are enough (theoretically only one)
- Exploit cross validation…