INTRODUCTION AUX RESEAUX DE NEURONES

Gérald PETITJEAN

gerald.petitjean@eurodecision.com

BIBLIOGRAPHIE

1. F. Blayo et M. Verleysen, ***Les Réseaux de Neurones Artificiels*** ,Que Sais-Je ?, n°3042
2. J.-P. Renard, ***Réseaux neuronaux (une introduction accompagnée d'un modèle Java)***, Vuibert
3. G. Dreyfus, M. Samuelides, J.-M. Martinez, M. B. Gordon, F. Badran, S. Thiria, L. Hérault, ***Réseaux de neurones (méthodologies et applications)***, Eyrolles
4. C. M. Bishop, ***Neural networks for pattern recognition***, Oxford University
5. T. Mitchell, ***Machine learning***, Mac Graw Hill
6. A. Cornuéjols, L. Miclet, Y. Kodratoff, ***Apprentissage artificiel***, Eyrolles

Sur le net :

C. Touzet, ***Les Réseaux de Neurones Artificiels : Introduction au Connexionnisme***, 1992

OUTILS / BIBLIOTHEQUES

Outils commerciaux :

1. Matlab : « neural networks » toolbox
   http://www.mathworks.com/products/neuralnet/
2. Netral : Neuro One
   http://www.netral.com/index.html
   http://www.netral.com/logiciels/neuroone-fr.html

Outils open-source / gratuits :

1. JOONE : bibilothèque JAVA open source (licence LGPL)
   http://www.jooneworld.com/
2. Scilab : ANN toolbox
   http://www.scilab.org/
   http://www.scilab.org/contrib/displayContribution.php?fileID=165
3. Matlab : « netlab » toolbox
   http://www.ncrg.aston.ac.uk/netlab/index.php

RESEAUX DE NEURONES

• ***Connexionisme***.

• Modélisation mathématique du cerveau humain.
• Réseaux de neurones formels = réseaux d ’unités de calcul élémentaire interconnectées.

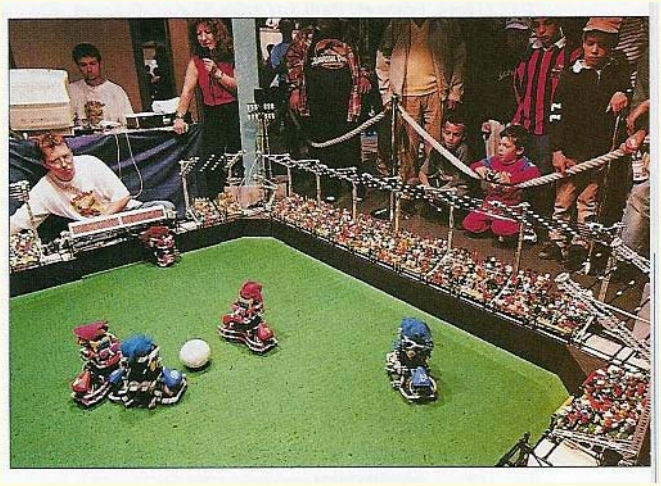
• 2 axes de recherche :
- étude et modélisation des phénomènes naturels d ’apprentissage (biologie, physiologie du cerveau)

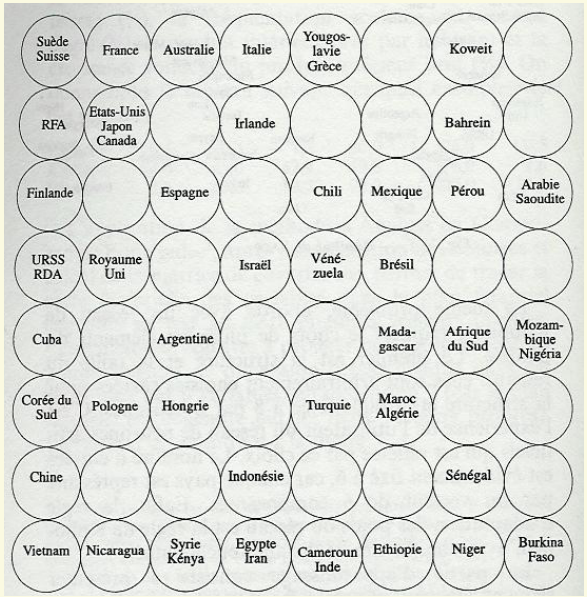
- algorithmes pour résoudre des problèmes complexes

RESEAUX DE NEURONES

Applications :

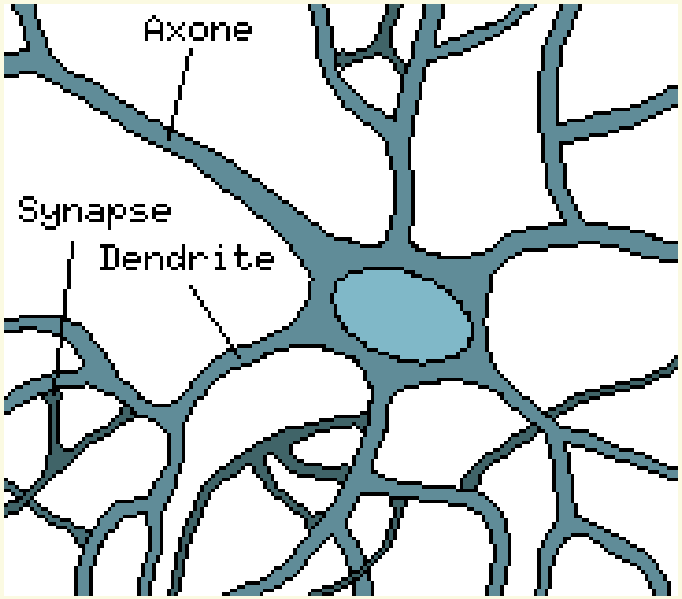
- statistiques : analyse de données / prévision / classification
- robotique : contrôle et guidage de robots ou de véhicules autonomes
- imagerie / reconnaissance de formes
- traitement du signal
- simulation de l'apprentissage





MODELE BIOLOGIQUE

• Les ***neurones*** reçoivent des signaux (impulsions électriques) par les ***dendrites*** et envoient l ’information par les ***axones***.
• Les contacts entre deux neurones (entre axone et dendrite) se font par l ’intermédiaire des ***synapses***.
• Les signaux n ’opèrent pas de manière linéaire : effet de seuil.



HISTORIQUE

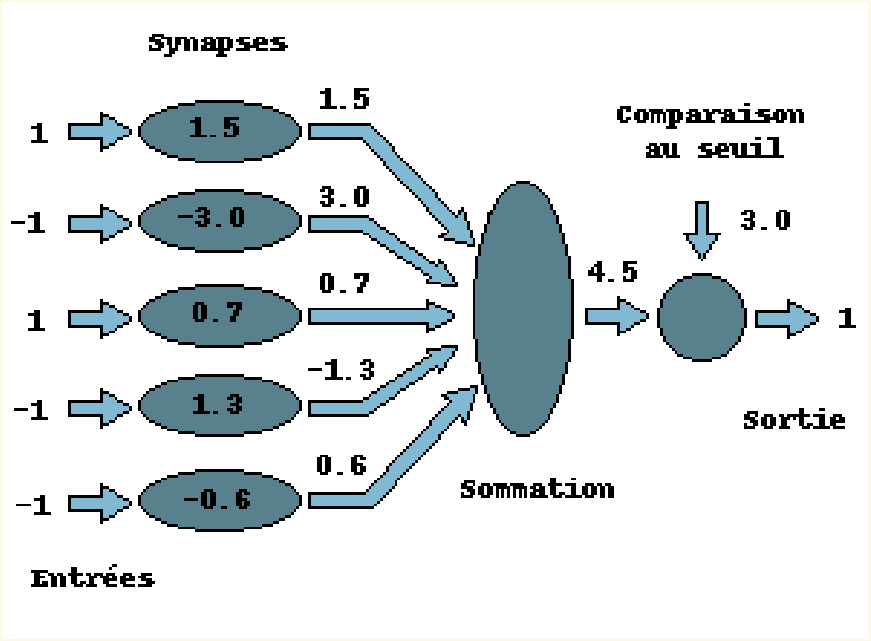
Historique :

• Mac Culloch et Pitts (1943) : définition d ’un neurone formel
• Loi de Hebb (1949)
• Rosenblatt (1958), Widrow et Hoff : modèle avec processus d ’apprentissage, perceptron
• Minsky et Papert (1969) : limites des perceptrons
• Kohonen (1972) : mémoires associatives
• Rumelhart – Mc Clelland (1980), Werbos – Le Cun : perceptron multi-couches, mécanismes d ’apprentissage performants (rétro-propagation du gradient).

NEURONE FORMEL

Principes :

- pas de notion temporelle
- coefficient synaptique : coefficient réel
- sommation des signaux arrivant au neurone
- sortie obtenue après application d ’une fonction de transfert



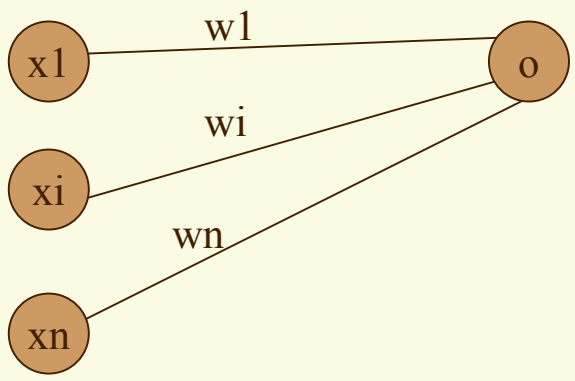
NEURONE FORMEL

Modélisation :

Le neurone reçoit les entrées x1, …, xi, …, xn.

Le potentiel d'activation du neurone p est défini comme la somme pondérée (les poids sont les coefficients synaptiques wi) des entrées.

La sortie o est alors calculée en fonction du seuil θ.



Soit : $p = x.w = x1.w1 + \ldots + xi.wi + \ldots + xn.wn$

Alors : $o = 1$ si $p > \theta$

$o = 0$ si $p \leq \theta$

DEFINITIONS

Définitions :

• Déterminer un réseau de neurones = Trouver les coefficients synaptiques.
• On parle de phase d'***apprentissage*** : les caractéristiques du réseau sont modifiées jusqu'à ce que le comportement désiré soit obtenu.

• ***Base d'apprentissage*** : exemples représentatifs du comportement ou de le fonction à modéliser. Ces exemples sont sous la forme de couples (entrée ; sortie) connus.

• ***Base d'essai*** : pour une entrée quelconque (bruitée ou incomplète), calculer la sortie. On peut alors évaluer la performance du réseau.

DEFINITIONS

Définitions :

• ***apprentissage supervisé*** : les coefficients synaptiques sont évalués en minimisant l 'erreur (entre sortie souhaitée et sortie obtenue) sur une base d 'apprentissage.

• ***apprentissage non-supervisé*** : on ne dispose pas de base d 'apprentissage. Les coefficients synaptiques sont déterminés par rapport à des critères de conformité : spécifications générales.

• ***sur-apprentissage*** : on minimise l 'erreur sur la base d 'apprentissage à chaque itération mais on augmente l 'erreur sur la base d 'essai. Le modèle perd sa capacité de généralisation : c 'est l ***'apprentissage par cœur***.

⇒ Explication : trop de variables explicatives dans le modèle ; on n 'explique plus le comportement global mais les résidus.

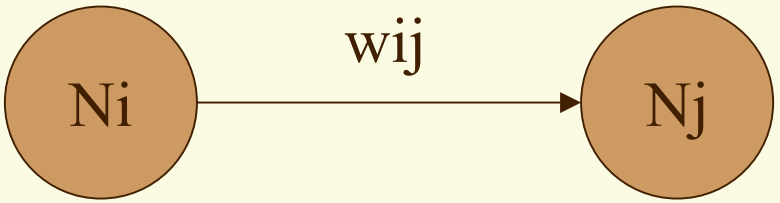
LOI DE HEBB

Réseau de neurones :

- n entrées e1, …, en
- m neurones N1, …, Nm.
- wij le coefficient synaptique de la liaison entre les neurones Ni et Nj
- une sortie o
- un seuil S
- Fonction de transfert : fonction Signe

  si $x > 0$: Signe(x) = +1

  si $x \leq 0$: Signe(x) = –1



LOI DE HEBB

Principe :

Si deux neurones sont activés en même temps, alors la force de connexion augmente.

Base d 'apprentissage :

On note S la base d 'apprentissage.

S est composée de couples (e, c) où :

e est le vecteur associé à l 'entrée (e1, …, en)

c la sortie correspondante souhaitée

Définitions :

ai est la valeur d 'activation du neurone Ni.

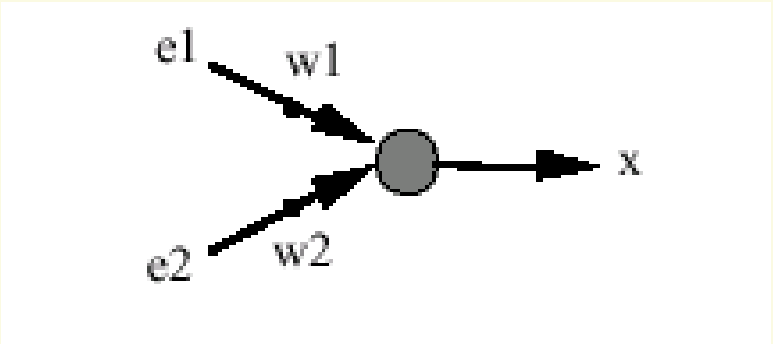
LOI DE HEBB

Algorithme :

- μ est une constante positive.
- Initialiser aléatoirement les coefficients wi
- Répéter :
  - Prendre un exemple (e, c) dans S
  - Calculer la sortie o du réseau pour l 'entrée e
  - Si c ≠ o
    - Modification des poids wij :
      - wij = wij + μ ∗ (ai ∗ aj)
    - Fin Pour
  - Fin Si
- Fin Répéter

LOI DE HEBB : exemple

Exemple :



e_1	e_2	x	
1	1	1	(1)
1	-1	1	(2)
-1	1	-1	(3)
-1	-1	-1	(4)

$\mu = 1$

Conditions initiales : coefficients et seuils nuls

LOI DE HEBB : exemple

Exemple :

- Exemple (1) : $o = \text{Signe}(w1 * e1 + w2 * e2) = \text{Signe}(0) = -1$

  $o = -1 \neq 1 = x$

  $\Rightarrow$ $w1 = w1 + e1 * x = 1$

  $w2 = w2 + e2 * x = 1$
- Exemple (2) : $o = \text{Signe}((w1 * e1 + w2 * e2) = \text{Signe}(0) = -1$

  $o = -1 \neq 1 = x$

  $\Rightarrow$ $w1 = w1 + e1 * x = 2$

  $w2 = w2 + e2 * x = 0$
- Exemples (3) et (4) OK
- Exemples (1) et (2) OK $\Rightarrow$ STOP

LOI DE HEBB

Remarque :

• Calcul des coefficients wij sans utiliser l 'algorithme itératif.

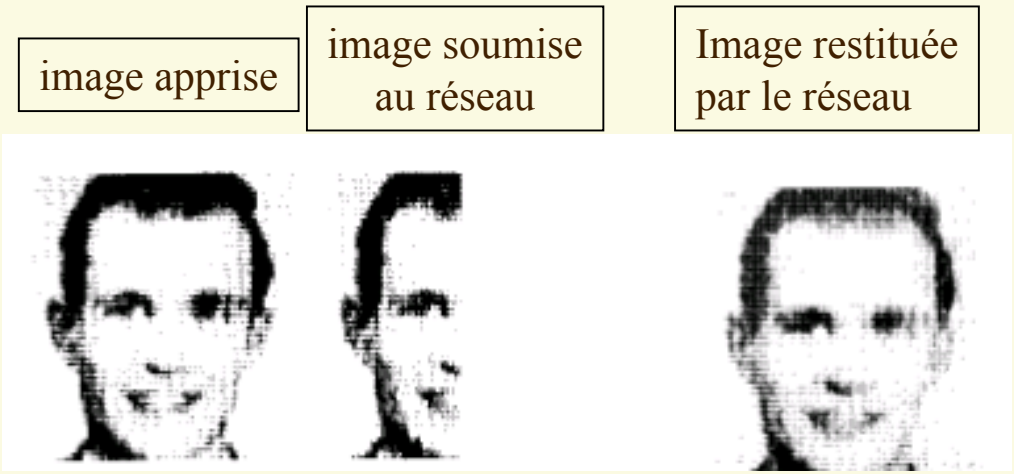
• Principe : les coefficients sont initialisés à 0, μ vaut 1, et on présente tous les exemples de la base d 'apprentissage.

• wij = Σ[(e,c) dans S] (ai ∗ aj)

LOI DE HEBB

Application : mémoires auto-associatives

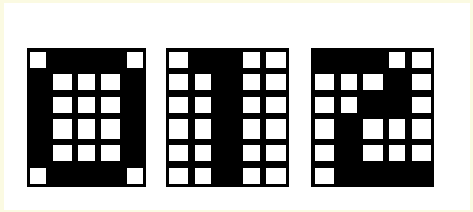
- Kohonen (1977)
- Reconstruction de données :
  - en entrée : une information partielle ou bruitée
  - en sortie : le système complète ou corrige l ’information

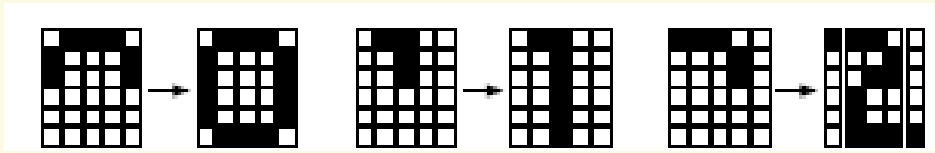


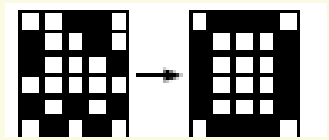
EXERCICE : Loi de Hebb

Reconstitution d ’images :

les chiffres 0, 1 et 2



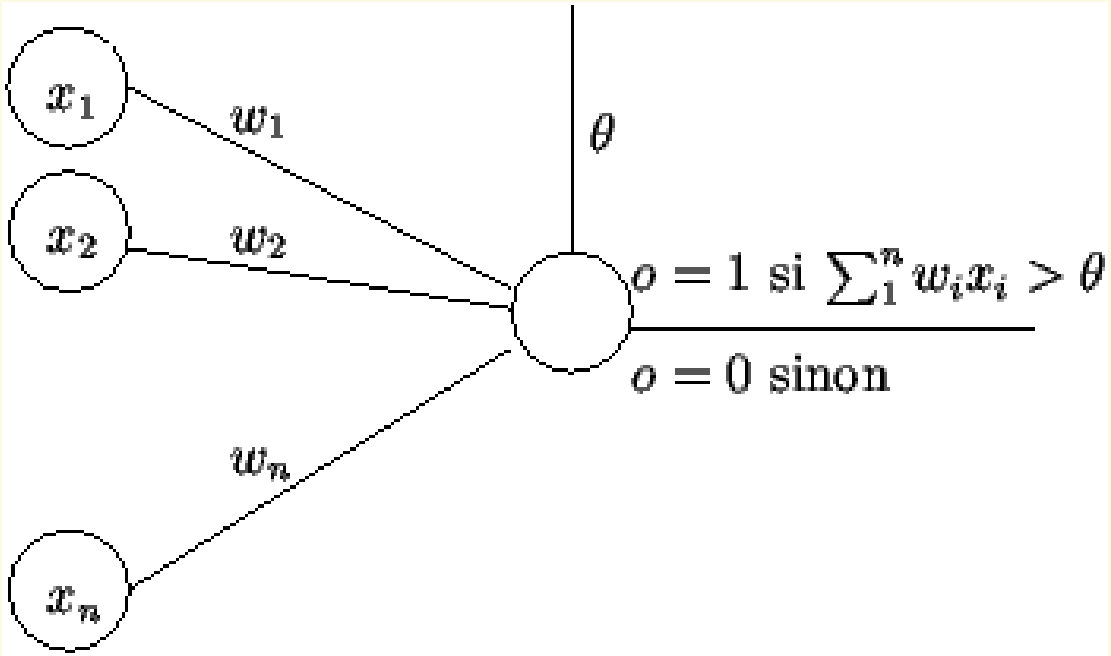




PERCEPTRON

Perceptron linéaire à seuil :

- n entrées x1, …, xn
- n coefficients synaptiques w1, …, wn
- une sortie o
- un seuil θ



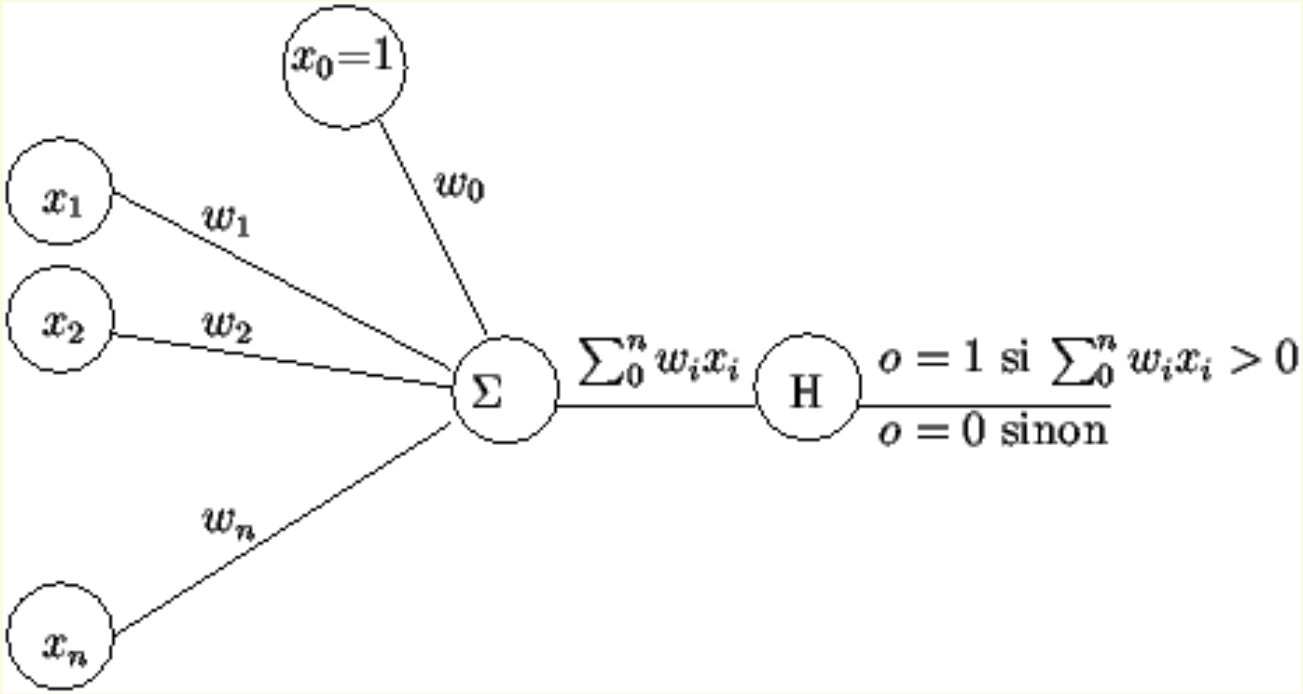
PERCEPTRON

On ajoute une entrée supplémentaire x0 (le biais), avec le coefficient synaptique suivant : w0 = – θ

On associe comme fonction de transfert la fonction de Heavyside :

f(x) = 1 si x > 0

f(x) = 0 sinon



PERCEPTRON

Apprentissage par l 'algorithme du perceptron

On note S la base d 'apprentissage.

S est composée de couples (x, c) où :

x est le vecteur associé à l 'entrée (x0, x1, …, xn)

c la sortie correspondante souhaitée

On cherche à déterminer les coefficients (w0, w1, …, wn).

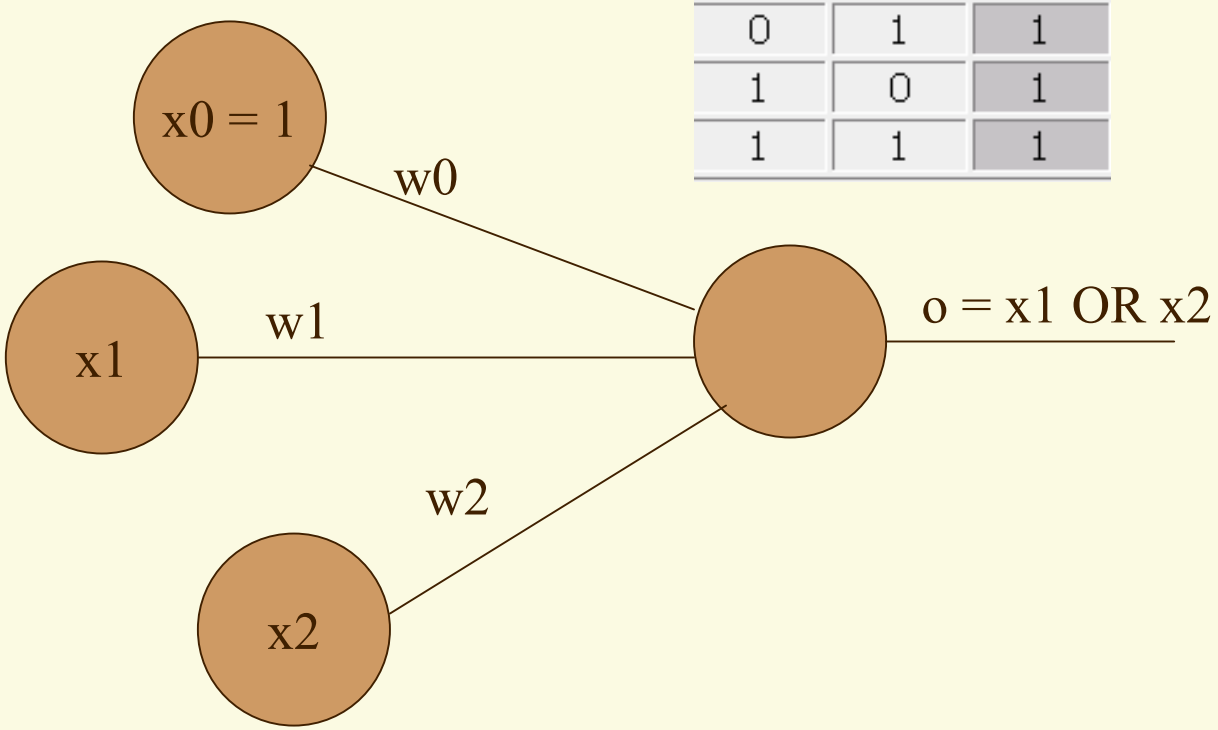
- Initialiser aléatoirement les coefficients wi.
- Répéter :
  - Prendre un exemple (x, c) dans S
  - Calculer la sortie o du réseau pour l 'entrée x
  - Mettre à jour les poids :
    - Pour i de 0 à n :
      - wi = wi + ε * (c – o) * xi
    - Fin Pour
- Fin Répéter

PERCEPTRON : exemple

Apprentissage par l 'algorithme du perceptron : exemple

Apprentissage du OU logique

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1



PERCEPTRON : exemple

Apprentissage par l 'algorithme du perceptron : exemple

ε = 1

x0 vaut toujours 1

Initialisation : w0 = 0 ; w1 = 1 ; w2 = – 1

Étape	w_0	w_1	w_2	Entrée	$\Sigma_0^2 w_i x_i$	o	c	w_0	w_1	w_2
init								0	1	-1
1	0	1	-1	100	0	0	0	0+0x1	1+0x0	-1+0x0
2	0	1	-1	101	-1	0	1	0+1x1	1+1x 0	-1+1x1
3	1	1	0	110	2	1	1	1	1	0
4	1	1	0	111	2	1	1	1	1	0
5	1	1	0	100	1	1	0	1+(-1)x1	1+(-1)x0	0+(-1)x0
6	0	1	0	101	0	0	1	0+1x1	1+1x0	0+1x1
7	1	1	1	110	2	1	1	1	1	1
8	1	1	1	111	3	1	1	1	1	1
9	1	1	1	100	1	1	0	1+(-1)x1	1+(-1)x0	1 +(-1)x0
10	0	1	1	101	1	1	1	0	1	1

Donc : w0 = 0 ; w1 = 1 ; w2 = 1

Ce perceptron calcule le OU logique pour tout couple (x1 ; x2)

PERCEPTRON

Apprentissage par l 'algorithme du perceptron : remarques

- ε bien choisi, suffisamment petit
- Si ε trop grand : risque d 'oscillation autour du minimum
- Si ε trop petit : nombre élevé d 'itérations
- En pratique : on diminue graduellement ε au fur et à mesure des itérations

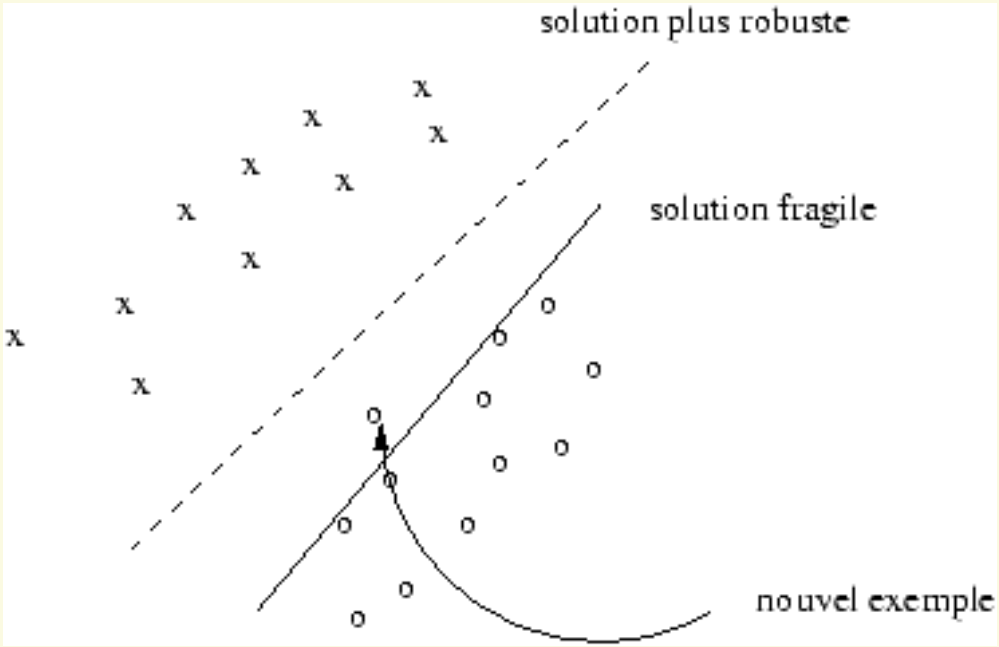
PERCEPTRON

Apprentissage par l 'algorithme du perceptron : remarques

• Si l 'échantillon n 'est pas linéairement séparable, l 'algorithme ne converge pas.

• L 'algorithme peut converger vers plusieurs solutions (selon les valeurs initiales des coefficients, la valeur de ε, l 'ordre de présentation des exemples).

• La solution n 'est pas robuste : un nouvel exemple peut remettre en cause le perceptron appris.



Optimisation : gradient (rappels)

Problème :

Trouver le minimum de la fonction f continue et dérivable : $x \rightarrow f(x)$

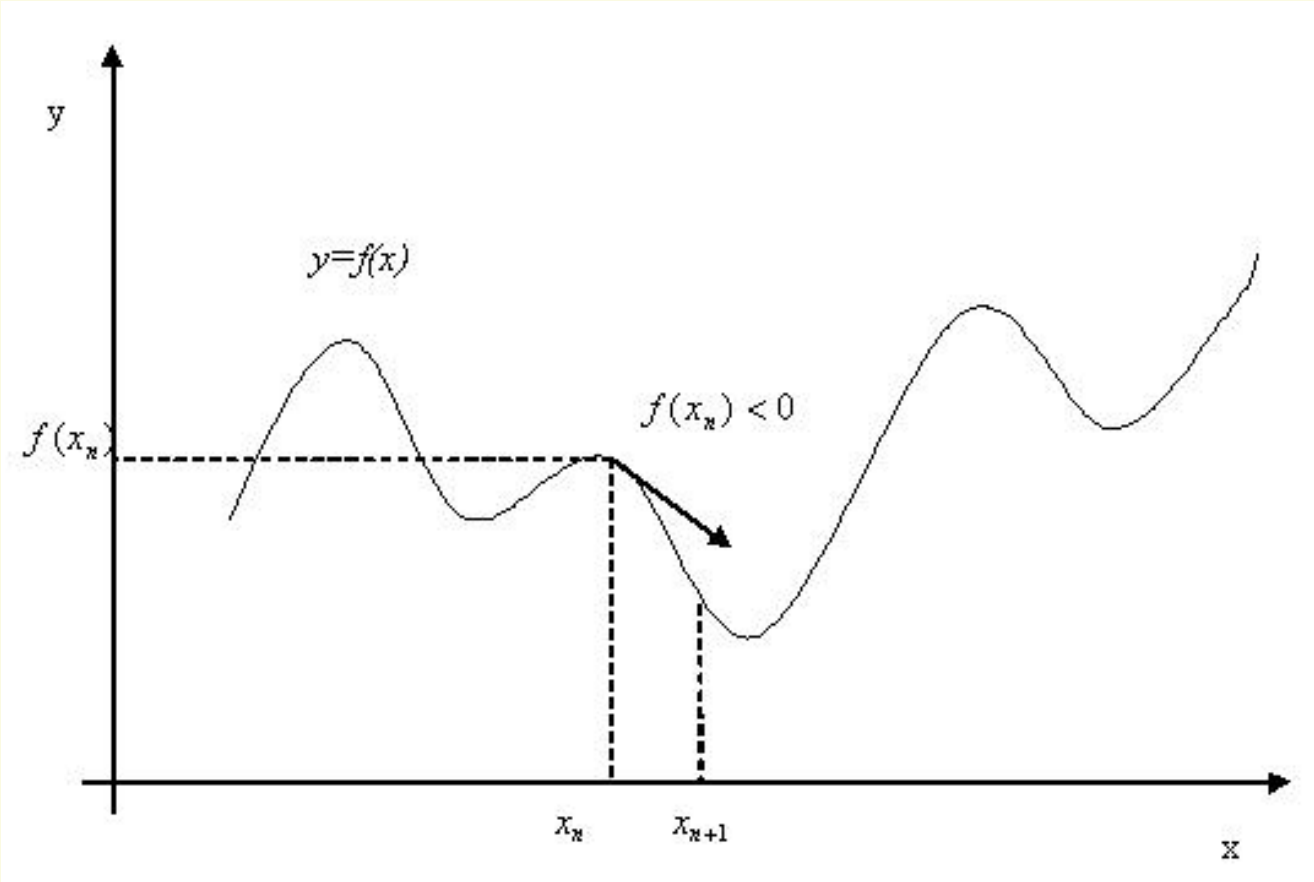
On construit la suite :

Xn telle que :

- Ont part d'une valeur initialeX0 quelconque
- $Xn+1 = Xn - \varepsilon * f'(Xn)$, avec ε valeur réelle non nulle « bien choisie » entre 0 et 1

Optimisation : gradient (rappels)

Interprétation graphique :



Optimisation : gradient (rappels)

Remarques :

- Le choix de ε est empirique
- Si ε trop petit : le nombre d'itérations est trop grand
- Si ε est trop grand : les valeurs de la suite risquent d'osciller ⇒ pas de convergence
- Rien ne garantit que le minimum trouvé est un minimum global

Optimisation : gradient (rappels)

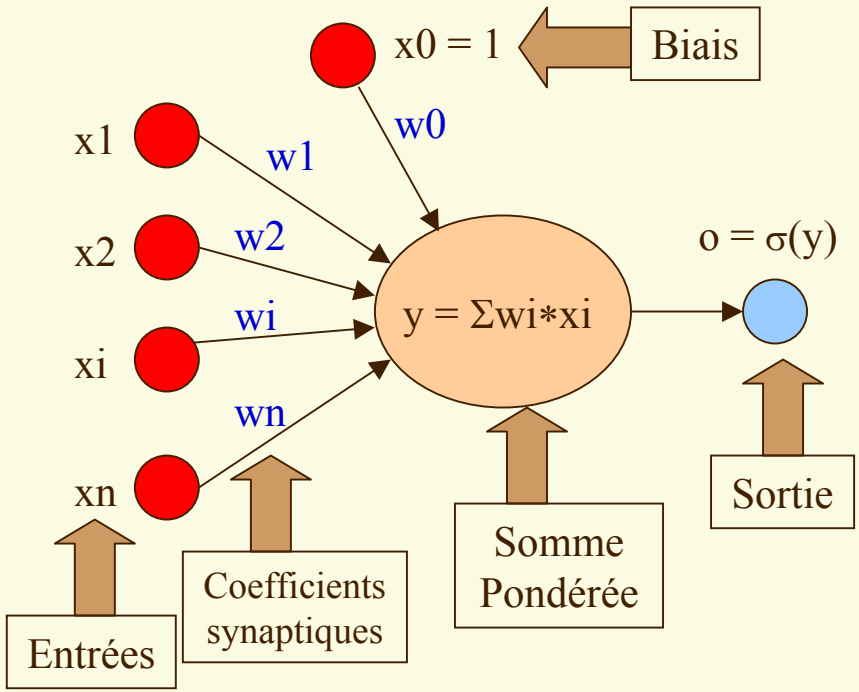
Dans la pratique :

- Critères d'arrêt :
  - nombre d'itérations max.
  - norme(f(Xi+1) – f(Xi)) / norme (Xi+1 – Xi) inférieure à η (valeur très petite)
- ε est réajusté à chaque itération
  - valeur initiale ε_0 (0.01par exemple)
  - 1-ère stratégie :
    - si f(Xi+1) > f(Xi) i.e. f augmente, on augmente ε de 10%
    - Sinon i.e. f diminue, on diminue ε en le divisant par 2
  - 2-ème stratégie : on diminue la valeur de ε toutes les K itérations

PERCEPTRON : REGLE DELTA GENERALISEE

Fonctionnement d ’un neurone :

- somme pondérée des signaux reçus (en tenant compte du biais)
- puis application d ’une fonction de transfert (ou d ’activation) : sigmoïde log, sigmoïde tangente hyperbolique, linéaire



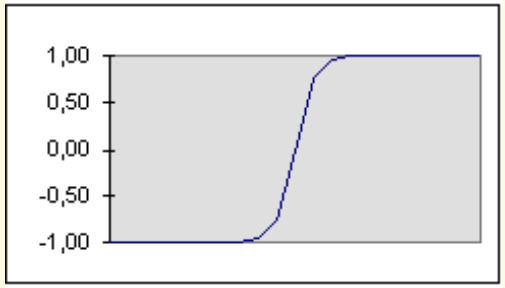
On note :
- y = x.w
- o = σ(x.w)

X0 = 1 : ***biais***
Représente le seuil d ’activation du neurone

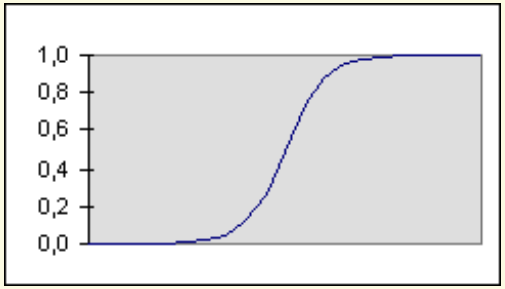
PERCEPTRON : REGLE DELTA GENERALISEE

Fonctions de transfert :

- Sigmoïde tangente hyperbolique : ***tansig***
  - sorties entre -1 et +1
  - $\sigma(x) = \tanh(x)$ et $\sigma'(x) = 1 - \tanh^2(x) = 1 - \sigma^2(x)$



- Sigmoïde log : ***losig***
  - sorties entre 0 et 1
  - $\sigma(x) = e^x/(e^x+1) = 1/(1+e^{-x})$ et $\sigma'(x) = \sigma(x)*(1-\sigma(x))$



- Linéaire : ***purelin***
  - $\sigma(y) = y$ et $\sigma'(y) = 1$

PERCEPTRON : REGLE DELTA GENERALISEE

Apprentissage par descente de gradient :

Soient le vecteur des entrées x et le vecteur des coefficients synaptiques w.

La sortie vaut alors : $o = \sigma(x.w) = \sigma(x0.w0 + \ldots + xn.wn)$,

σétant une fonction de transfert continue et dérivable.

Posons : $y = x.w$

Soit S la base d 'apprentissage composée de couples (x, c), où c est la sortie attendue pour x.

On définit ainsi l 'erreur sur le réseau pour la base d 'apprentissage S :

$E(w) = 1/2\ \Sigma[(x,c) \text{ dans } S]\ (c - o)^2$

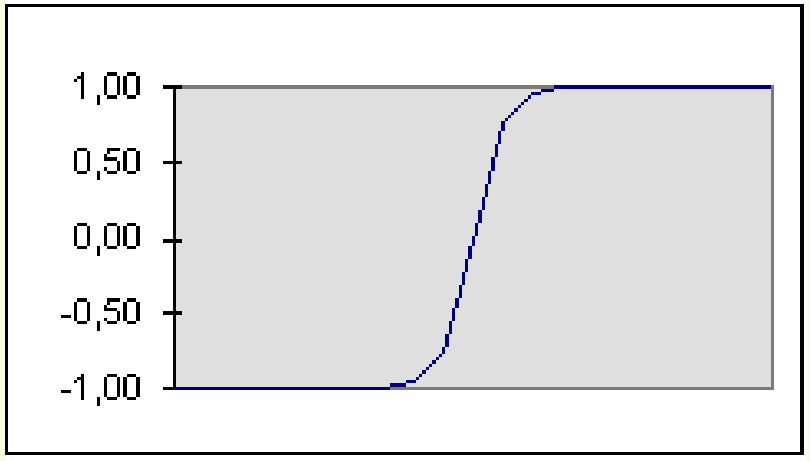
Problème : trouver w qui minimise E(w).

$\Rightarrow$ Méthode du gradient.

PERCEPTRON : REGLE DELTA GENERALISEE

- Fonction de transfert $\sigma(y)$
- Par exemple : σ est la tangente hyperbolique (sigmoïde tansig)

$$\sigma(y) = \tanh(y) \text{ et } \sigma'(y) = 1 - \tanh^2(y) = 1 - \sigma^2(y)$$



- La sortie o calculée par le perceptron pour une entrée x est :

$$o = \sigma(x.w)$$

PERCEPTRON : REGLE DELTA GENERALISEE

Méthode du gradient (rappel) :

$$x_{n+1} = x_n - \varepsilon * f'(x_n) = x_n + \Delta x_n$$

$$\frac{\partial E(W)}{\partial wi} = \frac{1}{2}\frac{\partial}{\partial wi}\sum_s (c-o)^2 = \frac{1}{2}\sum_s \frac{\partial}{\partial wi}(c-o)^2$$

$$\frac{\partial E(W)}{\partial wi} = \frac{1}{2}\sum_s 2(c-o)\frac{\partial}{\partial wi}(c-o) = \sum_s (c-o)\frac{\partial}{\partial wi}(c-\sigma(x\cdot w))$$

$$Or : \frac{\partial}{\partial wi}(c-\sigma(x.w)) = \frac{\partial}{\partial wi}(c-\sigma(y)) = \frac{\partial}{\partial y}(c-\sigma(y)) * \frac{\partial y}{\partial wi} = -\sigma'(y) * xi$$

$$\frac{\partial E(W)}{\partial wi} = \sum_s (c-o)(-xi)\sigma'(x\cdot w)$$

$$D'où : \Delta wi = -\varepsilon * \frac{\partial E(W)}{\partial wi} = \varepsilon \sum_s xi * (c-o) * \sigma'(x\cdot w)$$

PERCEPTRON : REGLE DELTA GENERALISEE

Apprentissage par descente de gradient : algorithme

- Initialiser aléatoirement les coefficients wi.
- Répéter :
  - Pout tout i :
    - $\Delta wi = 0$
  - Fin Pour
  - Pour tout exemple (x, c) dans S
    - Calculer la sortie o du réseau pour l 'entrée x
    - Pout tout i :
      - $\Delta wi = \Delta wi + \varepsilon * (c - o) * xi * \sigma'(x.w)$
    - Fin Pour
  - Fin Pour
  - Pour tout i :
    - $wi = wi + \Delta wi$
  - Fin Pour
- Fin Répéter

PERCEPTRON : REGLE DELTA GENERALISEE

Variante de la Règle Delta généralisée :

• On ne calcule pas les variations de coefficients en sommant sur tous les exemples de S mais on modifie les poids à chaque présentation d ’exemple.

Initialiser aléatoirement les coefficients wi.

• Répéter :

- •Prendre un exemple (x, c) dans S
- •Calculer la sortie o du réseau pour l ’entrée x
- •Pout i de 1 à n :
  - wi = wi + ε * (c – o) * xi * σ’(x.w)
- •Fin Pour

•Fin Répéter

PERCEPTRON : REGLE DELTA

Apprentissage : algorithme de Widrow-Hoff (adaline / règle delta)

La fonction de transfert est linéaire (purelin) : $\sigma(y) = y$

Donc : $\sigma'(y) = 1$

- Initialiser aléatoirement les coefficients wi.
- Répéter :
  - Prendre un exemple (x, c) dans S
  - Calculer la sortie o du réseau pour l 'entrée x
  - Pout i de 1 à n :
    - $wi = wi + \varepsilon * (c - o) * xi$
  - Fin Pour
- Fin Répéter

PERCEPTRONS / REGLES DELTA

Remarques

• ε : pas d'apprentissage

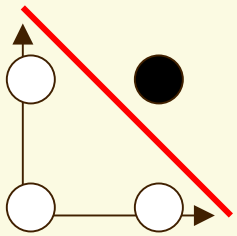
• Règles « delta » et « delta généralisée » : les algorithmes convergent vers la solution des moindres carrés.

• La règle « delta généralisée », par la non-linéarité de la fonction de transfert σ, permet de minimiser l 'importance d 'un élément étranger (erreur de mesure, bruit trop important, …).

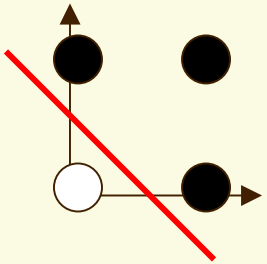
PERCEPTRONS / REGLES DELTA

Remarques

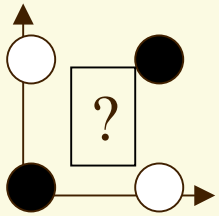
• Perceptrons = classificateurs linéaires (ensembles linéairement séparables).



AND



OR



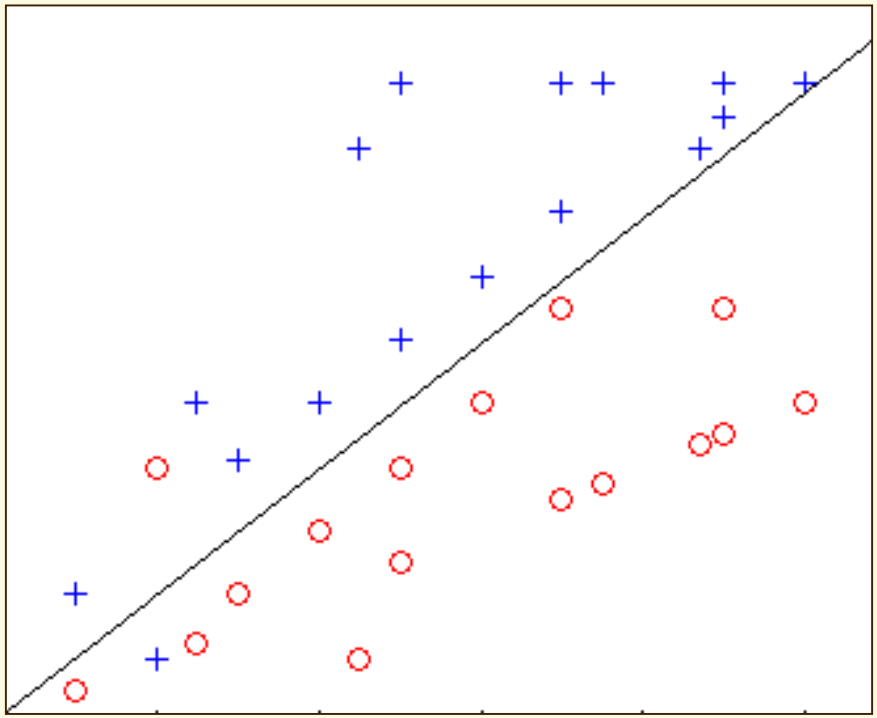
XOR

• Pour apprendre le OU Exclusif (XOR), on utilise un Perceptron Multi-Couches.

EXERCICE : perceptron

Classificateur linéaire :

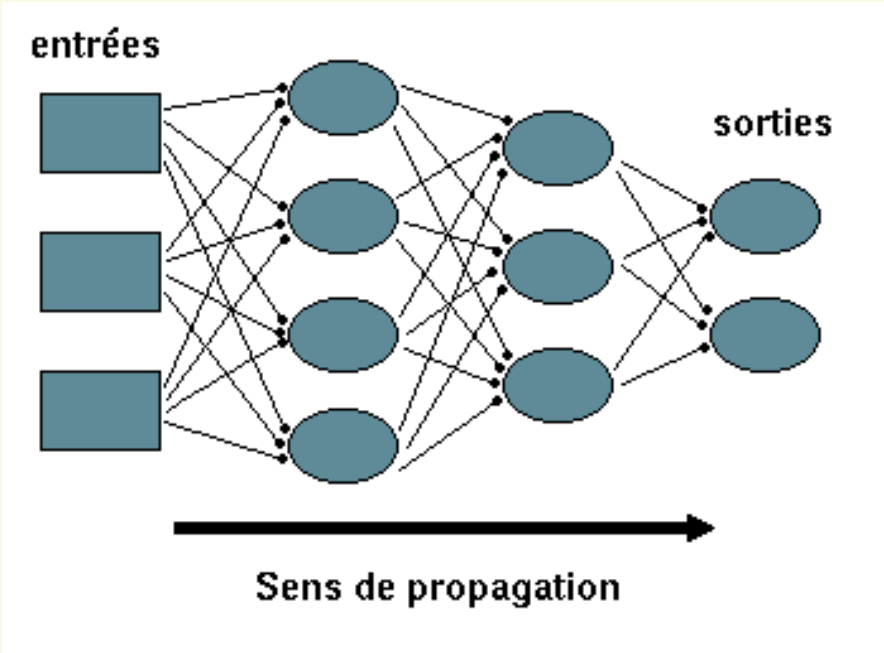
Trouver la droite qui sépare deux groupes de points.



PERCEPTRON MULTI-COUCHES

Les différentes couches :

- Cellules réparties dans q couches : C0, C1, …, Cq
- C0 : couche d ’entrée (la rétine) ⇒ les variables d ’entrée
- Cq : couche de sortie
- C1, …, Cq-1 : les couches cachées



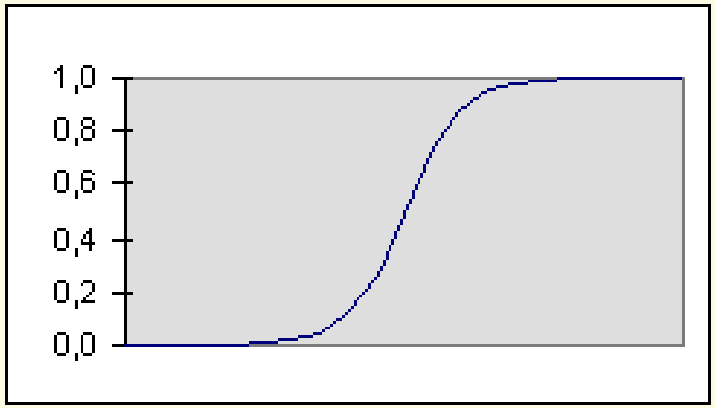
PERCEPTRON MULTI-COUCHES

Fonction de transfert :

• fonction sigmoïde logsig :

$$\sigma_k(x) = e^{kx}/(e^{kx}+1) = 1/(1+e^{-kx})$$

• k = 1 : $\sigma(x) = e^{x}/(e^{x}+1) = 1/(1+e^{-x})$



Dérivée :

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

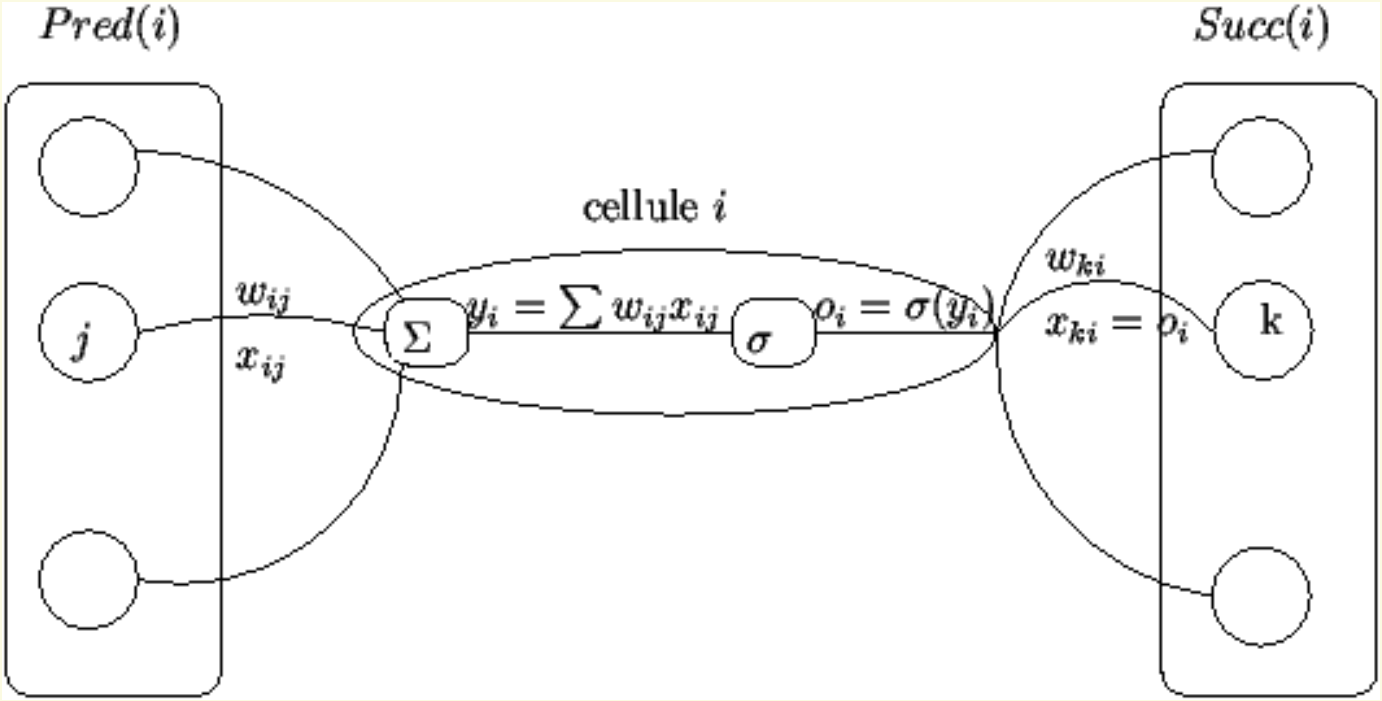
PERCEPTRON MULTI-COUCHES

Notations :

- n cellules
- Cellules désignées par un indice i, $0 \leq i < n$
- p cellules de sortie
- k indice d 'une cellule de sortie
- ck : sortie attendue pour la cellule de sortie k avec l 'entrée x
- ok : sortie calculée pour la cellule de sortie k avec l 'entrée x
- xij : entrée associée au lien entre cellule I vers cellule j
- wij : coefficient synaptique associé au lien entre cellule i vers cellule j
- Succ(i) : ensemble des cellules qui prennent comme entrée la sortie de la cellule i.
- Pred(i) : ensemble des cellules dont la sortie est une entrée de la cellule i.
- yi : entrée totale de la cellule i : $yi = \Sigma(j \in Pred(i)) (wij * xij)$
- oi : sortie de la cellule I : $oi = \sigma(yi)$

PERCEPTRON MULTI-COUCHES

Notations :



PERCEPTRON MULTI-COUCHES

Algorithme de rétropropagation du gradient :

- Initialiser aléatoirement les coefficients wij dans [-0.5 ; 0.5]
- Répéter
  - Prendre un exemple (x, c) de S
  - Calculer la sortie o
  - Pour toute cellule de sortie i
    - $\delta i = \sigma'(yi) * (ci - oi) = oi * (1 - oi) * (ci - oi)$
  - Fin Pour
  - Pour chaque couche de q – 1 à 1
    - Pour chaque cellule i de la couche courante
      - $\delta i = \sigma'(yi) * [\Sigma(k \in Succ(i)) (\delta k * wki)]$
        $= oi * (1 - oi) * [\Sigma(k \in Succ(i)) (\delta k * wki)]$
    - Fin Pour
  - Fin Pour
  - Pour tout poids wij
    - $wij = wij + \varepsilon * \delta i * xij$
  - Fin Pour
- Fin Répéter

PERCEPTRON MULTI-COUCHES

Ajout d 'un paramètre inertiel (momentum) β : éviter les oscilations

- Initialiser aléatoirement les coefficients wi dans [-0.5 ; 0.5]
- Répéter
  - Prendre un exemple (x, c) de S
  - Calculer la sortie o
  - Pour toute cellule de sortie i
    - $\delta i = \sigma'(yi) * (ci - oi) = oi * (1 - oi) * (ci - oi)$
  - Fin Pour
  - Pour chaque couche de q – 1 à 1
    - Pour chaque cellule i de la couche courante
      - $\delta i = \sigma'(yi) * [\Sigma(k \in Succ(i)) (\delta k * wki)]$
        $= oi * (1 - oi) * [\Sigma(k \in Succ(i)) (\delta k * wki)]$
    - Fin Pour
  - Fin Pour
  - Pour tout poids wij à litération k
    - $wij(k) = wij(k-1) + \varepsilon * \delta i * xij + \beta * (wij(k - 1) - wij(k - 2))$
  - Fin Pour
- Fin Répéter

PERCEPTRON MULTI-COUCHES

Remarques :

Perceptron Multi-Couches

= généralisation du perceptron et de la règle delta.

PERCEPTRON MULTI-COUCHES

Estimer la qualité du réseau :

• Présenter des exemples (pour lesquels on connaît la sortie souhaitée) qui ne font pas partie de la base d'apprentissage et comparer la sortie souhaitée avec la sortie calculée par le réseau.

• Attention au sur-apprentissage : la base d'apprentissage est parfaitement apprise mais la capacité de généralisation est faible.

• Dans la pratique : garder une partie de la base d'apprentissage pour évaluer la qualité du réseau

- 80 % de la base pour l'apprentissage
- 20 % restant de la base pour l'évaluation de la qualité

PERCEPTRON MULTI-COUCHES

Normaliser les données d'entrée (pour chaque neurone d'entrée i) :

- 1-ère méthode : grâce à la moyenne et à l'écart-type
  - Données continues

$$\bar{x}_i = \frac{1}{N}\sum_{k=1}^{N} x_i^k$$

$$\sigma_i^2 = \frac{1}{N-1}\sum_{k=1}^{N} (x_i^k - \bar{x}_i)^2$$

$$\tilde{x}_i^k = \frac{x_i^k - \bar{x}_i}{\sigma_i}$$

- 2-ème méthode : en se ramenant dans un intervalle du type [0 ; 1]
  - Données continues
  - Données discrètes

PERCEPTRON MULTI-COUCHES

Remarque :

Le nombre de couches cachées et le nombre de neurones par couche ont une influence sur la qualité de l'apprentissage.

PERCEPTRON MULTI-COUCHES

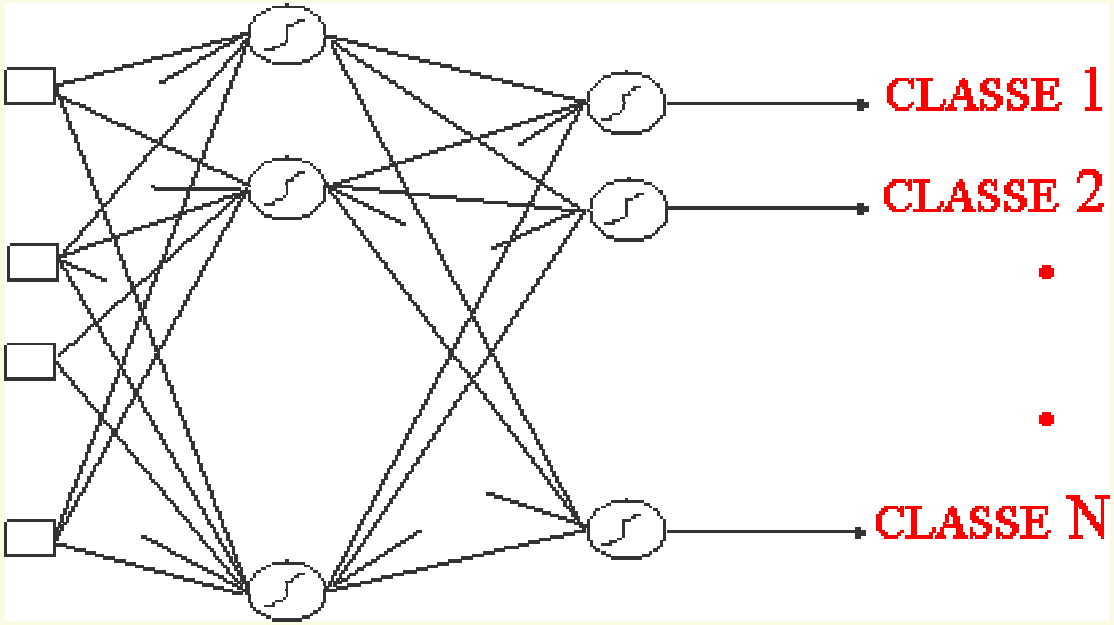
Classification / Discrimination :

- Réseaux de neurones à 2 ou 3 couches (1 ou 2 couches cachées)
- Fonctions de transfert « logsig »
- Sorties dans l'intervalle [0 ; 1]

PERCEPTRON MULTI-COUCHES

Classification / Discrimination :

- 1-ère modélisation :
  - ➢chaque neurone de sortie indique la probabilité d'appartenance à la classe correspondante
  - ➢ l'exemple fourni en entrée appartient à la classe pour laquelle la probabilité d'appartenance est la plus forte



PERCEPTRON MULTI-COUCHES

Classification / Discrimination :

- Inconvénients :
  - ➢ apprentissage très long (très nombreuses connexions)
  - ➢ une nouvelle classe : il faut tout réapprendre !

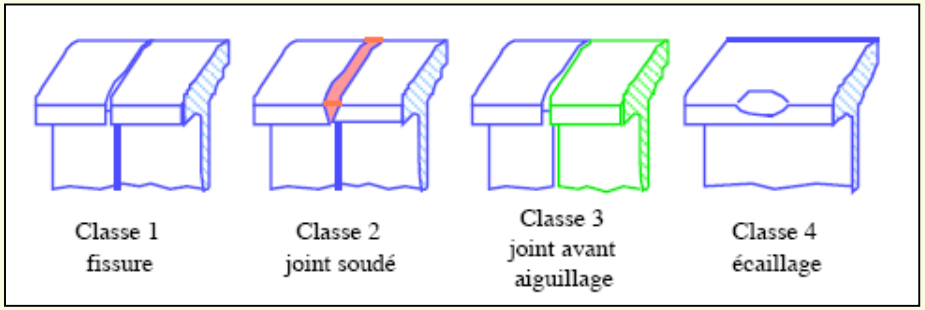
- 2-ème modélisation : un réseau par classe
  - ➢ pour chaque réseau, une sortie indique si l'exemple soumis appartient à la classe correspondante

PERCEPTRON MULTI-COUCHES

Classification / Discrimination :

Applications :

- Reconnaissance de formes
  - ➢codes postaux, signatures, visages,
  - ➢ défauts de rails



- Traitement du signal : reconnaissance de la parole
- Informatique décisionnelle (à quelle classe de consommateurs appartient un individu ?)
- …

PERCEPTRON MULTI-COUCHES

Régression (Approximation de fonction) :

On donne x en entrée.

Le réseau calcule f(x) en sortie

- Réseaux de neurones à 2 couches (1 couche cachée)
- Fonctions de transfert « logsig » pour la couche cachée
- Fonctions de transfert « purelin » pour la couche de sortie
- Perceptron Multi-Couches = ***approximateur universel***.

  $\Rightarrow$ Permet d'approximer n'importe quelle fonction, à condition que le nombre de neurones dans les couches cachées soit suffisant !

PERCEPTRON MULTI-COUCHES

Prévision :

On donne K valeurs successives de f en entrée.
Le réseau calcule la valeur suivante en sortie

Même type de réseau que pour la régression.

PERCEPTRON MULTI-COUCHES

Régression / Prévision :

Applications :

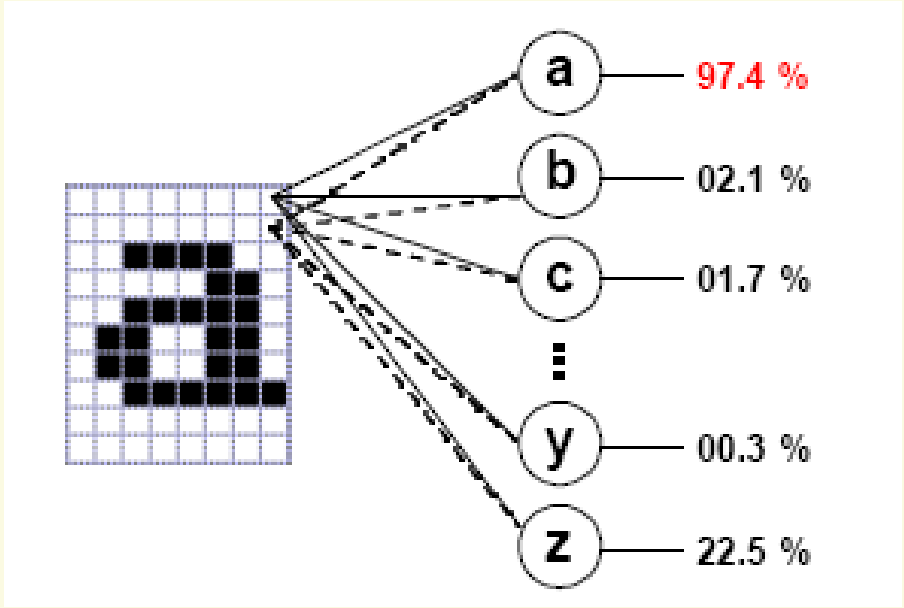
- approximation de fonctions « compliquées »
- régression à partir de données expérimentales (relevés ou mesures expérimentales)
- prévisions météorologiques
- prévisions financières
- …

EXERCICES : PMC

Classification :

Reconnaissance de formes :

- les lettres de l ’alphabet
- les chiffres de 0 à 9

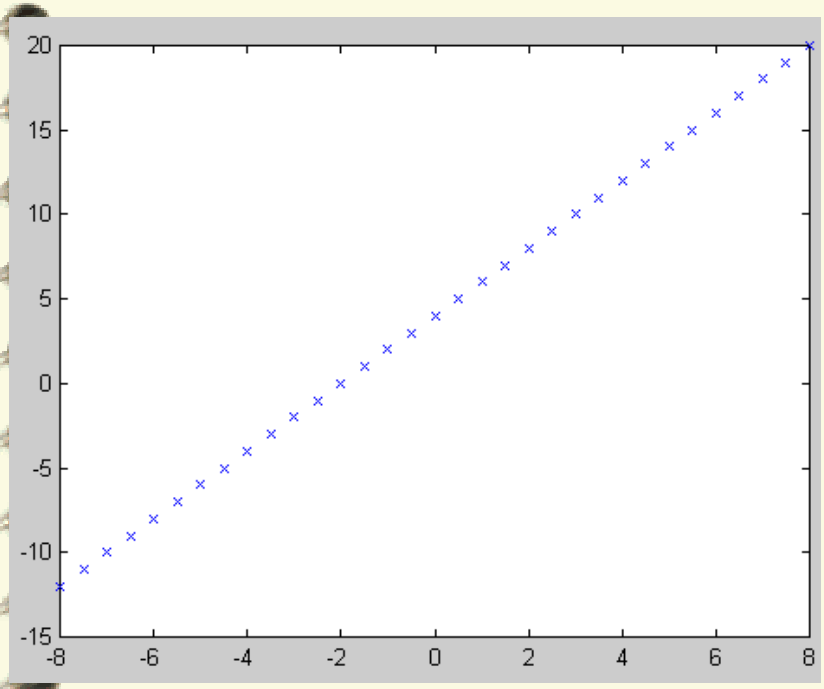


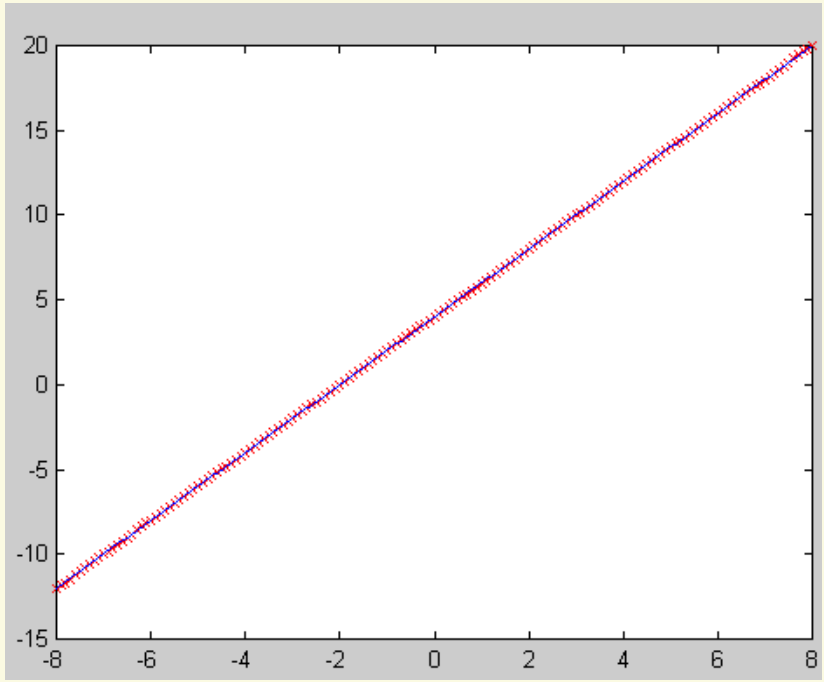
EXERCICES : PMC

Régression :

Approximation de fonctions :

- y = 2*x + 4 sur [-8 ; 8]



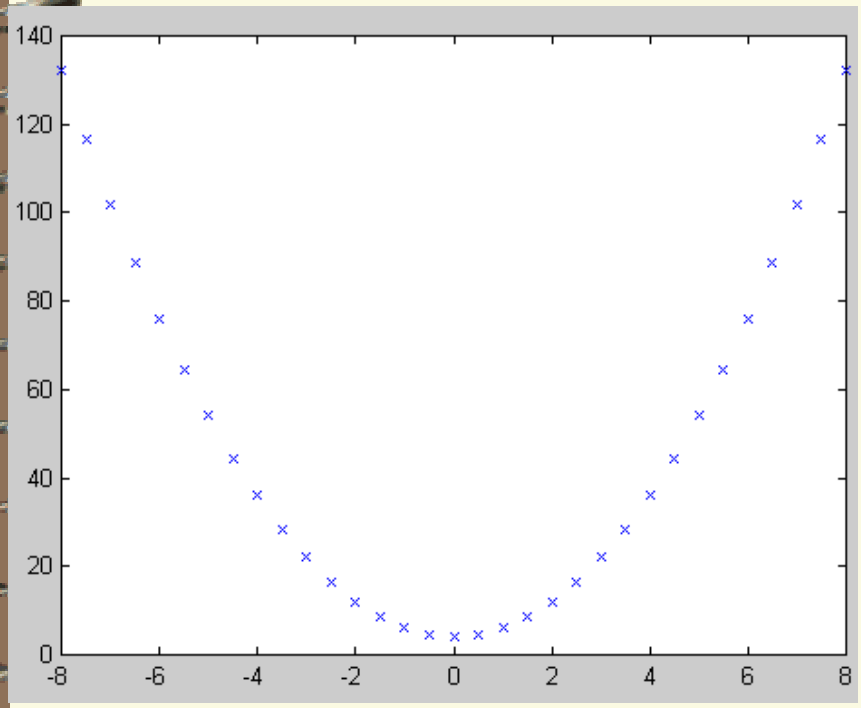


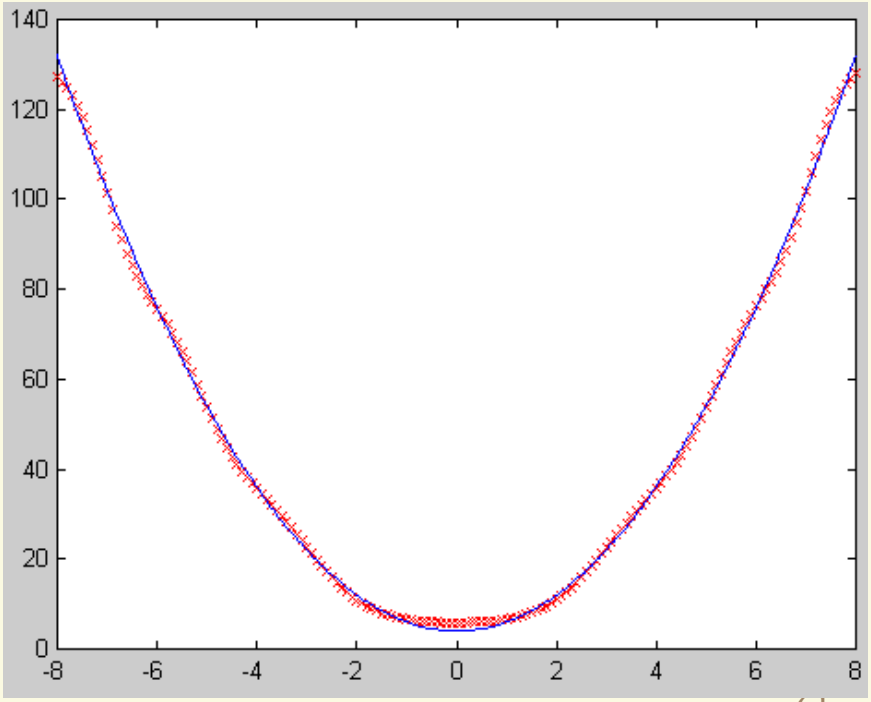
EXERCICES : PMC

Régression :

Approximation de fonctions :

- y = 2*x^2 + 4 sur [-8 ; 8]



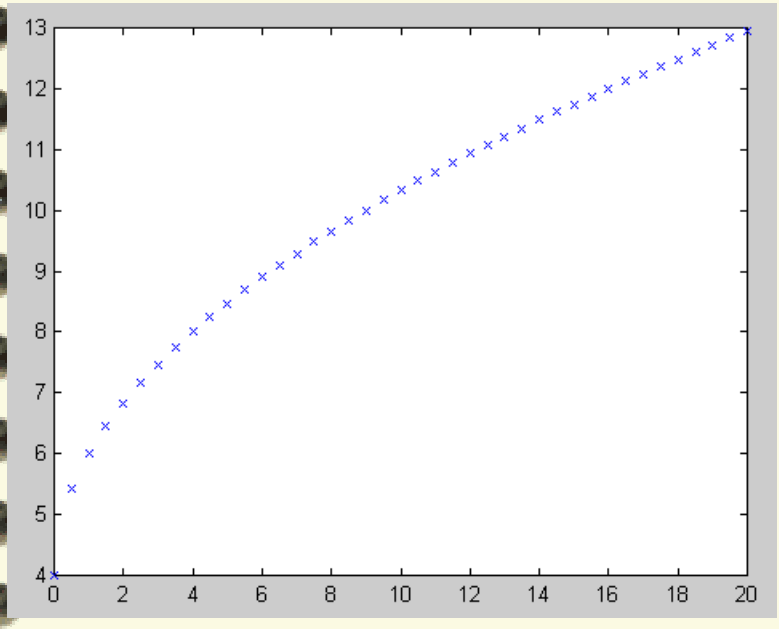


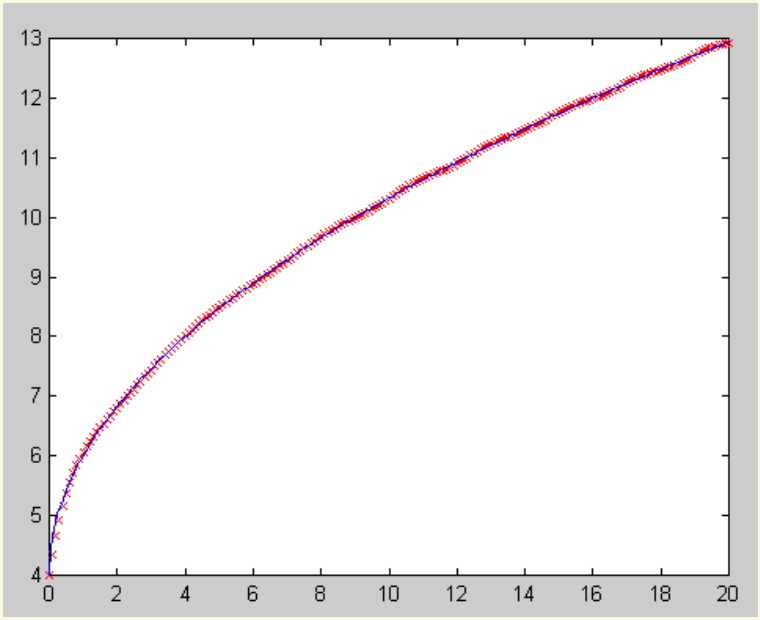
EXERCICES : PMC

Régression :

Approximation de fonctions :

- $y = 2*\sqrt{x} + 4$ sur [0 ; 20]



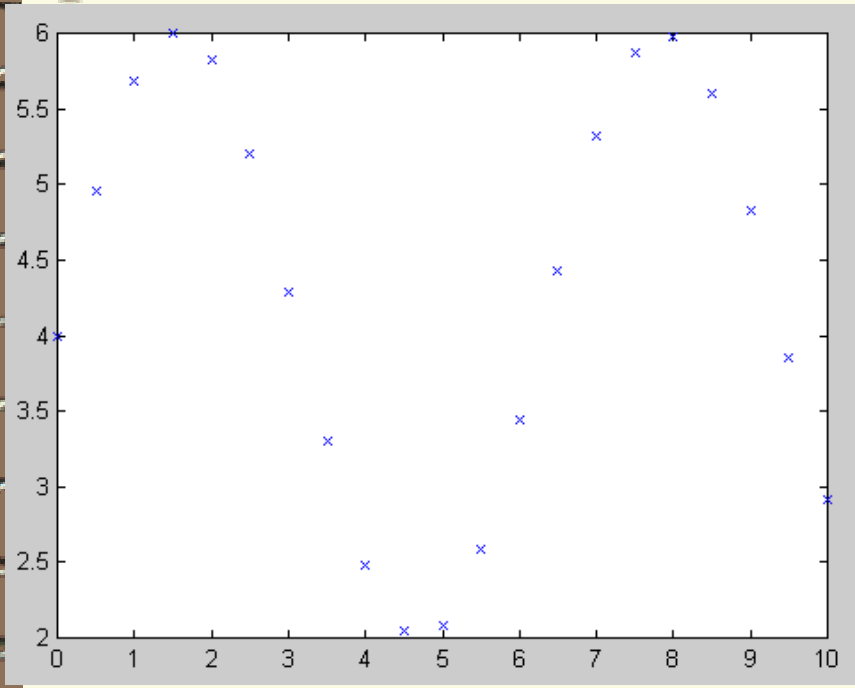


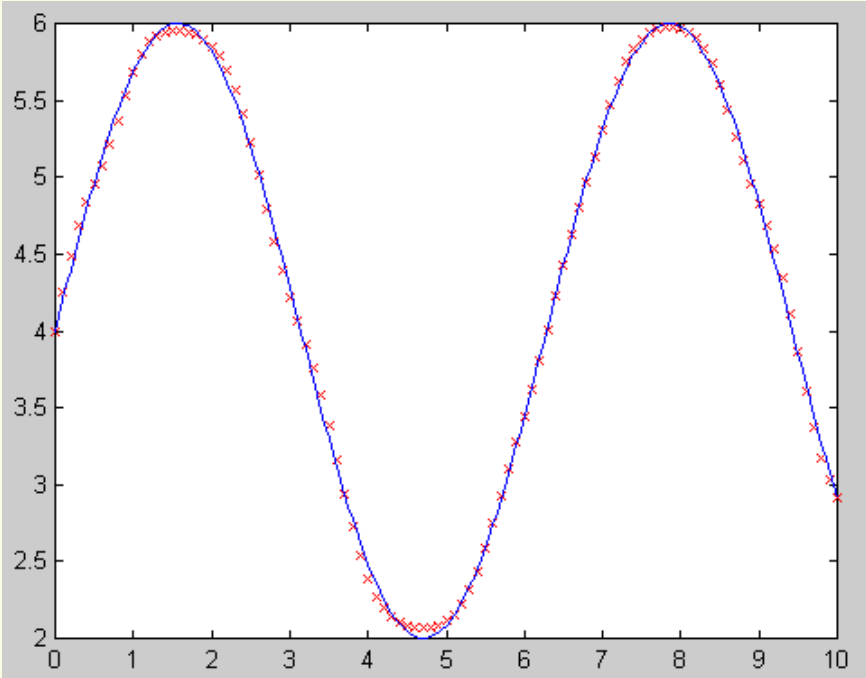
EXERCICES : PMC

Régression :

Approximation de fonctions :

- y = 2*sin(x) + 4 sur [0 ; 10]



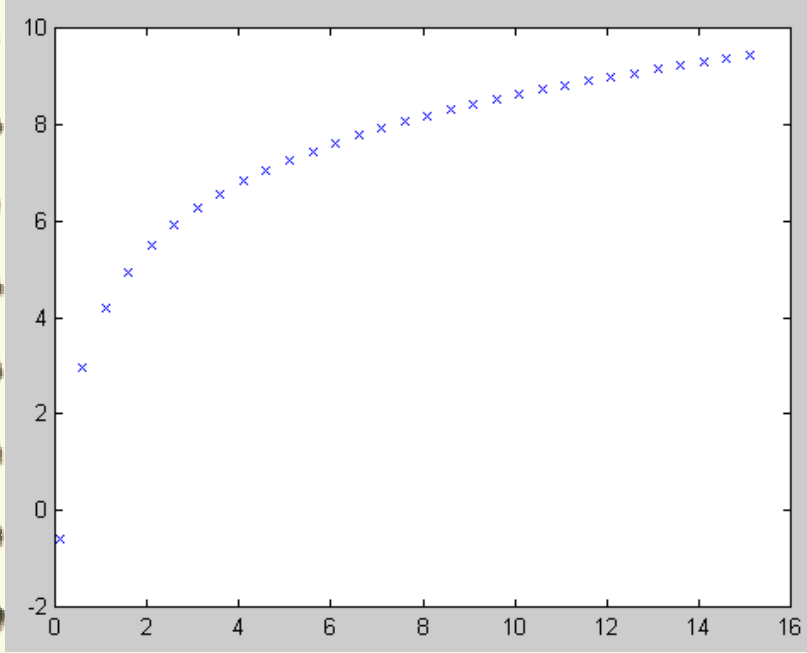


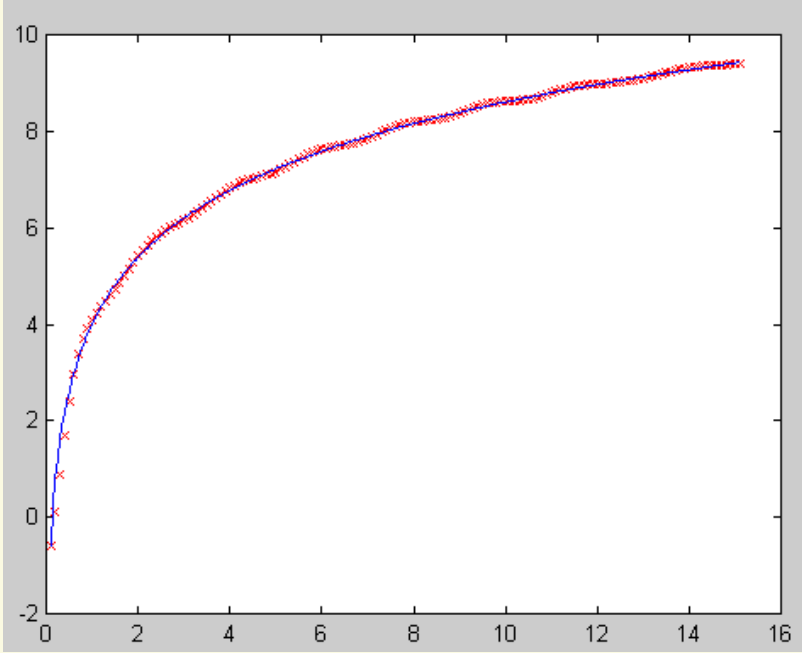
EXERCICES : PMC

Régression :

Approximation de fonctions :

- y = 2*ln(x) + 4 sur [0.1 ; 15.1]



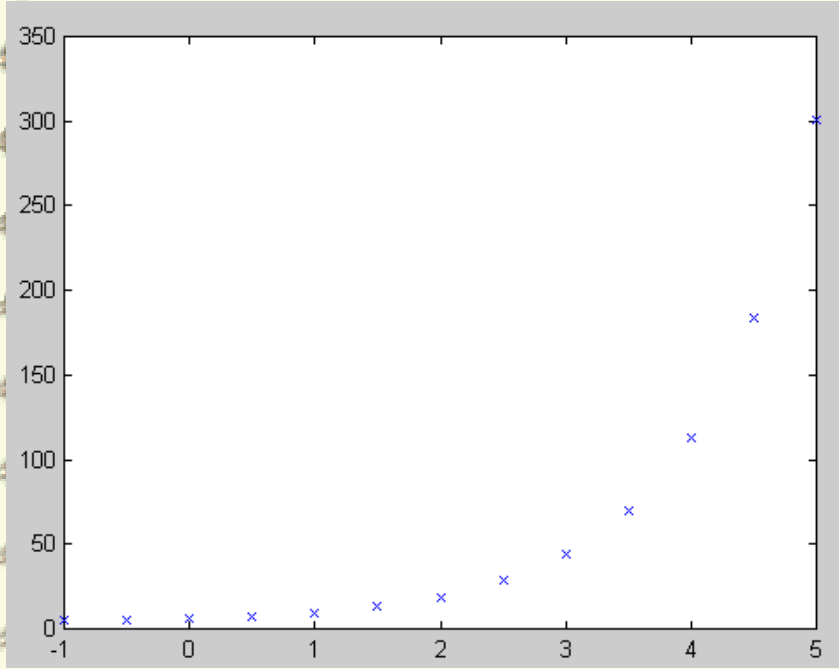


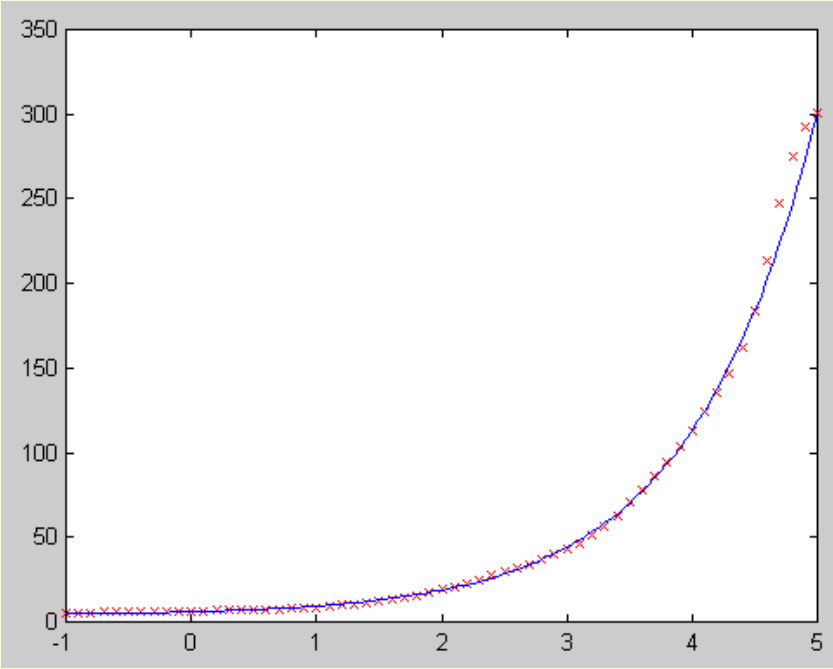
EXERCICES : PMC

Régression :

Approximation de fonctions :

- y = 2*exp(x) + 4 sur [-1 ; 5]





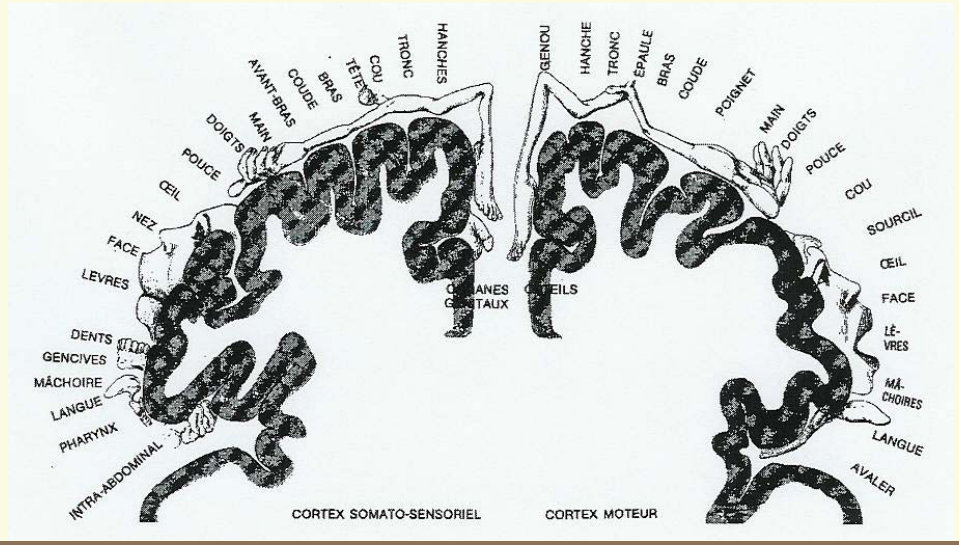
CARTE DE KOHONEN

• Réseaux de Neurones à ***Compétition*** : un seul neurone de sortie est activé pour une entrée donnée.

• Mise en correspondance de l'espace d'entrée avec l'espace du réseau.

⇒ ***Auto-organisation***

• ***Origine biologique*** : une zone du cortex correspond à une zone des organes sensoriels et moteurs



CARTE DE KOHONEN

• Applications :

⇒ robotique : pilotage de robots

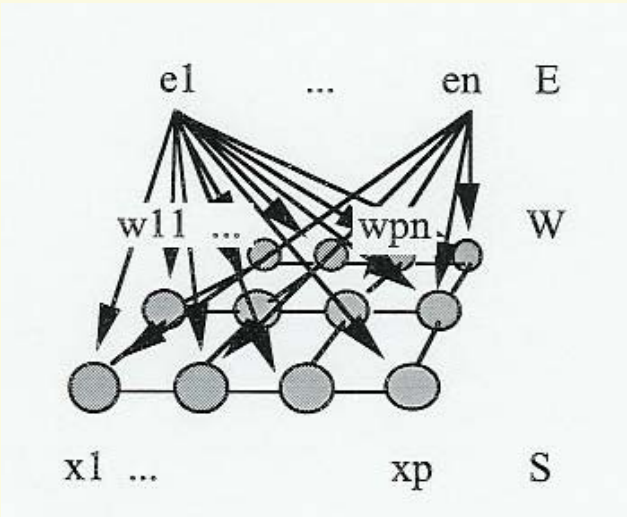
⇒ compression de données

⇒ statistiques : méthode semblable à l'Analyse en Composantes Principales (ACP)

CARTE DE KOHONEN

Principes (en 2D) :

- n cellules d'entrée e = (e1, …, en)
- une ***carte*** : réseau de m neurones de sortie x1, …, xm
- connexions latérales (coefficients fixes) entre les neurones de sortie : un neurone est connecté à ses 4 plus proches voisins
- connexions de coefficient wij entre une cellule d'entrée ei et un neurone de sortie xj



CARTE DE KOHONEN

Principes :

Pour une entrée, un seul neurone sur la carte est sélectionné (valeur 1).

On encourage le vainqueur : « the winner takes all ».

Ce neurone correspond le plus possible à l'entrée : ***minimisation d'une distance***.

Remarques :

En pratique, on normalise les entrées.

CARTE DE KOHONEN

Algorithme d'apprentissage :

- Initialiser aléatoirement les coefficients wij
- Répéter
  - Prendre une entrée e = (e1, …, ei, …, en)
  - Calculer la distance dj de chaque neurone xj par rapport à e

$$d_j = \sqrt{\sum_{i=1}^{n} (e_i - w_{ij})^2}$$

  - Sélectionner le neurone xk le plus proche de e : dk = Min(dj)
  - Modifier les coefficients pour le neurone sélectionné et ses plus proches voisins (4 pour une carte 2D) :
    - Pour tout i :
      - wik = wik + μ ∗ (ej – wik)
      - wil = wil + β ∗ (ej – wil) où xl est un voisin de xk
    - Fin Pour
- Fin Répéter

CARTE DE KOHONEN

Algorithme d'utilisation :

La sortie sk du neurone xk, pour une entrée e, est donnée par :

$sk = 1$ si $dk = Min(di)$

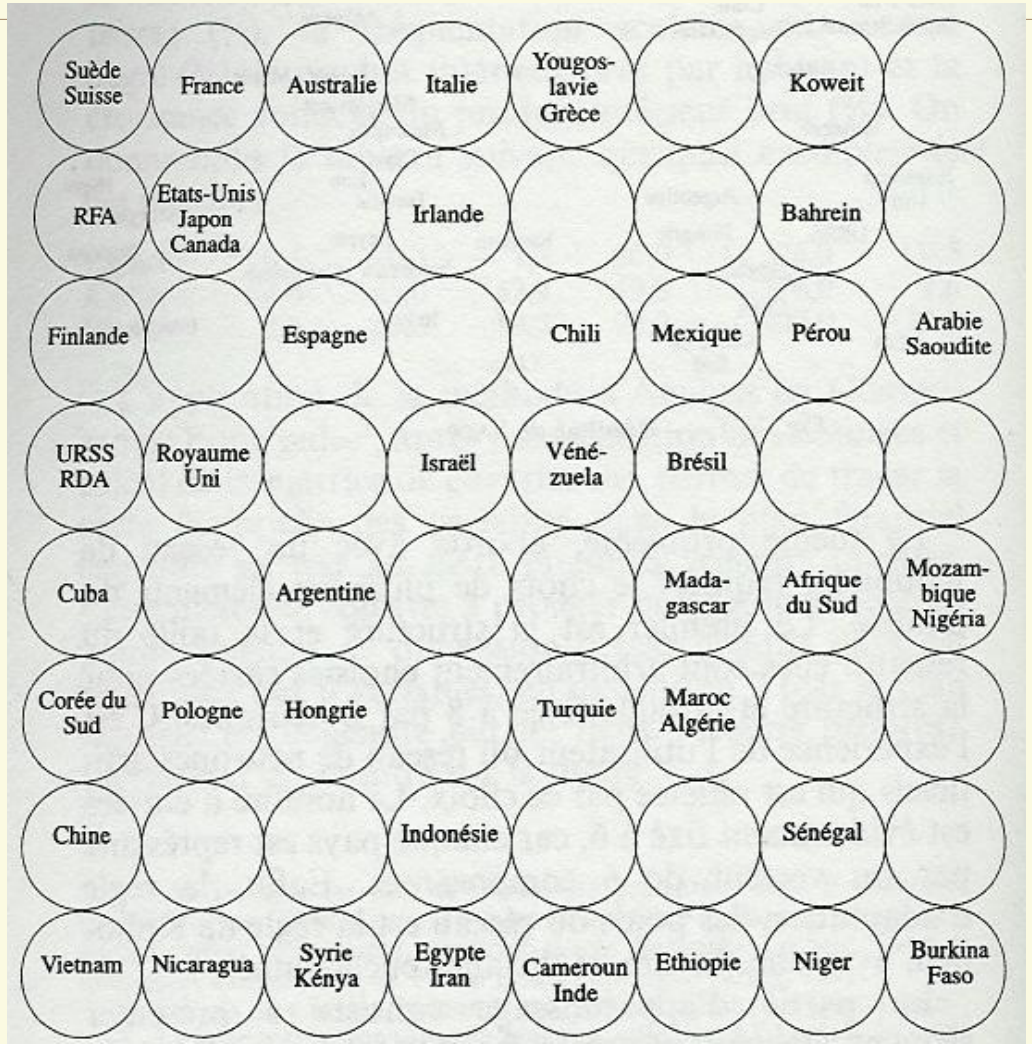
$sk = 0$ si $i \neq k$

CARTE DE KOHONEN

Application :
Analyse de données (cf. analyse en composantes principales)

Exemple :
Analyse de données socio-économiques (PIB, croissance du PIB, mortalité infantile, taux d'illettrisme, …) de 52 pays
1 neurone de la carte = 1 groupe de pays (même situation socio-économique)
Neurones voisins = groupes de pays proches (du point de vue de la situation socio-économique)

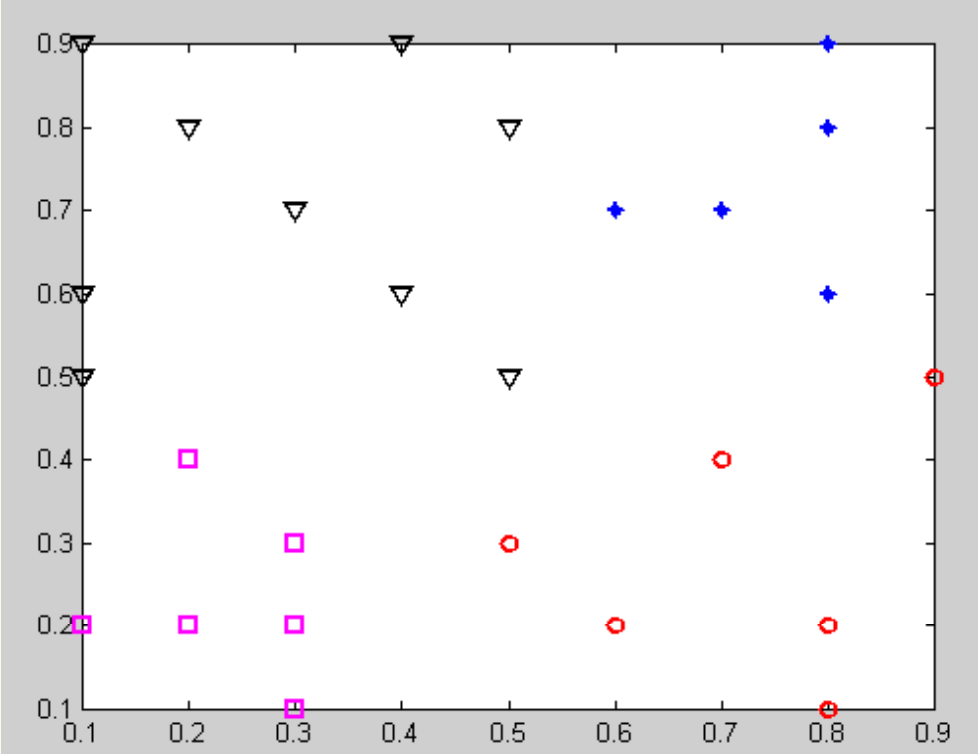
CARTE DE KOHONEN



EXERCICE : carte de Kohonen

Analyse de données / clustering:

4 classes dans un nuage de points



RESEAUX RECURRENTS / BOUCLES

1) SYSTEMES DYNAMQUES A TEMPS DISCRET

Équations d'état :

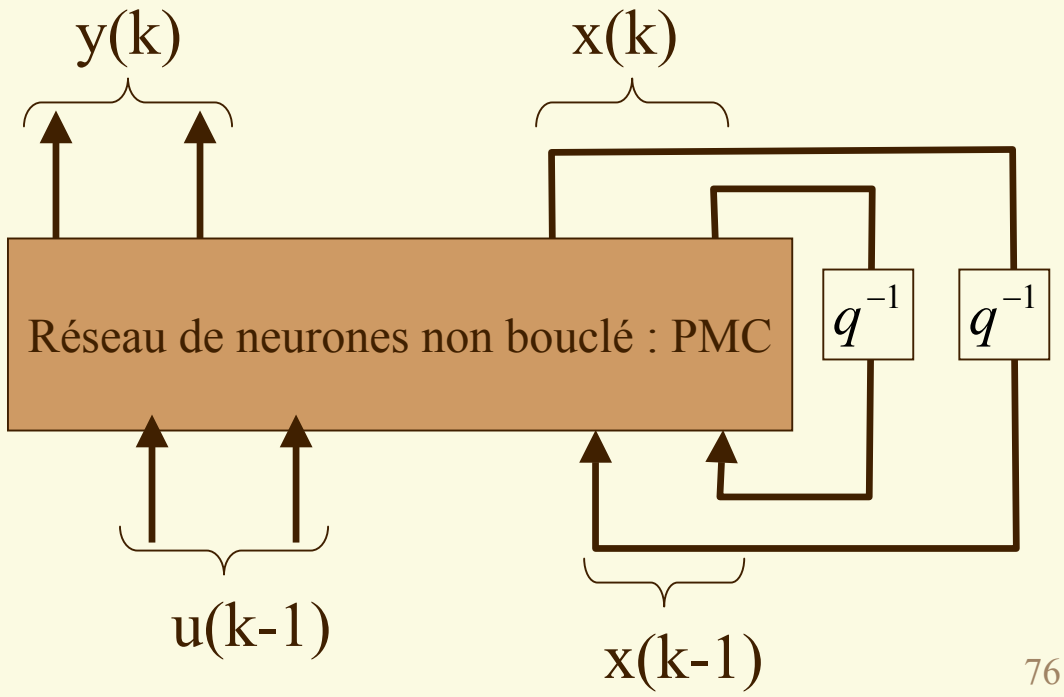
- $\begin{cases} x(k) = \varphi[(x(k-1) , u(k-1)] \\ y(k) = \phi[x(k)] \end{cases}$
- Avec u(k) : vecteur des entrées à l'instant k*t
  y(k) : vecteur des sorties à l'instant k*t
  x(k) : vecteur des variables d'état k*t

RESEAUX RECURRENTS / BOUCLES

1) SYSTEMES DYNAMQUES A TEMPS DISCRET

Représentation sous forme de réseau de neurones récurrent ou bouclé :

q^{-1} Symbole représentant un retard d'une unité de temps



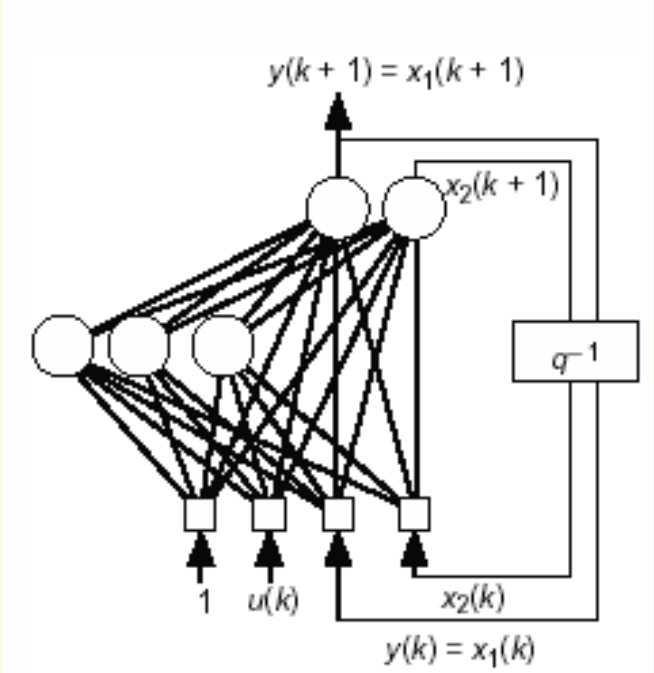
RESEAUX RECURRENTS / BOUCLES

1) SYSTEMES DYNAMQUES A TEMPS DISCRET

Applications :

- Automatique / Robotique : contrôle commande, asservissement

Commande d'un actionneur hydraulique d'un bras de robot

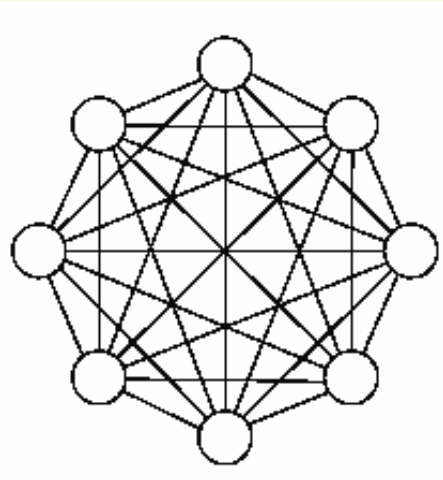


- Séries temporelles

RESEAUX RECURRENTS / BOUCLES

2) RESEAUX DE HOPFIELD

- Réseaux totalement connectés



- Mémoires auto-associatives
  - Associer quelque chose à une suite de données
  - Exemple : on fournit une partie du visage et le on retrouve le visage entier

RESEAUX RECURRENTS / BOUCLES

2) RESEAUX DE HOPFIELD

- Chaque neurone a 2 états possibles : -1 ou +1
- L'activation du neurone i est :
  - $\begin{cases} ai = 1 \text{ si } wi1*y1 + \ldots + wij*yj + \ldots + win*yn > 0 \\ ai = -1 \text{ sinon} \end{cases}$

    La fonction de transfert est la fonction Signe
  - Avec :

    yi état du neurone i

    wij coefficient synaptique entre les neurones i et j

    wij = wji (matrice W symétrique avec diagonale nulle)

RESEAUX RECURRENTS / BOUCLES

2) RESEAUX DE HOPFIELD

- Modèle dynamique :

L'état du neurone i à l'instant t+1 est donné par :

$y_i(t) = a_i(t)$

$= \text{Signe}(w_{i1} * y_1(t) + \ldots + w_{ij} * y_j(t) + \ldots + w_{in} * y_n(t))$

RESEAUX RECURRENTS / BOUCLES

2) RESEAUX DE HOPFIELD

- Apprentissage :
  - On apprend avec une base de :
    - couples {xk ; f(xk)}
    - vecteurs yk
  - Algorithme d'apprentissage : analogue à la loi de Hebb :

$$wij = \frac{1}{N} \sum_{k=1}^{P} y_i^k \times y_j^k$$

N : nombre de neurones (nombre d'entrées)
P : nombre de vecteurs yk

  - Convergence vers des états stables (points fixes) et les bassins d'attraction associés
  - Bassin d'attraction du vecteur y = ensemble des vecteurs initiaux qui conduisent au vecteur y
  - Attention : états stables parasites !

RESEAUX RECURRENTS / BOUCLES

2) RESEAUX DE HOPFIELD

- Utilisation :

  Après présentation des entrées, réinjecter les sorties jusqu'à ce que l'état interne du réseau devienne stable

  - ❑ On fournit x en entrée et le réseau donne f(x)
  - ❑ On fournit y incomplet ou bruité et le réseau reconstitue y

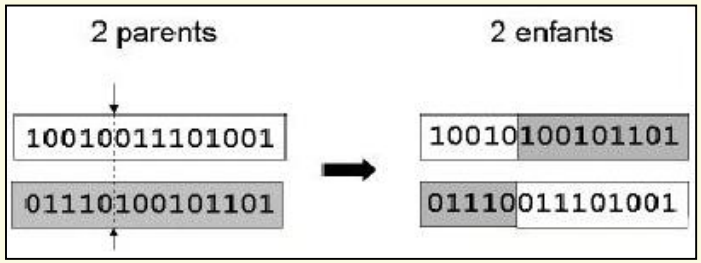
Optimisation : AG (rappels)

Principe des algorithmes génétiques :

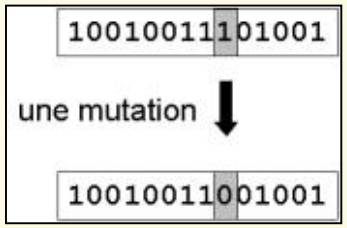
```
Génération aléatoire d'une population initiale
nb_iter = 0
Evaluation de l'adaptation de chaque individu de la population
Tant que (solution non satisfaisante) et (nb_generations < nb_ generations _max)
Faire
        Incrémenter le nombre d'itérations nb_ generations
        Sélectionner les parents P1 et P2 de façon aléatoire
        Appliquer l'opérateur de croisement aux parents P1 et P2
        Faire subir des mutations aléatoires à la population
        Evaluer l'adaptation de chaque individu
        Sélectionner les plus adaptés
Fin Tant que
```

Opérateurs

- croisement : ***exploiter***



- mutation : ***explorer***



RESEAUX EVOLUTIONNAIRES

Exemple 1 :

Problématique :

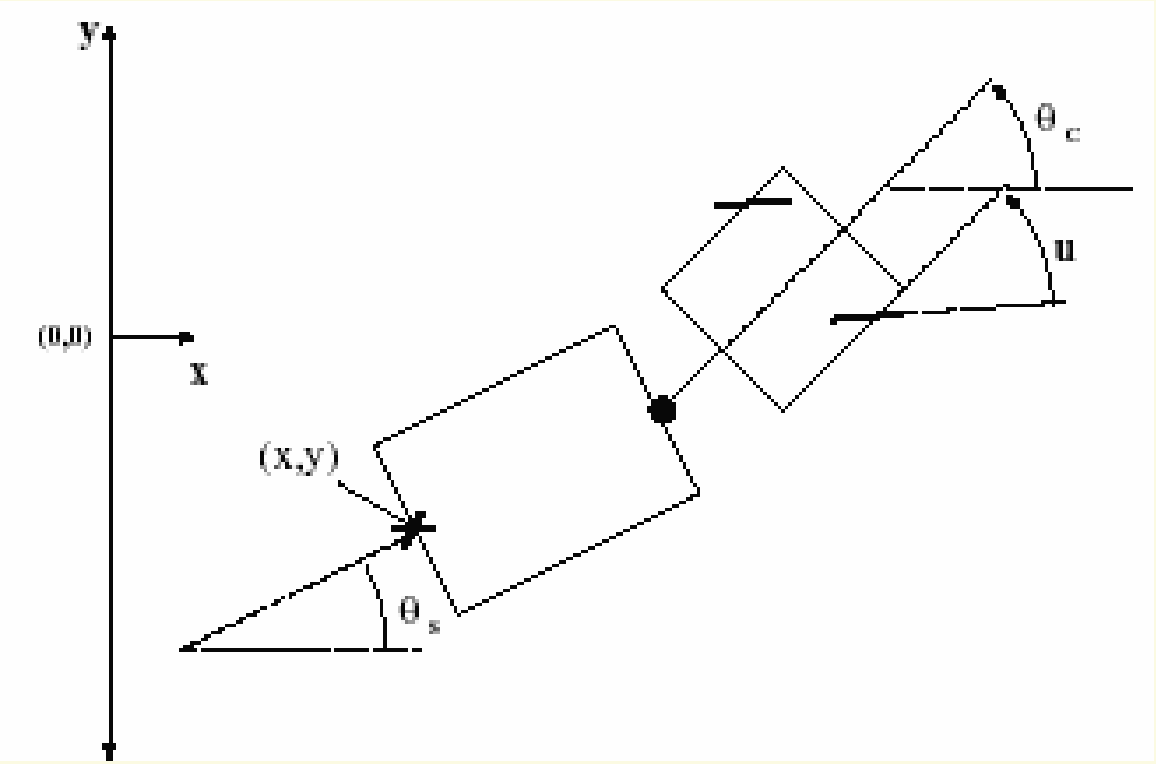
- Garer un semi-remorque en marche arrière

Solution :

- Pilotage du véhicule par un réseau de neurones (PMC)
- Apprentissage par un algorithme évolutionnaire
  - Pas de base d'apprentissage
  - L'expérience permet d'apprendre : comportement émergent

RESEAUX EVOLUTIONNAIRES

Principe :



RESEAUX EVOLUTIONNAIRES

Variables :

x, y	coordinates of the center rear of the trailer
θ_S	angle of the trailer with x-axis
θ_C	angle of the cab with x-axis
u	steering angle of the front wheels relative to cab orientation

Paramètres :

r	$3m$	distance front wheel moves per time step.
L_S	$14m$	length of the trailer, from rear to pivot.
L_C	$6m$	length of the cab, from pivot to front axle.

Contraintes :

$x > 0$	the loading dock is at $x = 0$
$\lvert\theta_S - \theta_C\rvert \leq 90°$	the angle between the cab and trailer can't exceed 90°
$-70° \leq u \leq 70°$	limit of the steering of the front wheel

Équations de mouvement :

$$A = r * \cos(u)$$

$$B = A * \cos(\theta_C[t] - \theta_S[t])$$
$$x[t+1] = x[t] - B * \cos(\theta_S[t])$$
$$y[t+1] = y[t] - B * \sin(\theta_S[t])$$
$$\theta_S[t+1] = \theta_S[t] - \arcsin\left(\frac{A * \sin(\theta_C[t] - \theta_S[t])}{L_S}\right)$$
$$\theta_C[t+1] = \theta_C[t] + \arcsin\left(\frac{r * \sin(u)}{L_S + L_C}\right)$$

$\theta_C[t+1]$ is then adjusted to respect the constraint on $\theta_S - \theta_C$

RESEAUX EVOLUTIONNAIRES

PMC :

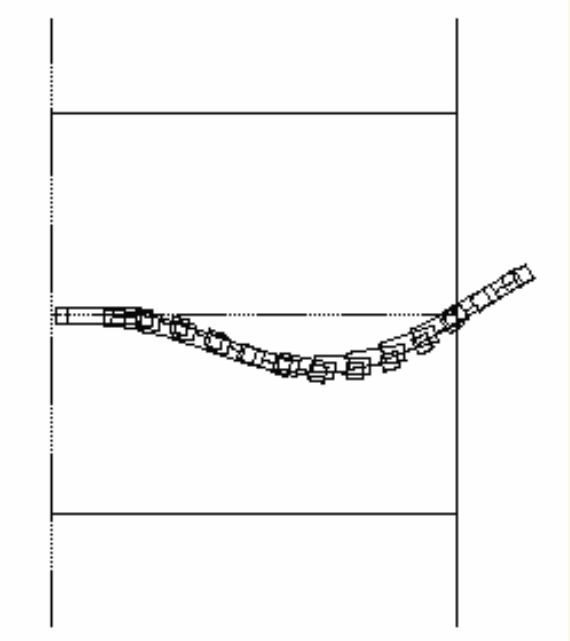
- 3 couches cachées
- Entrées : x, y, θs, θc
- Sortie : u

RESEAUX EVOLUTIONNAIRES

Principe :

- Position de départ : x0, y0, θs0, θc0
- 1 mouvement (itération) : la position suivante est donnée par la sortie du réseau u et par les équations de mouvement
- On autorise 300 mouvements (itérations) max.
- Le point de garage est donné par :

  Xg = 0 ; Yg = 0 ; θsg = 0



RESEAUX EVOLUTIONNAIRES

Apprentissage par algorithme évolutionnaire :

- un individu = 1 PMC = liste des coefficients synaptiques
- population de PMC
- opérateurs de croisement, mutation, sélection
- au bout de N générations : le PMC optimal pour une fonction d ’évaluation (***fitness function***) $\Leftrightarrow$ le PMC le plus adapté

RESEAUX EVOLUTIONNAIRES

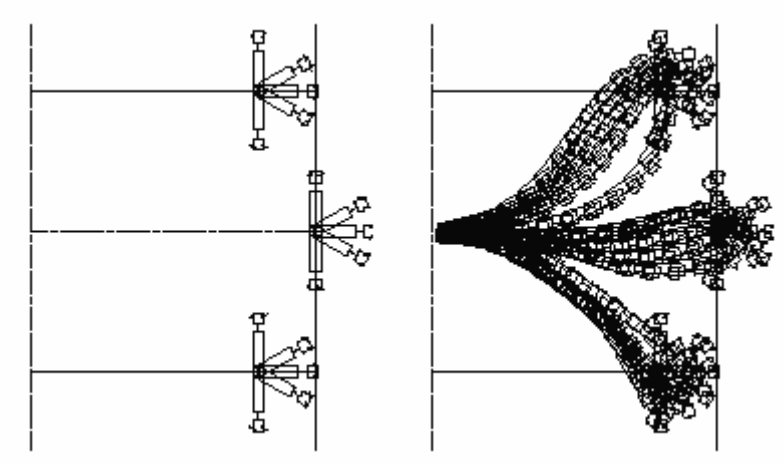
Fitness function :

- On veut minimiser
  - la distance au point de garage
  - le nombre de mouvements (itérations)
- fitness function = $1 / (\varepsilon + d^2 + \gamma * nb\_iter)$, où
  - d est la distance au point de garage
  - nb_iter est le nombre de mouvements (itérations) effectués
  - $\varepsilon = 0.1$ et $\gamma = 0.001$

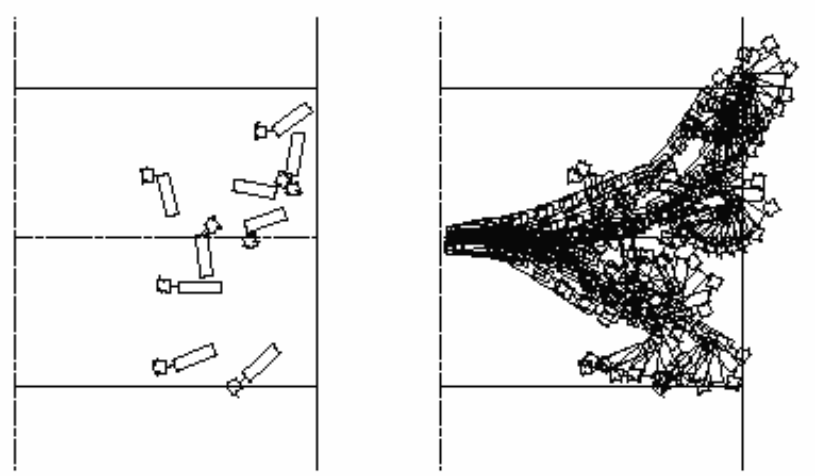
RESEAUX EVOLUTIONNAIRES

Résultats :

- Apprentissage :



- Tests (généralisation) :



RESEAUX EVOLUTIONNAIRES

Exemple 2 :

Problématique :

- Apprentissage du comportement suivant
  - Trouver le plus court-chemin jusqu'à un point-cible
  - Et éviter des obstacles

Solution :

- Pilotage du robot par un réseau de neurones (PMC)
- Apprentissage par un algorithme évolutionnaire
  - Pas de base d'apprentissage
  - L'expérience permet d'apprendre : comportement émergent

RESEAUX EVOLUTIONNAIRES

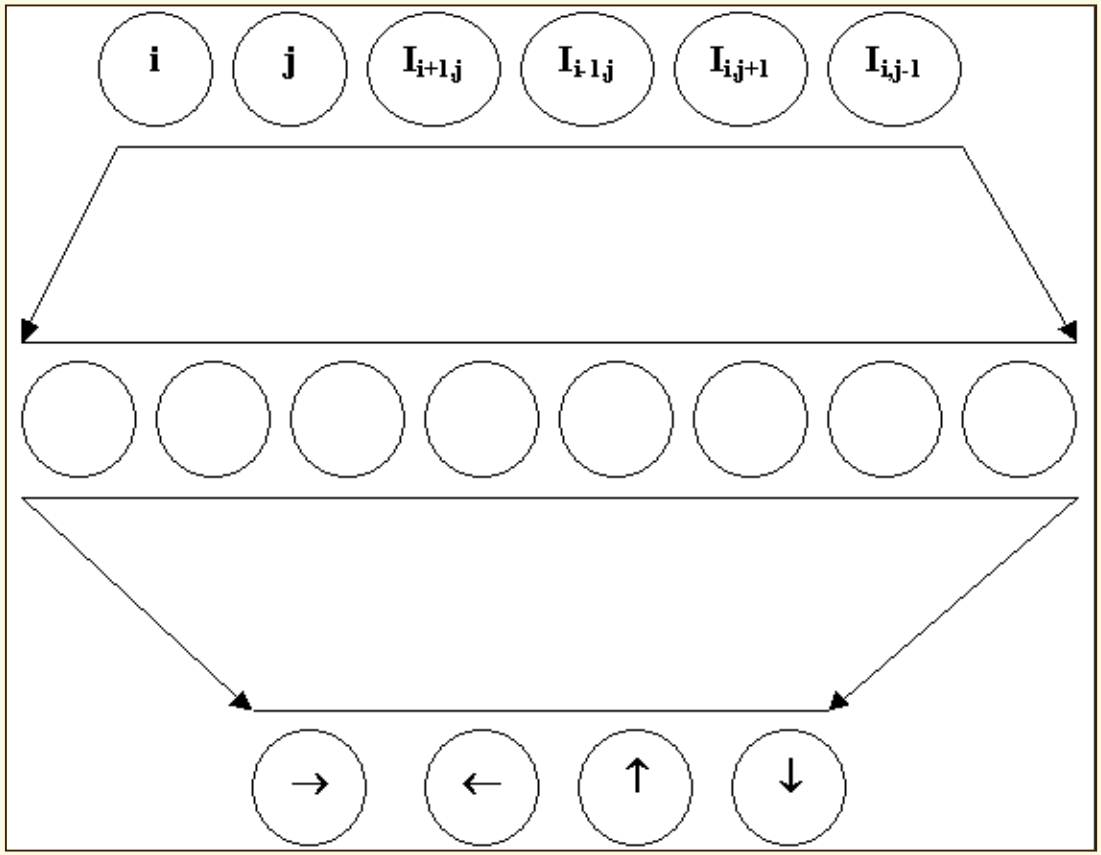
- Modélisation de l ’univers : quadrillage NxM

- Modélisation d ’un obstacle : indice de présence Iij d ’une menace sur une case (i ; j)
  - Iij = 0 si aucune menace et aucun obstacle
  - Iij = -1 si un obstacle
  - Iij = -0.5 si un obstacle sur les cases voisines

- PMC :
  - 6 entrées : coordonnées du point courant et indices de présence d ’un obstacle sur les cases voisines
  - 4 sorties : probabilité de déplacement vers le nord, le sud, l ’ouest ou l ’est. La sortie la plus élevée indique la direction à suivre

RESEAUX EVOLUTIONNAIRES

PMC :



RESEAUX EVOLUTIONNAIRES

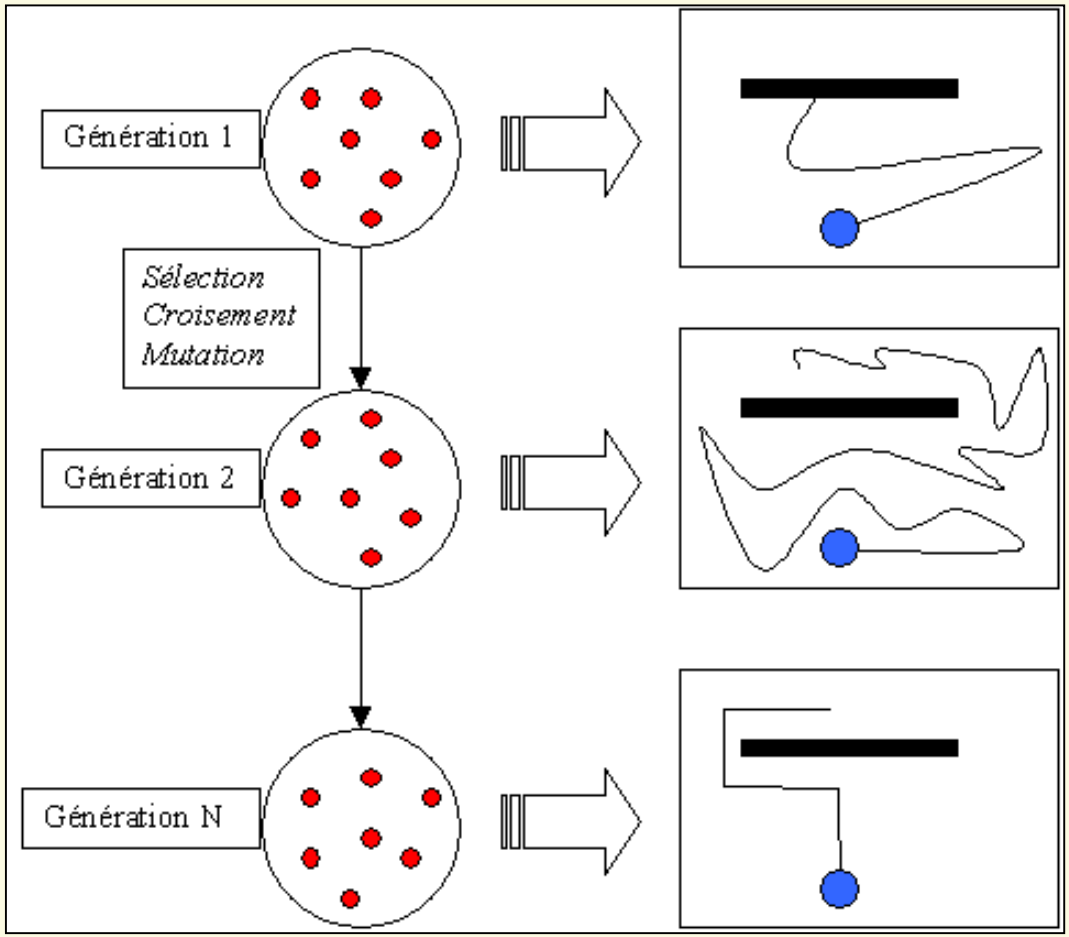
• Pas de base d ’apprentissage ! Pas de couples {*position de départ ; liste des déplacements successifs*}

• Apprentissage par des algorithmes génétiques

- ➢ un individu = 1 PMC = liste des coefficients synaptiques
- ➢ population de PMC
- ➢ opérateurs de croisement, mutation, sélection
- ➢ au bout de N générations : le PMC optimal pour une fonction d ’évaluation (***fitness function***) ⇔ le PMC le plus adapté

RESEAUX EVOLUTIONNAIRES

Principe :



RESEAUX EVOLUTIONNAIRES

Principe de l'apprentissage :

Pout tout point de départ d'apprentissage X_i
    Point_Courant = X_i
    $nb\_iter_i = 0$
    Tant que (point_cible A non atteint) et
            (Point_Courant n'est pas dans un obstacle) et
            (nb_iter_i < nb_iter_max) Faire
        Incrémenter nb_iter_i
        Calculer les 4 sorties du PMC pour Point_Courant (coordonnées et voisinage)
        Déplacement_Elémentaire = Max (Sortie $\rightarrow$, Sortie $\leftarrow$, Sortie $\uparrow$, Sortie $\downarrow$)
        Déplacer Point_Courant selon Déplacement_Elémentaire
    Fin Tant que
    Si (Point_Courant est dans un obstacle) Alors
        nb_iter_i = nb_iter_max
    Fin Si
Fin Pour tout

RESEAUX EVOLUTIONNAIRES

- Comment sélectionner les meilleurs réseaux ?
- Tenir compte de plusieurs critères :
  - la distance au point-cible
  - le nombre de déplacements effectués
  - le cumul des indices de présence de menace des cases parcourues
- D 'où la *fitness function* suivante à maximiser :

$$f(x) = \sum_{i=1}^{K} \frac{\alpha}{1+10\times d(X_i, A)} + \frac{\beta}{nb\_iter_i}$$

Avec Xi un point de départ parmi les K points de départ d 'apprentissage,

A le point-cible à atteindre et nb_iter_i le nombre de déplacements effectués depuis Xi.

RESEAUX EVOLUTIONNAIRES

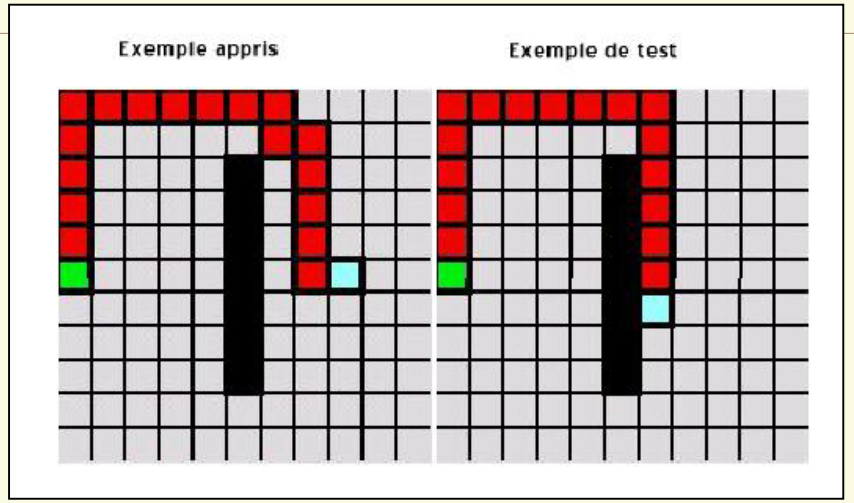
Paramètres AG :

- taille de la population : 100
- nombre de générations : 500
- taux de mutation : 0.1
- taux de croisement : 0.7
- fitness function :
  - $\alpha = 500$,
  - $\beta = 100$

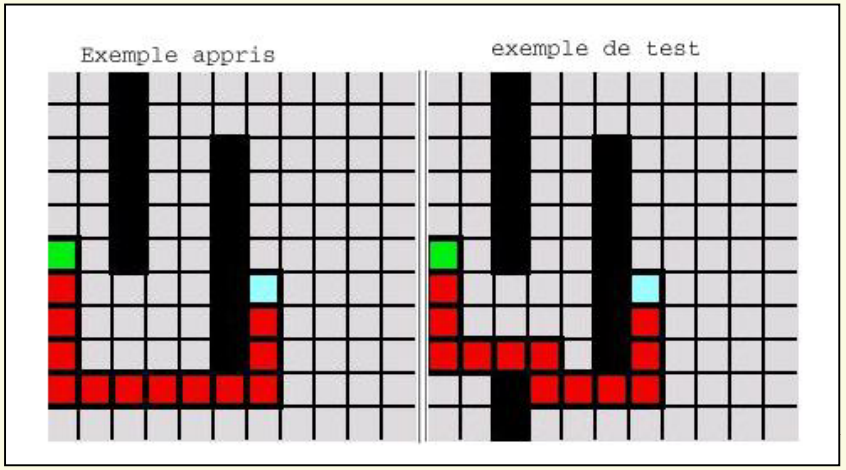
RESEAUX EVOLUTIONNAIRES

Tests : généralisation sur

- les points de départ



- la position et le nombre des obstacles



RESEAUX EVOLUTIONNAIRES

Références :

• ***Darwin revisité par la sélection artificielle***, La Recherche n° de février 2002

• M. Schoenauer et E. Ronald, ***Neuro-Genetic Truck Backer-Upper Controller***, 1994

• D. Floreano et F. Mondada, ***Evolution of Homing Navigation in a Real Mobile Robot***, 1996

EXERCICE: Réseaux évolutionnaires

Apprentissage du comportement suivant :

- Trouver le plus court-chemin jusqu'à un point-cible
- Eviter des obstacles
- Traverser le moins de menaces (zones dangereuses) possible