



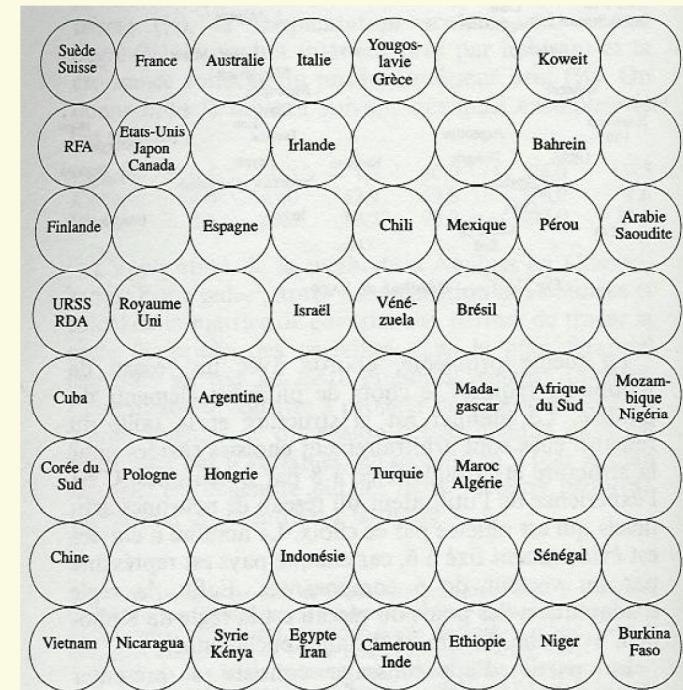
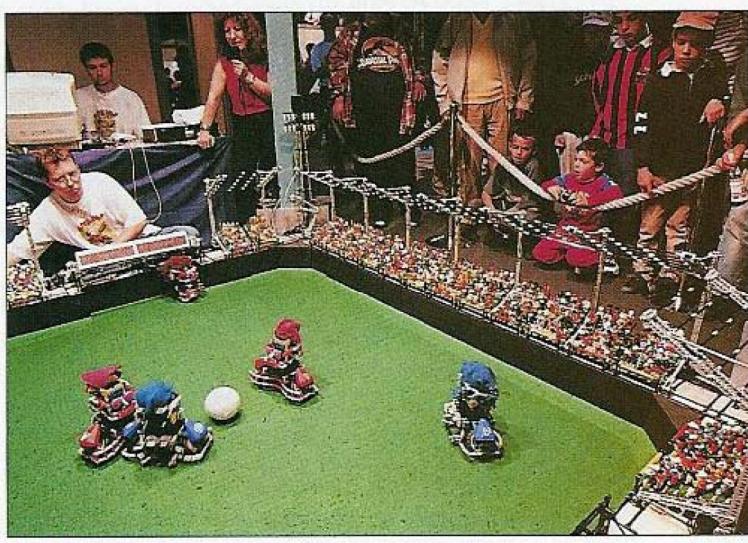
RESEAUX DE NEURONES

- *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

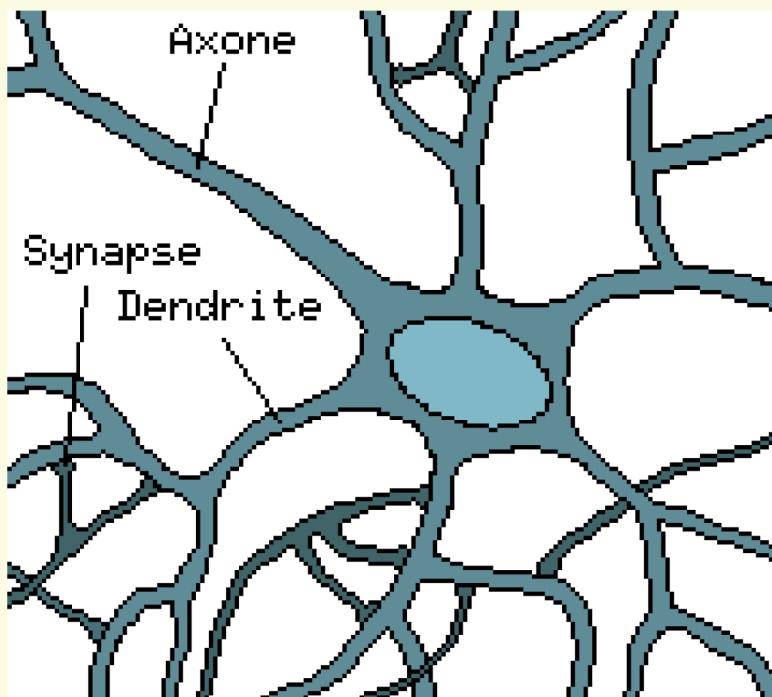
Applications :

- 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

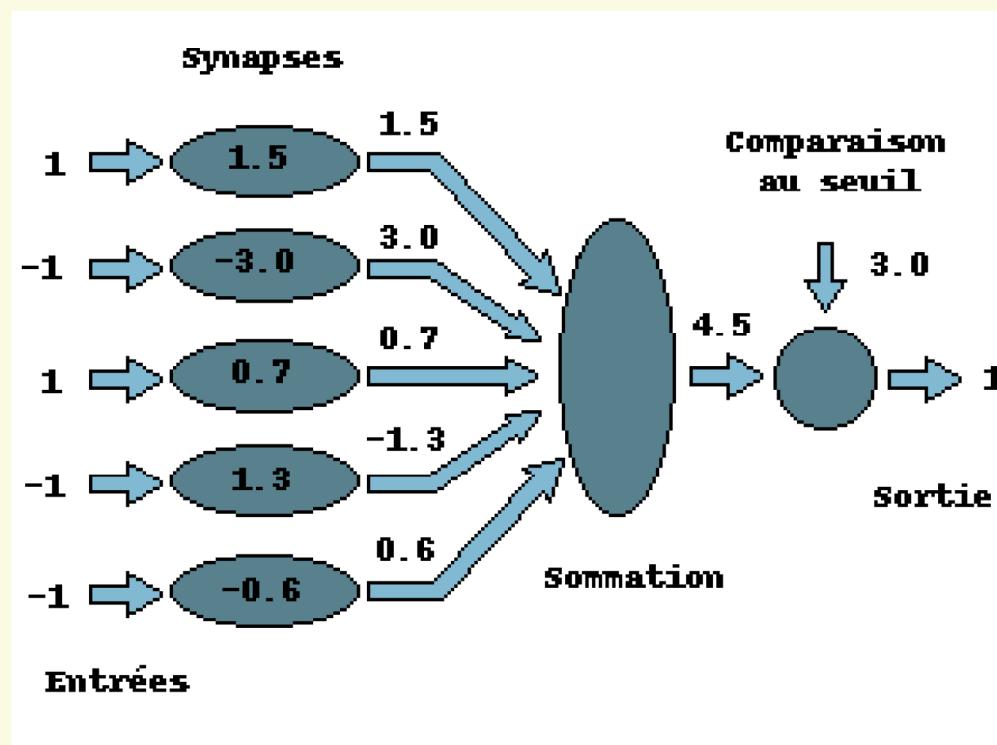
Historique :

- 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

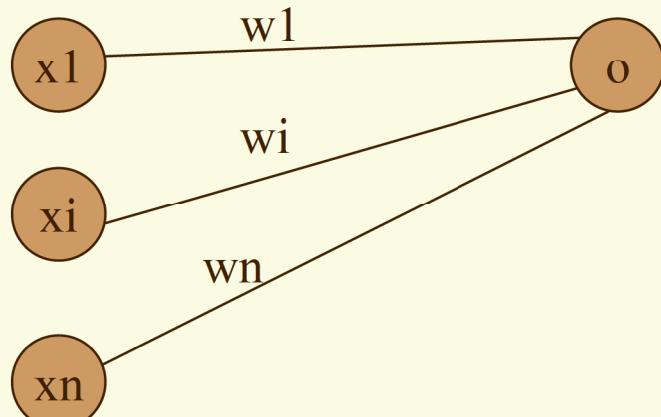
Modélisation :

Le neurone reçoit les entrées $x_1, \dots, x_i, \dots, x_n$.

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

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

$$\text{Soit : } p = x \cdot w = x_1 \cdot w_1 + \dots + x_i \cdot w_i + \dots + x_n \cdot w_n$$



$$\begin{aligned} \text{Alors : } & o = 1 \text{ si } p > \theta \\ & o = 0 \text{ si } p \leq \theta \end{aligned}$$

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 la 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

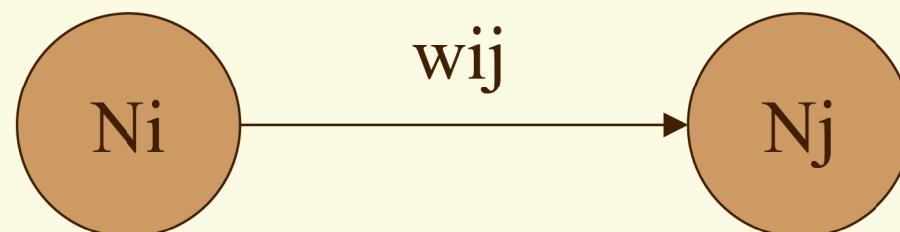
Définitions :

- *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 e_1, \dots, e_n
- m neurones N_1, \dots, N_m .
- w_{ij} le coefficient synaptique de la liaison entre les neurones N_i et N_j
- une sortie o
- un seuil S
- Fonction de transfert : fonction Signe
 - si $x > 0$: $\text{Signe}(x) = +1$
 - si $x \leq 0$: $\text{Signe}(x) = -1$



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 (e_1, \dots, e_n)

c la sortie correspondante souhaitée

Définitions :

a_i est la valeur d 'activation du neurone N_i .

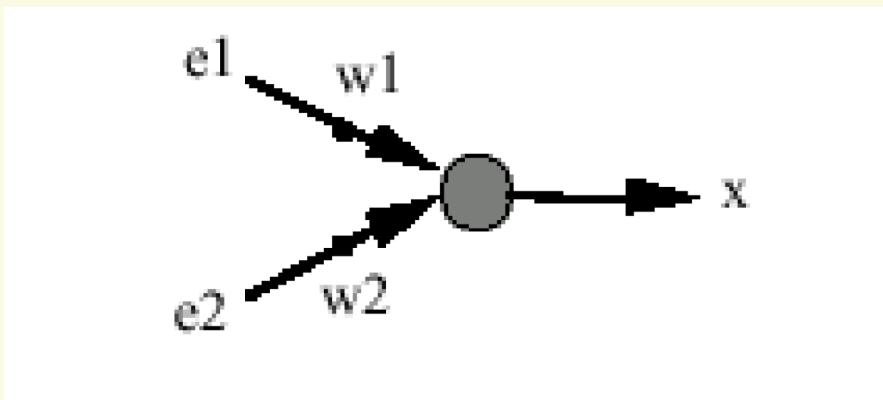
LOI DE HEBB

Algorithme :

- μ est une constante positive.
- Initialiser aléatoirement les coefficients w_i
- Répéter :
 - Prendre un exemple (e, c) dans S
 - Calculer la sortie o du réseau pour l 'entrée e
 - Si $c \neq o$
 - Modification des poids w_{ij} :
 - $w_{ij} = w_{ij} + \mu * (a_i * a_j)$
 - 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) : $o = \text{Signe}(w_1 * e_1 + w_2 * e_2) = \text{Signe}(0) = -1$
 $o = -1 \neq 1 = x$
 $\Rightarrow w_1 = w_1 + e_1 * x = 1$
 $w_2 = w_2 + e_2 * x = 1$
- Exemple (2) : $o = \text{Signe}((w_1 * e_1 + w_2 * e_2)) = \text{Signe}(0) = -1$
 $o = -1 \neq 1 = x$
 $\Rightarrow w_1 = w_1 + e_1 * x = 2$
 $w_2 = w_2 + e_2 * x = 0$
- Exemples (3) et (4) OK
- Exemples (1) et (2) OK \Rightarrow STOP



LOI DE HEBB

Remarque :

- Calcul des coefficients w_{ij} 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.
- $w_{ij} = \sum[(e,c) \text{ dans } S] (a_i * a_j)$

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

image apprise

image soumise
au réseau

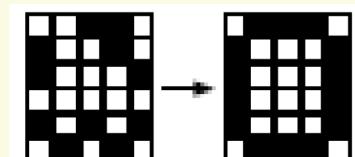
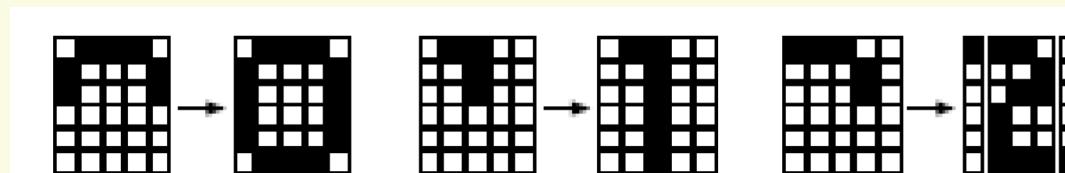
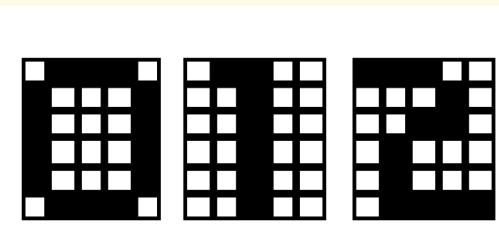
Image restituée
par le réseau



EXERCICE : Loi de Hebb

Reconstitution d'images :

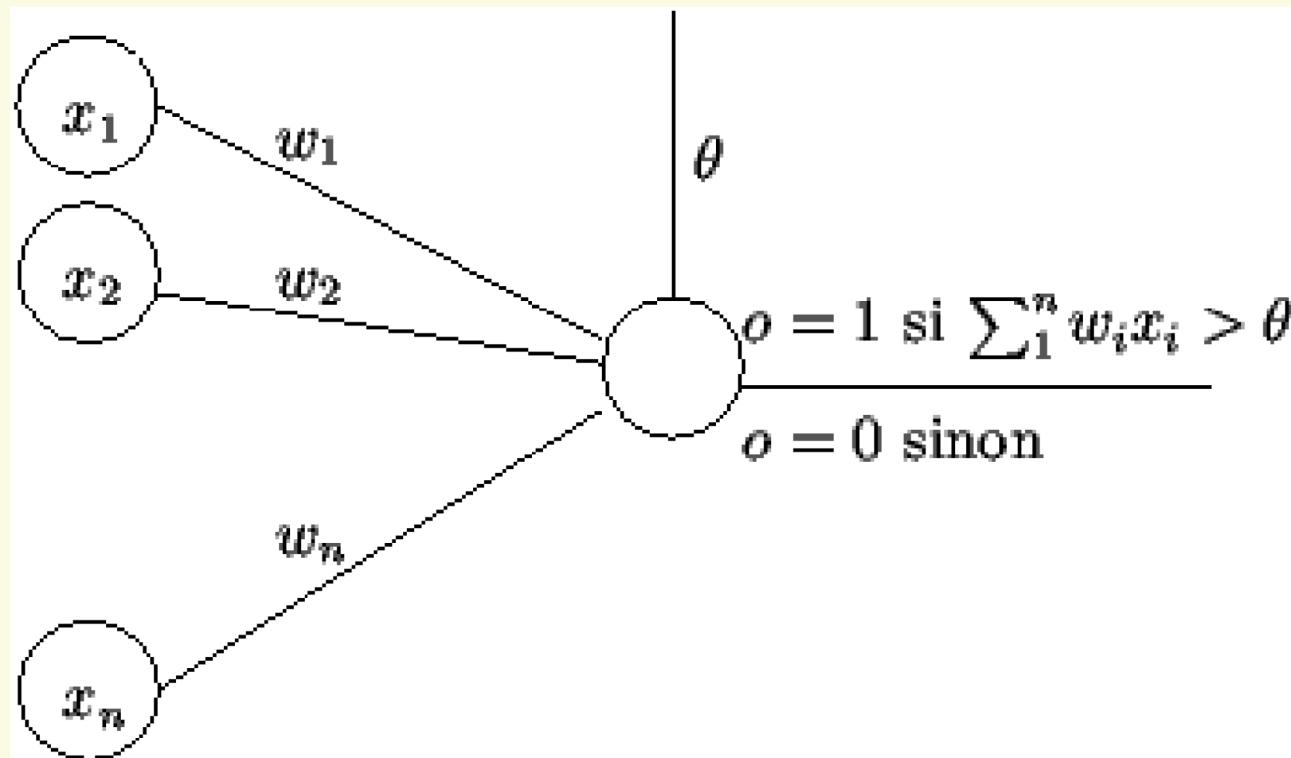
les chiffres 0, 1 et 2



PERCEPTRON

Perceptron linéaire à seuil :

- n entrées x_1, \dots, x_n
- n coefficients synaptiques w_1, \dots, w_n
- une sortie o
- un seuil θ



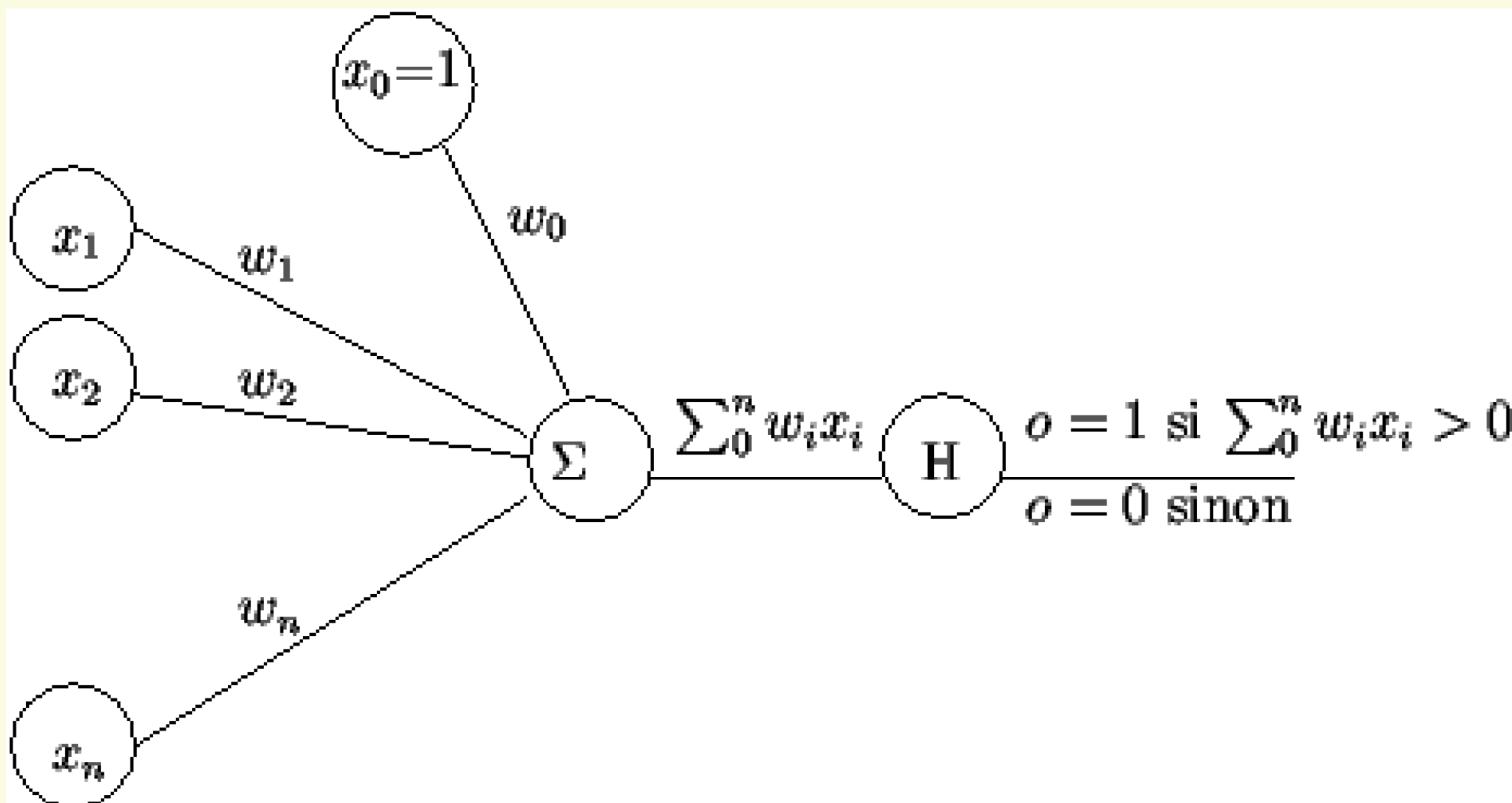
PERCEPTRON

On ajoute une entrée supplémentaire x_0 (le biais), avec le coefficient synaptique suivant : $w_0 = -\theta$

On associe comme fonction de transfert la fonction de Heavyside :

$$f(x) = 1 \text{ si } x > 0$$

$$f(x) = 0 \text{ sinon}$$



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 (x_0, x_1, \dots, x_n)
c la sortie correspondante souhaitée

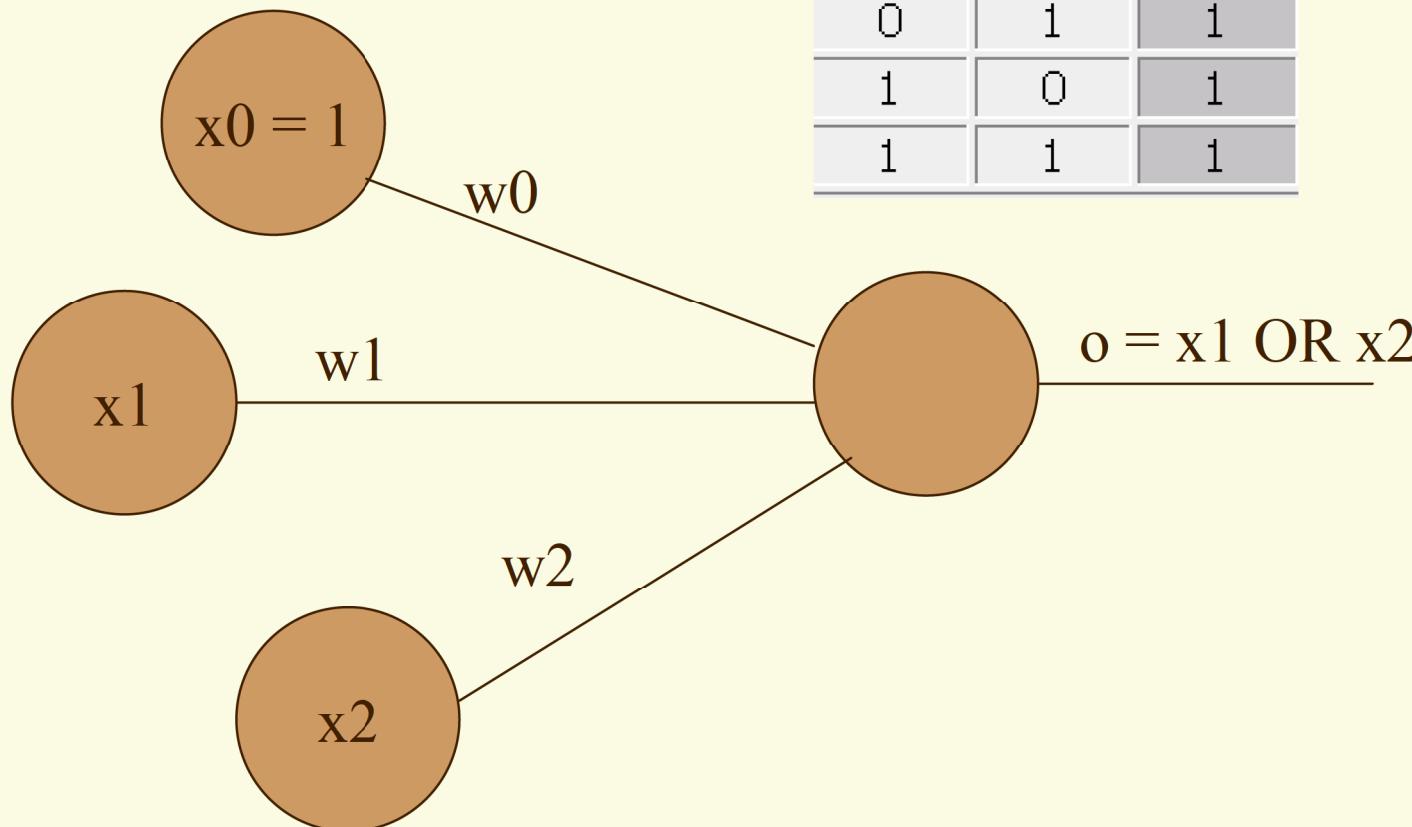
On cherche à déterminer les coefficients (w_0, w_1, \dots, w_n).

- Initialiser aléatoirement les coefficients w_i .
- 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 :
 - $w_i = w_i + \varepsilon * (c - o) * x_i$
 - 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

$\varepsilon = 1$

x_0 vaut toujours 1

Initialisation : $w_0 = 0$; $w_1 = 1$; $w_2 = -1$

| Étape | w_0 | w_1 | w_2 | Entrée | $\sum w_i x_i$ | 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 | $0+1x1$ | $1+1x0$ | $-1+1x1$ |
| 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 : $w_0 = 0$; $w_1 = 1$; $w_2 = 1$

Ce perceptron calcule le OU logique pour tout couple (x_1 ; x_2)

PERCEPTRON

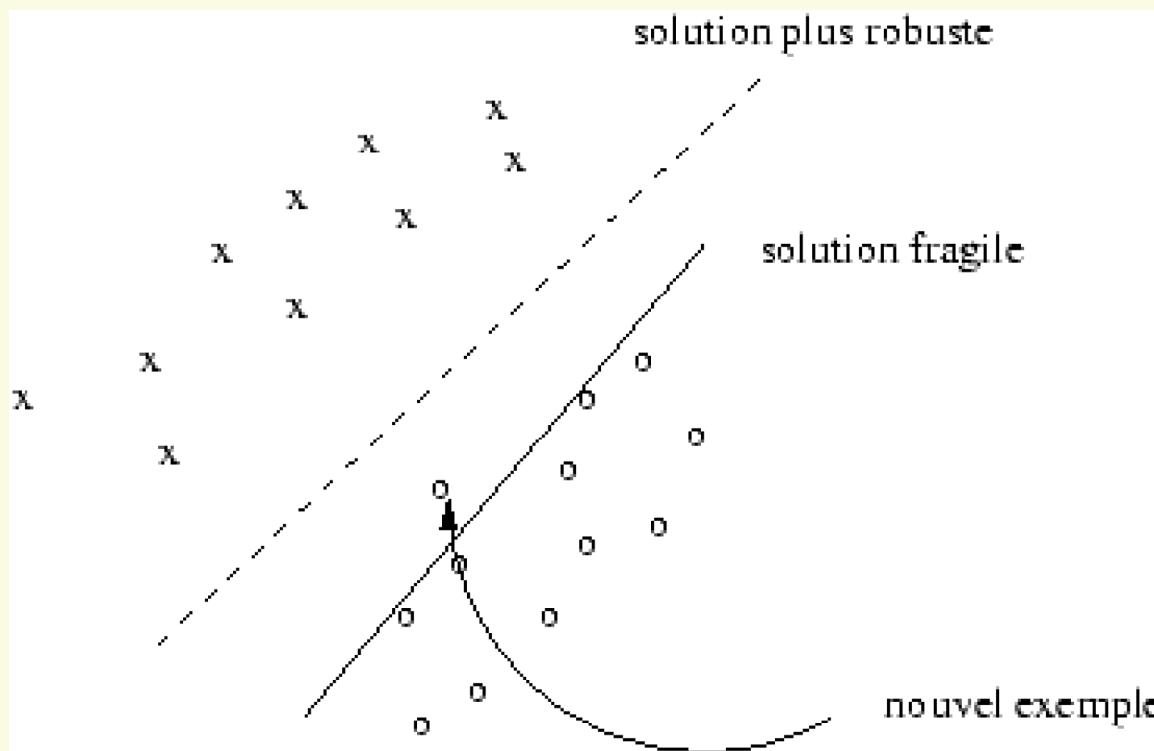
Apprentissage par l 'algorithme du perceptron : remarques

- 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 : $x \rightarrow f(x)$

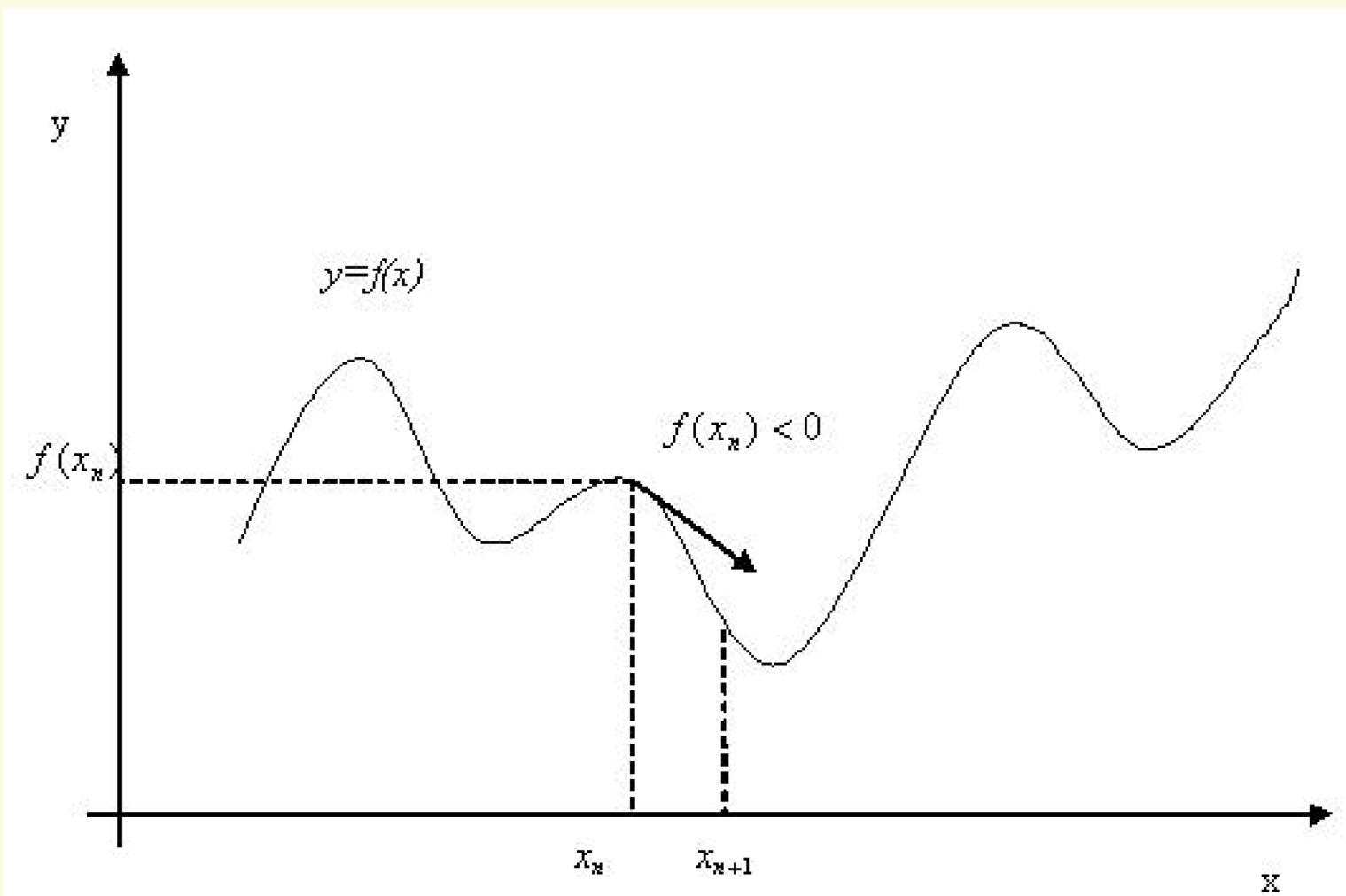
On construit la suite :

X_n telle que :

- On part d'une valeur initiale X_0 quelconque
- $X_{n+1} = X_n - \varepsilon * f'(X_n)$, 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 \Rightarrow pas de convergence
- Rien ne garantit que le minimum trouvé est un minimum global

Optimisation : gradient (rappels)

Dans la pratique :

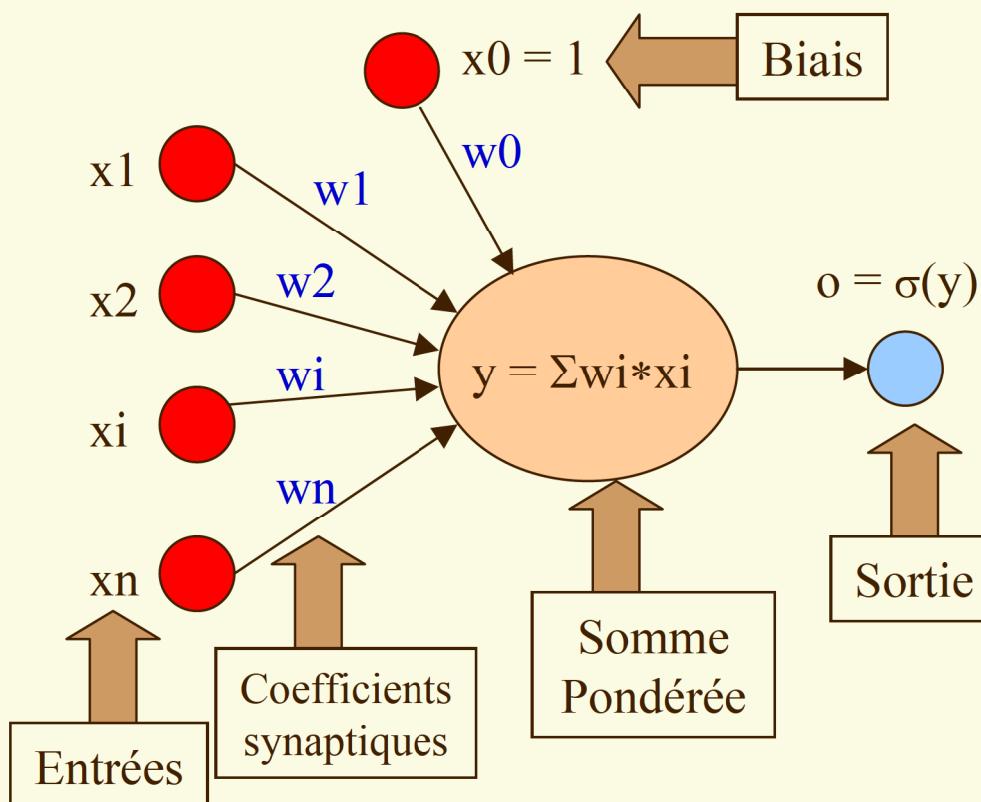
- Critères d'arrêt :
 - nombre d'itérations max.
 - norme($f(X_{i+1}) - f(X_i)$) / norme ($X_{i+1} - X_i$) inférieure à η (valeur très petite)
- ε est réajusté à chaque itération
 - valeur initiale ε_0 (0.01 par exemple)
 - 1-ère stratégie :
 - si $f(X_{i+1}) > f(X_i)$ i.e. f augmente, on augmente ε 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 :

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

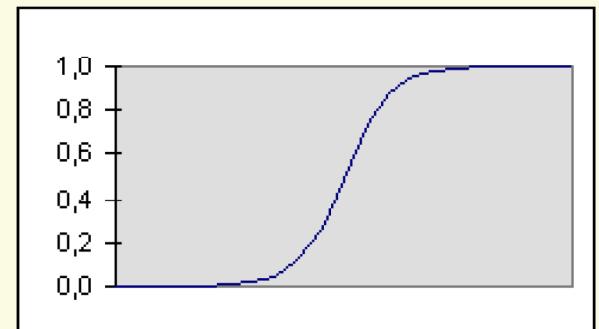
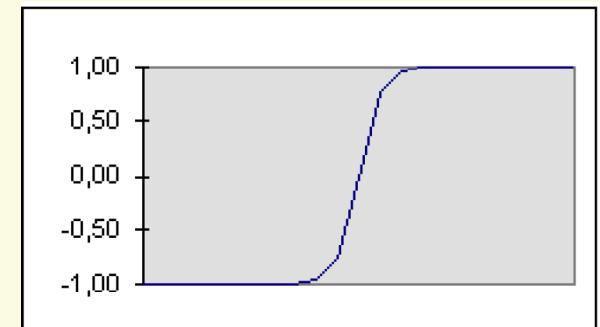
$X_0 = 1$: **biais**
Représente le seuil
d 'activation du neurone

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 : $o = \sigma(x.w) = \sigma(x_0.w_0 + \dots + x_n.w_n)$, σ étant une fonction de transfert continue et dérivable.

Posons : $y = x.w$

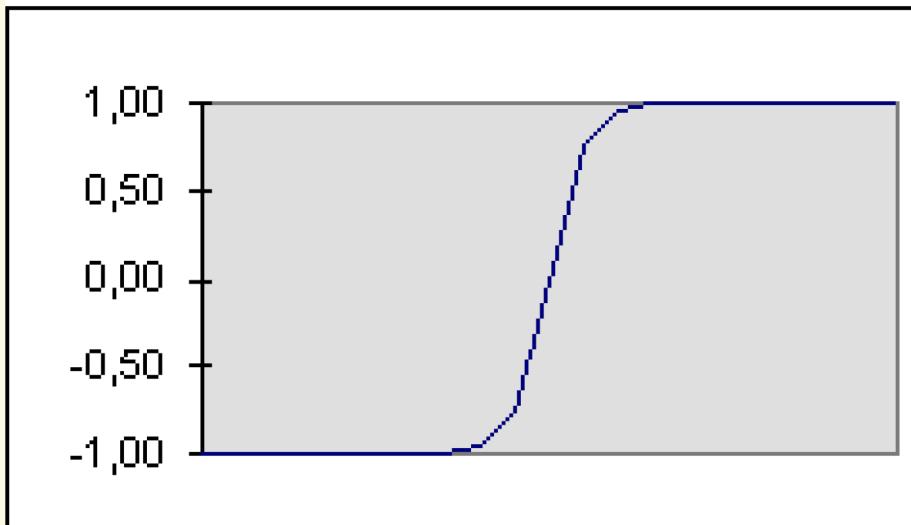
Soit S la base d 'apprentissage composée de couples (x, c) , où c est la sortie attendue pour x .

On définit ainsi l 'erreur sur le réseau pour la base d 'apprentissage S :
 $E(w) = 1/2 \sum_{(x,c) \text{ dans } S} (c - o)^2$

Problème : trouver w qui minimise $E(w)$.
⇒ Méthode du gradient.

PERCEPTRON : REGLE DELTA GENERALISEE

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

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

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

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

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

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

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

PERCEPTRON :

REGLE DELTA GENERALISEE

Apprentissage par descente de gradient : algorithme

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

- Répéter :
 - Prendre un exemple (x, c) dans S
 - Calculer la sortie o du réseau pour l 'entrée x
 - Pour i de 1 à n :
 - $w_i = w_i + \varepsilon * (c - o) * x_i * \sigma'(x.w)$
 - 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 w_i .
- Répéter :
 - Prendre un exemple (x, c) dans S
 - Calculer la sortie o du réseau pour l'entrée x
 - Pour i de 1 à n :
 - $w_i = w_i + \varepsilon * (c - o) * x_i$
 - Fin Pour
- Fin Répéter

PERCEPTRONS / REGLES DELTA

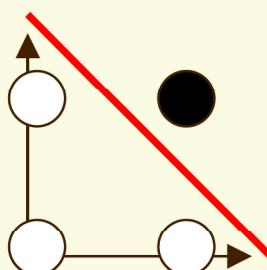
Remarques

- 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, ...).

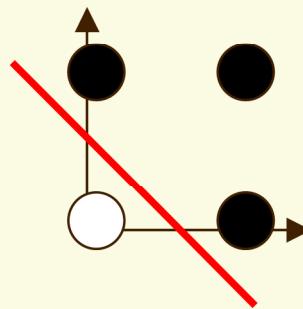
PERCEPTRONS / REGLES DELTA

Remarques

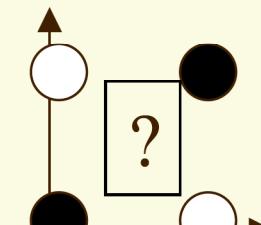
- Perceptrons = classificateurs linéaires (ensembles linéairement séparables).



AND



OR



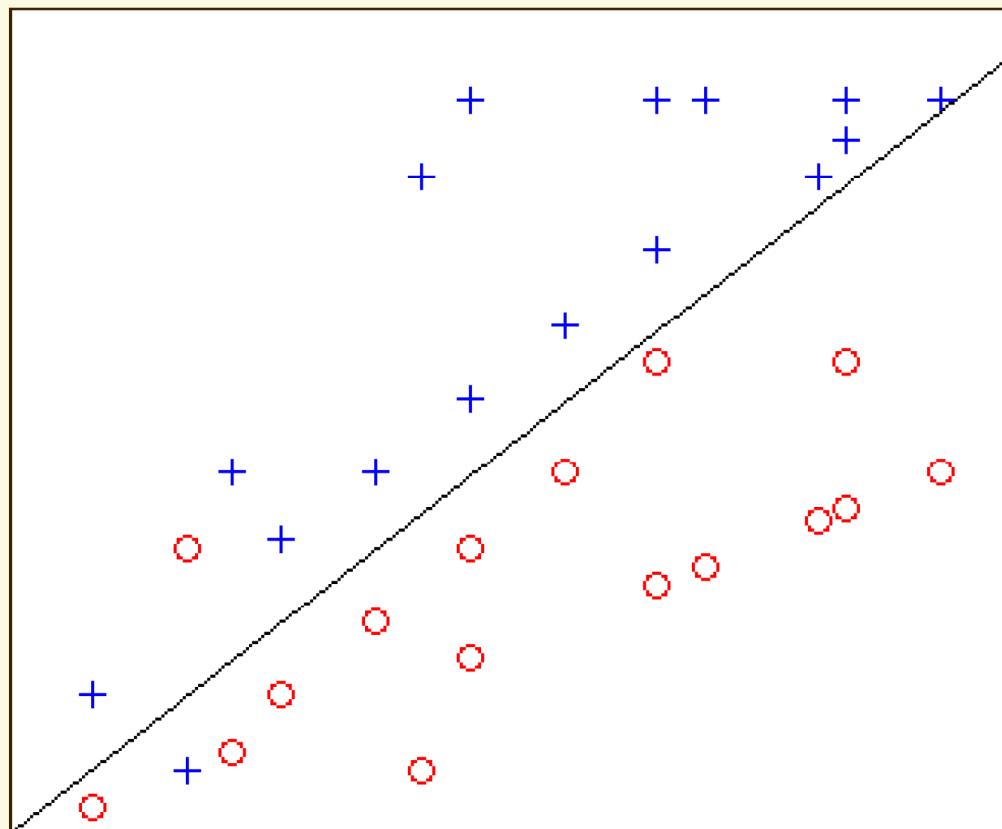
XOR

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

EXERCICE : perceptron

Classificateur linéaire :

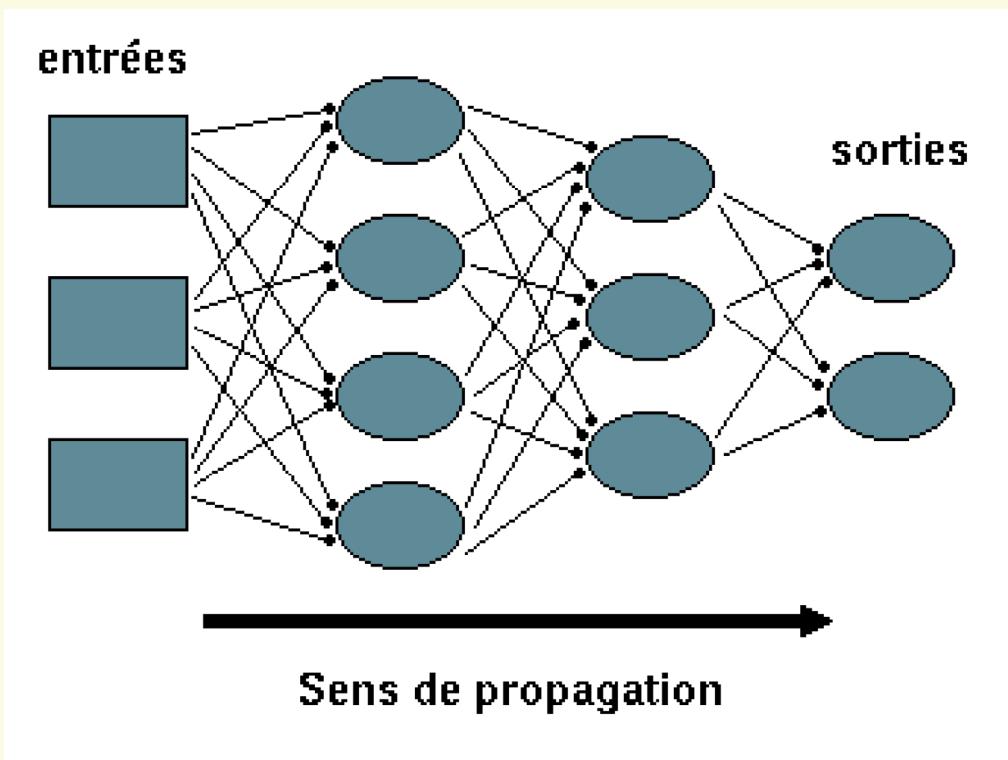
Trouver la droite qui sépare deux groupes de points.



PERCEPTRON MULTI-COUCHES

Les différentes couches :

- Cellules réparties dans q couches : C₀, C₁, ..., C_q
- C₀ : couche d 'entrée (la rétine) \Rightarrow les variables d 'entrée
- C_q : couche de sortie
- C₁, ..., C_{q-1} : les couches cachées



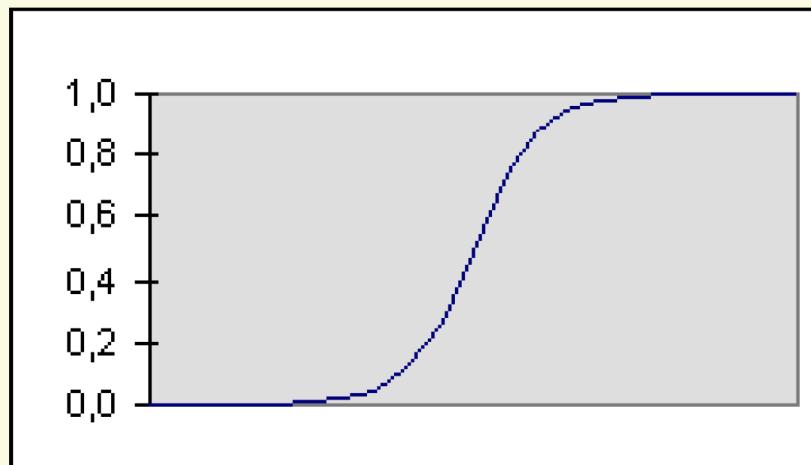
PERCEPTRON MULTI-COUCHES

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 :

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

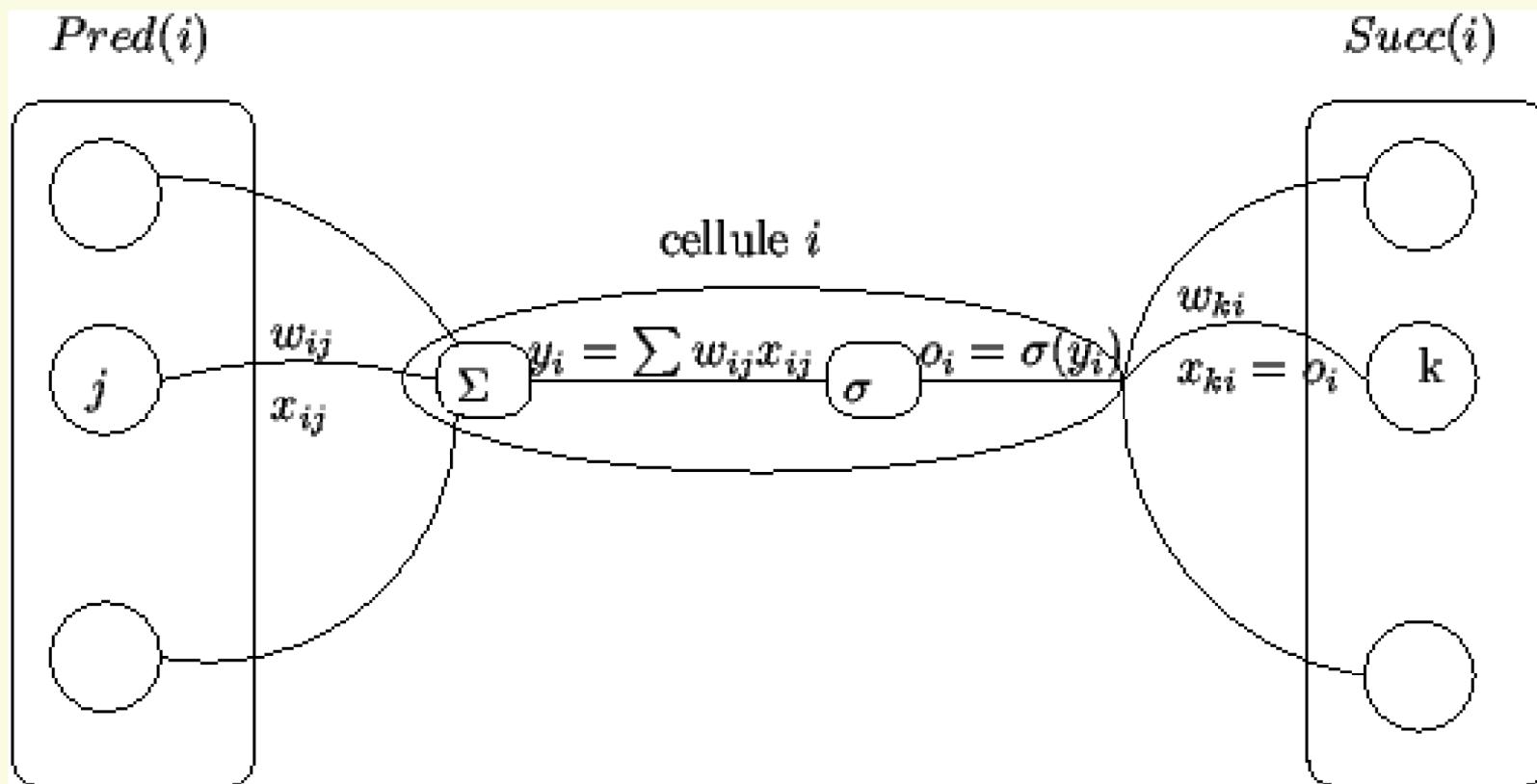
PERCEPTRON MULTI-COUCHES

Notations :

- n cellules
- Cellules désignées par un indice i , $0 \leq i < n$
- p cellules de sortie
- k indice d'une cellule de sortie
- c_k : sortie attendue pour la cellule de sortie k avec l'entrée x
- o_k : sortie calculée pour la cellule de sortie k avec l'entrée x
- x_{ij} : entrée associée au lien entre cellule I vers cellule j
- w_{ij} : coefficient synaptique associé au lien entre cellule i vers cellule j
- $\text{Succ}(i)$: ensemble des cellules qui prennent comme entrée la sortie de la cellule i .
- $\text{Pred}(i)$: ensemble des cellules dont la sortie est une entrée de la cellule i .
- y_i : entrée totale de la cellule i : $y_i = \sum(j \in \text{Pred}(i)) (w_{ij} * x_{ij})$
- o_i : sortie de la cellule I : $o_i = \sigma(y_i)$

PERCEPTRON MULTI-COUCHES

Notations :



PERCEPTRON MULTI-COUCHES

Algorithme de rétropropagation du gradient :

- Initialiser aléatoirement les coefficients w_{ij} dans $[-0.5 ; 0.5]$
- Répéter
 - Prendre un exemple (x, c) de S
 - Calculer la sortie o
 - Pour toute cellule de sortie i
 - $\delta_i = \sigma'(y_i) * (c_i - o_i) = o_i * (1 - o_i) * (c_i - o_i)$
 - Fin Pour
 - Pour chaque couche de $q - 1$ à 1
 - Pour chaque cellule i de la couche courante
 - $\delta_i = \sigma'(y_i) * [\sum(k \in \text{Succ}(i)) (\delta_k * w_{ki})]$
 $= o_i * (1 - o_i) * [\sum(k \in \text{Succ}(i)) (\delta_k * w_{ki})]$
 - Fin Pour
 - Fin Pour
 - Pour tout poids w_{ij}
 - $w_{ij} = w_{ij} + \varepsilon * \delta_i * x_{ij}$
 - Fin Pour
- Fin Répéter

PERCEPTRON MULTI-COUCHES

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

- Initialiser aléatoirement les coefficients w_i dans [-0.5 ; 0.5]
- Répéter
 - Prendre un exemple (x, c) de S
 - Calculer la sortie o
 - Pour toute cellule de sortie i
 - $\delta_i = \sigma'(y_i) * (c_i - o_i) = o_i * (1 - o_i) * (c_i - o_i)$
 - Fin Pour
 - Pour chaque couche de $q - 1$ à 1
 - Pour chaque cellule i de la couche courante
 - $\delta_i = \sigma'(y_i) * [\sum(k \in \text{Succ}(i)) (\delta_k * w_{ki})]$
 - $= o_i * (1 - o_i) * [\sum(k \in \text{Succ}(i)) (\delta_k * w_{ki})]$
 - Fin Pour
 - Fin Pour
 - Pour tout poids w_{ij} à litération k
 - $w_{ij}(k) = w_{ij}(k-1) + \varepsilon * \delta_i * x_{ij} + \beta * (w_{ij}(k - 1) - w_{ij}(k - 2))$
 - 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

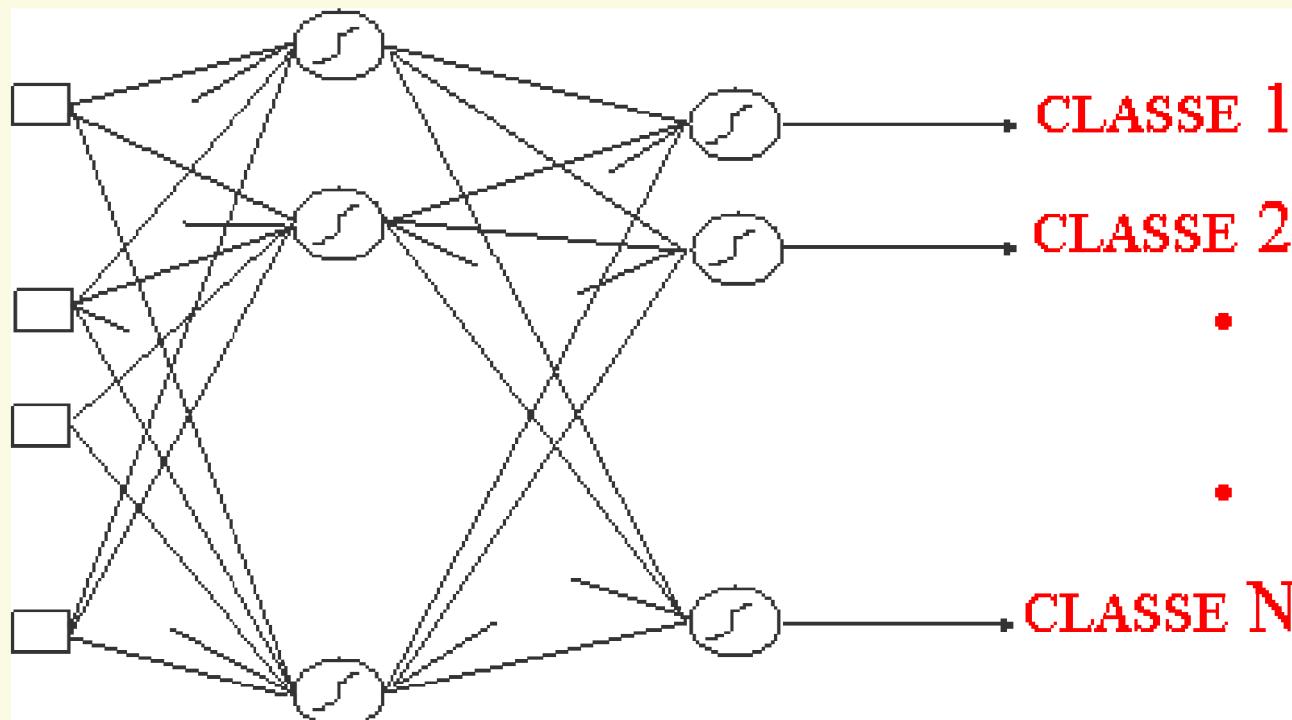
Classification / Discrimination :

- 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 :

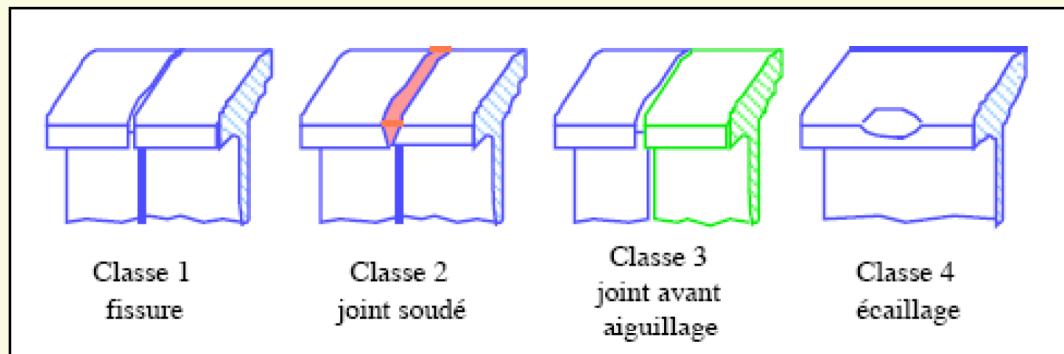
- Inconvénients :
 - apprentissage très long (très nombreuses connexions)
 - une nouvelle classe : il faut tout réapprendre !
- 2-ème modélisation : un réseau par classe
 - pour chaque réseau, une sortie indique si l'exemple soumis appartient à la classe correspondante

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*.
 ⇒ 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
- ...