

# INTRODUCTION AUX RESEAUX DE NEURONES

**Fabien MOUTARDE**  
**Centre de Robotique (CAOR)**  
**Mines ParisTech (ENSMP)**

`Fabien.Moutarde@mines-paristech.fr`

`http://www.ensmp.fr/~moutarde`

## PLAN GENERAL

**A • Présentation générale des réseaux de neurones**

**B • Application des réseaux neuronaux en Automatique**

**C • Travaux pratiques sur machine**

**1) INTRODUCTION - HISTORIQUE**

**2) LES RESEAUX DE NEURONES FORMELS**

**3) LES TECHNIQUES D'APPRENTISSAGE**

**4) EXEMPLE**

## HISTORIQUE

---

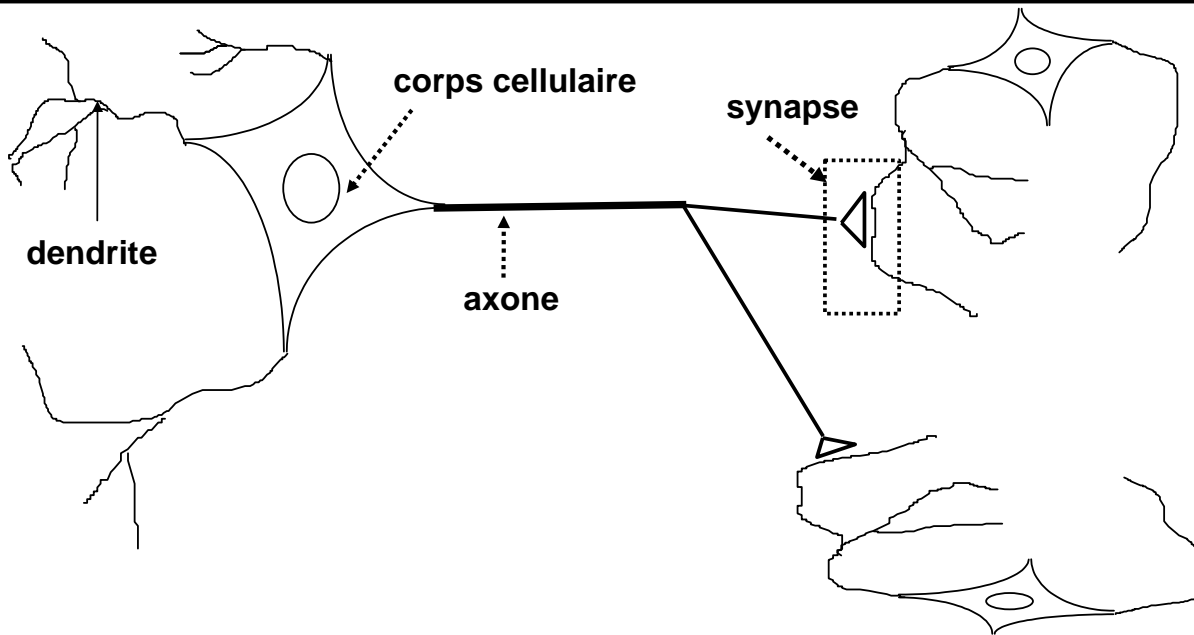
- **Compréhension et modélisation cerveau**



- **Essai imitation pour reproduire fonctions évoluées**



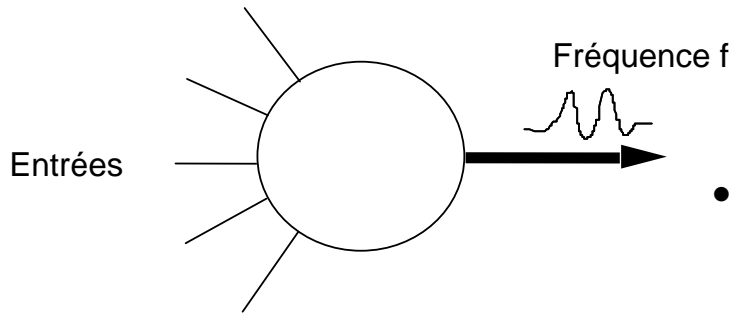
- **Outils mathématiques pour l'ingénieur**



- **Signal électrique dendrites --> corps --> axone --> synapses**

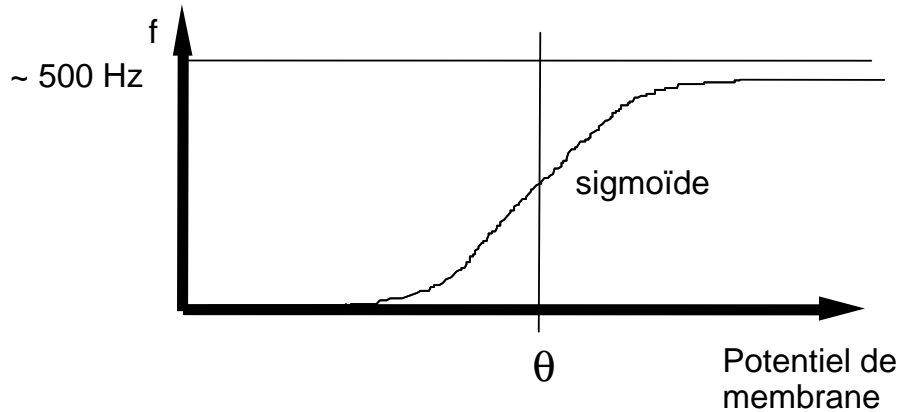
## FORMATION DU CERVEAU - APPRENTISSAGE - MEMOIRE

- **FORMATION:** programme héréditaire dicté par
  - un réseau de neurones totalement connecté
  - le mécanisme interne de chaque type de neurones
  - le potentiel d'évolution des synapses
- **EVOLUTION ET APPRENTISSAGE :**
  - interaction avec l'extérieur
  - évolution des synapses qui peuvent
    - dégénérer
    - se stabiliser de façon +/- réversible (mémoire)



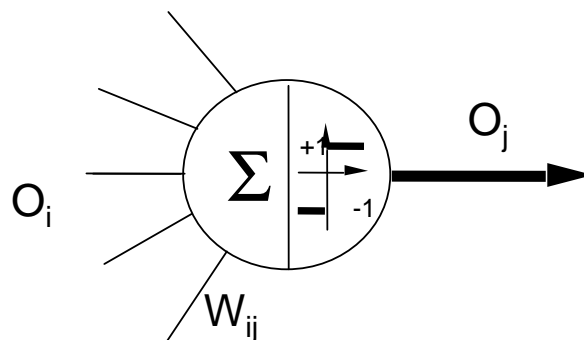
- potentiel de membrane

$$q \approx \sum_i C_i f_i$$



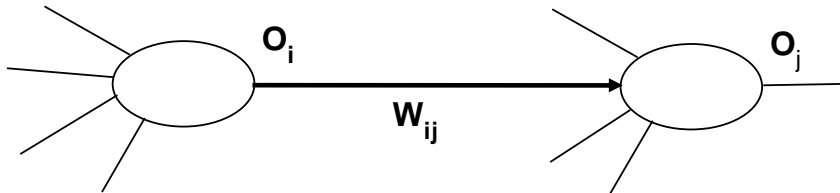
- LES DEBUTS : Mc Culloch et Pitts (1943)

- modèle de neurone simple
- but de modéliser le cerveau



- Règle de Hebb (1949)

**Renforcement des connexions entre neurones  
activés simultanément**



$$W_{ij}(t + \delta t) = W_{ij}(t) + \lambda O_i O_j$$

## PREMIERS "RESEAUX DE NEURONES FORMELS"

---

- LE PERCEPTRON (Rosenblatt, 1957)
- ADALINE (Widrow, 1962)

neurone formel Mac Culloch & Pitts  
+  
règle d'apprentissage de Hebb



**réalisation de fonctions booléennes  
par apprentissage à partir d'exemples**

- **PERCEPTRONS, le livre de Minsky et Papert (1969)**

**Etude approfondie du perceptron et de ses limites intrinsèques :**

- certaines fonctions booléennes impossibles à apprendre  
(XOR, ...)
- limitation aux séparations linéaires

- **Les progrès des ordinateurs séquentiels ont poussé le développement du traitement symbolique et de l'Intelligence Artificielle (systèmes experts) et l'abandon des réseaux de neurones.**

- **La mécanique statistique au secours des réseaux de neurones (J.J. Hopfield, 82)**  
--> intérêt de la dynamique des réseaux totalement connectés
- **LA RETROPROPAGATION DU GRADIENT et la résolution du Credit Assignment Problem (Rumelhart 86, Le Cun 85) : apprentissage de réseaux à couches cachées.**
- **Puissance de calcul des ordinateurs**  
==> résolution empirique de problèmes réels.
- **Des résultats mathématiques : approximateurs universels parcimonieux**

- « deep » neural networks, avec plusieurs couches cachées de grandes dimensions :
  - initialisation « intelligente » non-supervisée
  - apprentissage usuel ensuite → « fine-tuning » des poids
  - couches cachées → features appris (bas niveau sur première couche cachée, + ht niveau sur 2eme)

## Les réseaux de neurones aujourd'hui

- Une myriade de modèles connexionnistes et d'algorithmes d'apprentissage (et des milliers de thèses!)
- Beaucoup de produits logiciels (et de freewares) pour développer des réseaux neuronaux
- Quelques chips spécialisés

**Des outils mathématiques pour l'ingénieur (souvent extension non-linéaire de techniques classiques) à mettre sur le même plan que :**

- classifieurs usuels (Arbres de Décisions Binaire, Plus Proche Voisin, ...)
- splines et autres outils d'interpolation
- modèles ARMA, ...
- filtres linéaires adaptatifs, ...
- contrôleurs PID, ...
- ...

## Apprentissage d'une relation entrée-sortie connue par l'exemple :

- Reconnaissance de formes
- Classification
- Identification
- Prédiction
- Filtrage
- Commande, régulation

**+ Optimisation combinatoire (réseaux de Hopfield)**

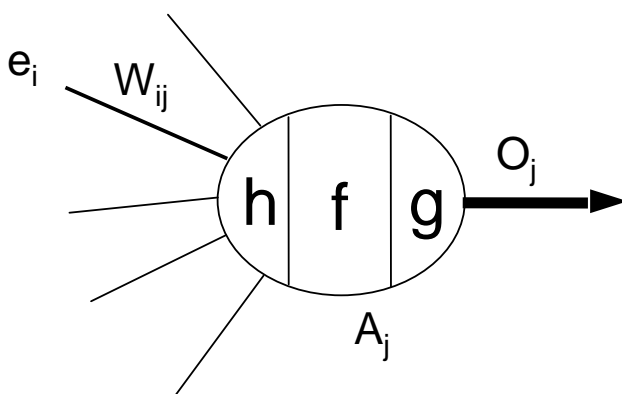
- reconnaissance de caractères (EasyReader), de visage (Mimetics), de la parole, de l'écriture (ordinateur sans clavier), de signature acoustique (Thomson), d'objets (Silac: profilés d'aluminium...)
- diagnostic (allumage voiture, Renault ; photocopieur, Canon; circuits VLSI, IBM...)
- compression de données (JVC)
- prévision de consommation d'eau (Lyonnaise, CGE), d'électricité (EDF), de trafic routier (Cofiroute), de cours boursiers...
- identification de procédé industriel (Air liquide, Elf Atochem, ciments Lafarge...)
- commande de véhicule (Sagem)

## DEFINITIONS DU NEURONE FORMEL

Une définition très générale : processeur qui applique une opération simple à ses entrées et que l'on peut relier à d'autres pour former un réseau qui peut réaliser une relation entrée-sortie quelconque.

**DEFINITION USUELLE** : processeur très simple qui calcule une somme pondérée et qui applique à cette somme une fonction de transfert non linéaire (échelon, sigmoïde, gaussienne, ...)

## LE NEURONE FORMEL



$e_i$  : entrées du neurone  
 $A_j$  : activation du neurone  
 $O_j$  : sortie du neurone

$W_{ij}$  : poids (synaptiques)  
 $h$  : fonction d'entrée  
 $f$  : fonction d'activation  
 (ou de transfert)  
 $g$  : fonction de sortie

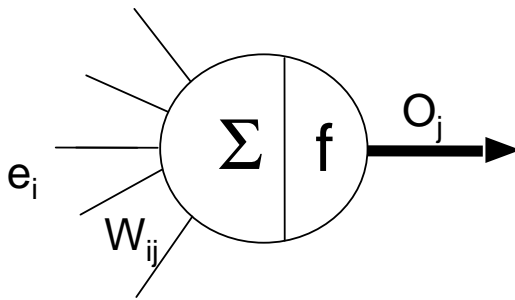
$$A_j = f( h(e_i, \{W_{ij}, i=0 \text{ à } k_j\}) )$$

$$O_j = g(A_j) (= A_j \text{ le plus souvent})$$

**La combinaison (h,f,g) définit le type de neurone**

# LE NEURONE SOMMATEUR

## PRINCIPE

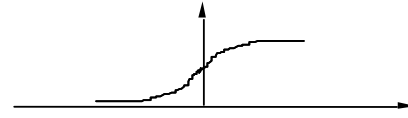


$$O_j = f \left( W_{0j} + \sum_{i=1}^{n_j} W_{ij} e_i \right)$$

$W_{0j}$  = "biais"

## FONCTIONS D'ACTIVATION

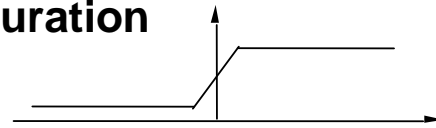
- **Fonction seuil (Heaviside) ou signe**  
--> neurones binaires
- **Fonction sigmoïde**  
--> neurones les plus utilisés



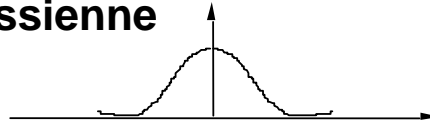
- **Identité**

--> neurones linéaires

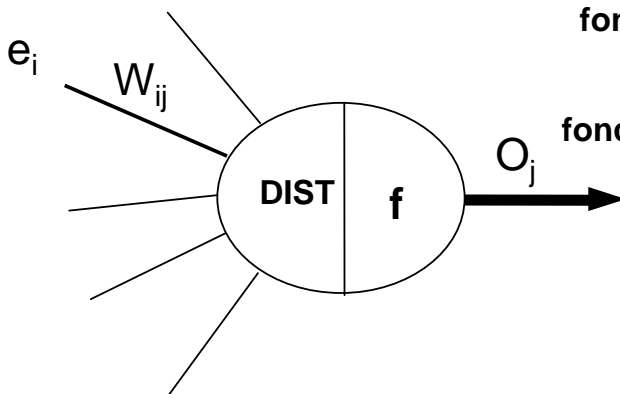
- **Saturation**



- **Gaussienne**



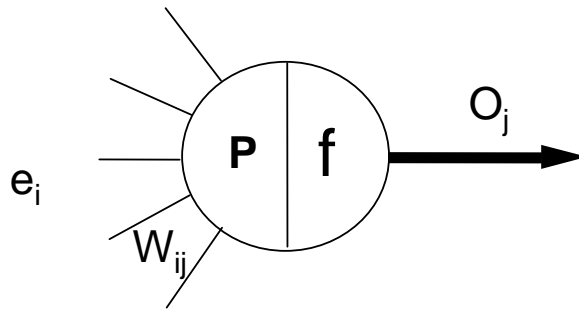
# LE NEURONE DISTANCE



$$\text{fonction d'entrée : } h \left( \begin{pmatrix} e_1 \\ \dots \\ e_k \end{pmatrix}, \begin{pmatrix} W_{1j} \\ \dots \\ W_{kj} \end{pmatrix} \right) = \sum_i (e_i - W_{ij})^2$$

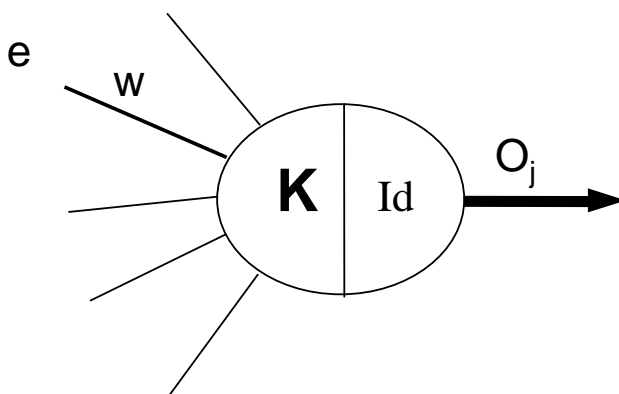
fonction d'activation = Identité ou gaussienne

**L'activation représente donc la distance entre le vecteur d'entrée  $(e_i)_i$  et le vecteur poids  $(W_{ij})_i$**



Par exemple, neurone d'ordre 2 :  $O_j = f \left( \sum_{i,k} (W_{ij} W_{kj} e_i e_k) \right)$

# NEURONES DE TYPE NOYAU



fonction d'entrée :  $h(e, w) = K(e, w)$   
 avec  $K$  symétrique et « positif »  
 au sens de Mercer :  $\forall \psi \text{ tq } \int \psi^2(x) dx < \infty$ ,  
 alors  $\int K(u, v) \psi(u) \psi(v) du dv \geq 0$   
fonction d'activation = identité

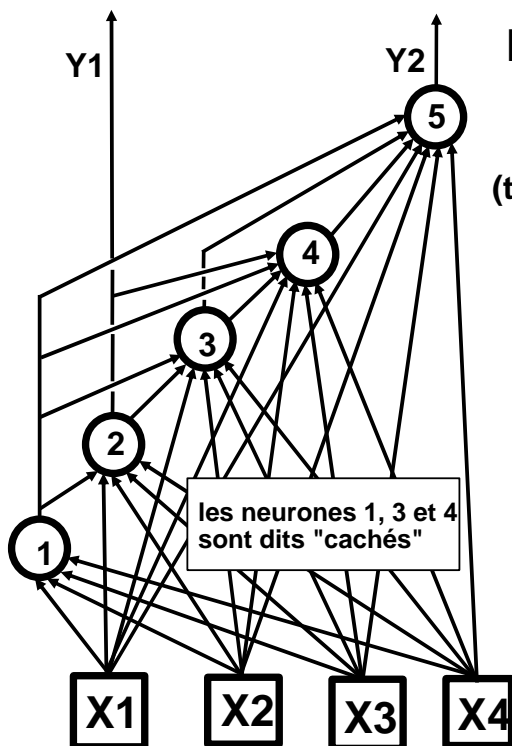
**Exemples de noyaux « admissibles » :**

- Polynomiaux :  $K(u, v) = [u \cdot v + 1]^p$
- Radial Basis Function :  $K(u, v) = \exp(-\|u - v\|^2 / 2\sigma^2)$   
 → équivalent à distance+activation gaussienne
- Sigmoides :  $K(u, v) = \tanh(u \cdot v + \theta)$   
 → équivalent à sommateur+activation sigmoïde

## DEUX FAMILLES DE RESEAUX

- **RESEAUX NON BOUCLES** : sont utilisés en classification, reconnaissance des formes (caractères, parole, ...), en prédiction
- **RESEAUX BOUCLES** : sont utilisés comme mémoire associative (Hopfield, réseaux à attracteurs) ou pour des tâches de traitement du signal ou de commande.

## RESEAUX NON BOUCLES

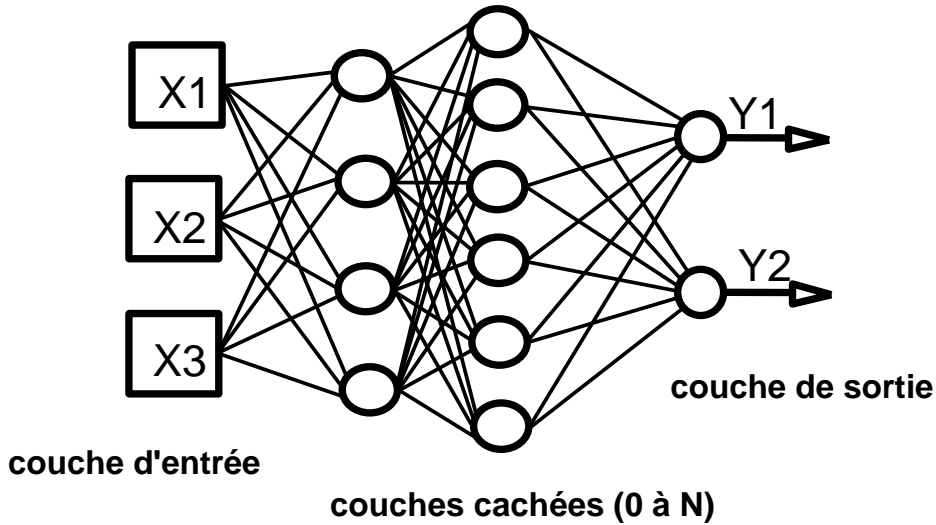


Les neurones peuvent être ordonnés d'une façon telle qu'il n'y a aucune connexion "vers l'arrière" (terme anglais : "FEEDFORWARD" neural network)



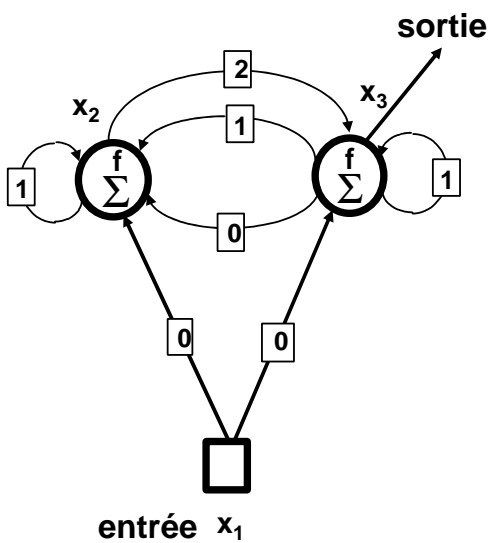
Le temps n'intervient pas comme variable fonctionnelle (i.e. le réseau n'a pas de mémoire et ses sorties ne dépendent pas de son passé)

# RESEAUX NON BOUCLES A COUCHES

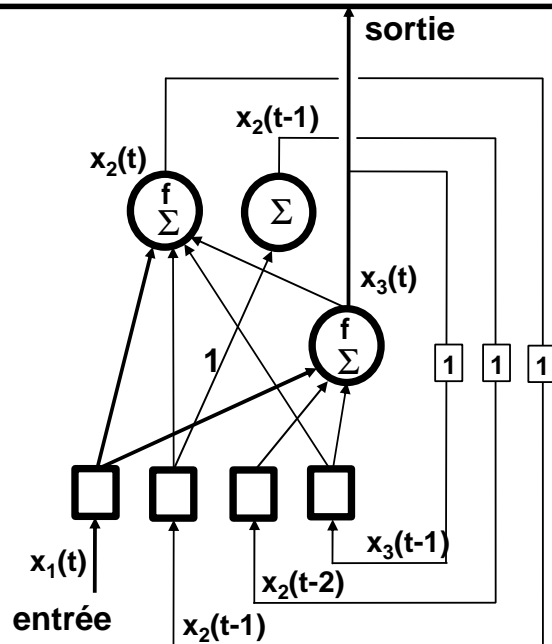


En général, avec neurones sommateurs à sigmoïde  
C'est le réseau le plus utilisé  
Nom anglais : Multi-Layer Perceptron (MLP)

# RESEAUX BOUCLES

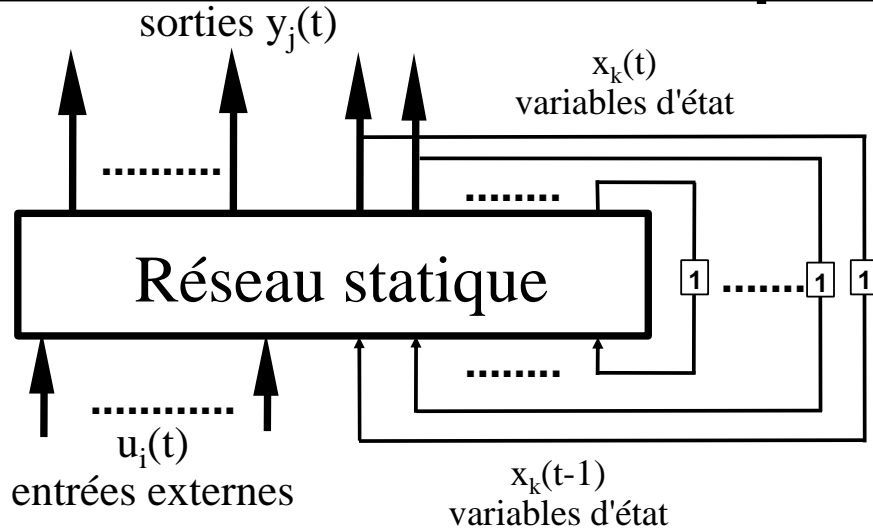


A chaque connexion  
est associé un délai



Forme équivalente

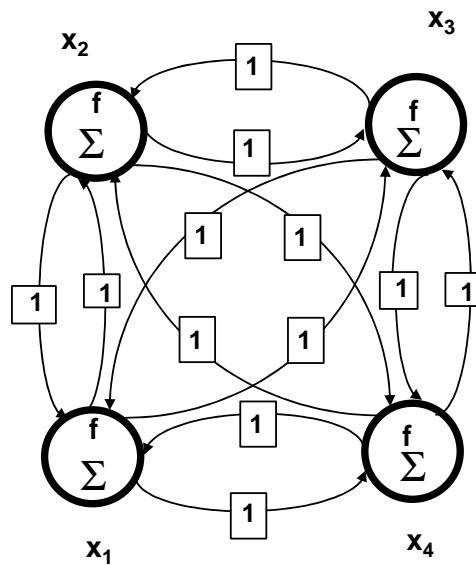
# RESEAUX BOUCLES : forme canonique



## FORME CANONIQUE DES RESEAUX BOUCLES

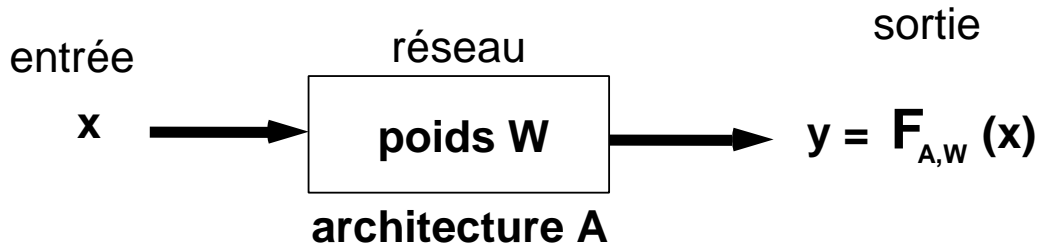
Les sorties à t dépendent non seulement des entrées externes à t, mais aussi (via les variables d'état) de toute la séquence des entrées externes précédentes (et de l'initialisation des variables d'état)

# RESEAU DE HOPFIELD



## Réseau bouclé totalement connecté

pas d'entrées externes (à part initialisation des états des neurones)



- **Deux modes :**

- **apprentissage** : à partir d'exemples de couples (entrée, sortie) , le réseau modifie
  - les paramètres  $W$  (poids des connexions)
  - éventuellement son architecture  $A$   
(en créant/éliminant neurones ou connexions)
- **reconnaissance** :  
calcul de la sortie associée à une entrée

## PLAN

- Généralités
- Apprentissage supervisé
- Apprentissage non supervisé
- Méthodologie pour l'apprentissage

## MODES D'APPRENTISSAGE

### SUPERVISE

exemples :  $(x_1, d_1) \dots (x_m, d_m)$   
Architecture : A  
Poids initiaux :  $W^o$

### NON SUPERVISE

exemples :  $x_1, \dots, x_m$   
Architecture : A  
Poids initiaux :  $W^o$

**TROUVER DES POIDS W (et éventuellement modifier A) POUR QUE :**

- interpolation :  $d_k = F_{A,W}(x_k)$  pour  $k=1, \dots, m$
- généralisation :  $d = F_{A,W}(x)$   
pour de nouveaux couples  $(x,d)$

- Les images des exemples sont bien groupées
- Les images des nouveaux exemples sont "proches" de leurs semblables

## L'APPRENTISSAGE SUPERVISE

- Réseaux de neurones sommateurs à seuil
  - la séparation linéaire
  - les réseaux multicouches : la rétropropagation
- Réseaux de neurones "distance"
  - Learning Vector Quantization (LVQ)
  - Méthodes à noyaux type Radial Basis Function
- Réseaux à neurones d'ordre supérieur
- Support Vector Machines (SVM)

- L'apprentissage supervisé est l'adaptation des coefficients synaptiques d'un réseau afin que pour chaque exemple, la sortie du réseau corresponde à la sortie désirée.
- **FONCTION DE COÛT** : problème de minimisation  
Soient  $n$  exemples  $(X_p; D_p)$  et  $Y_p$  la sortie donnée par le réseau  
On définit l'erreur qui peut être quadratique de la forme

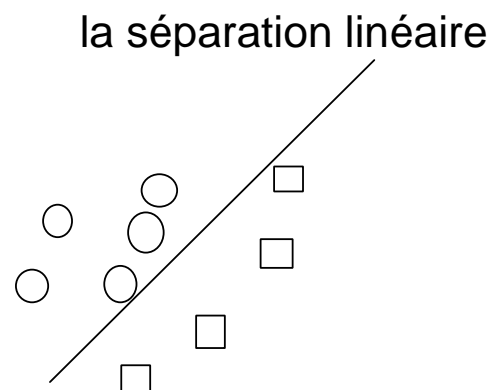
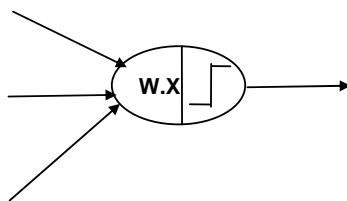
$$E(W) = \sum_p (Y_p - D_p)^2$$

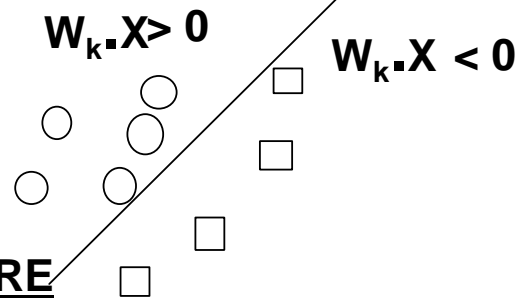
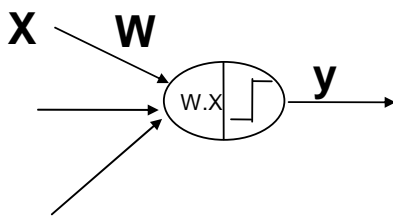
L'apprentissage revient à déterminer  $W = \text{ArgMin}(E)$ .

En général, on utilise des méthodes de gradient, total, partiel ou stochastique :

$$W(t+1) = W(t) - \lambda \cdot \text{grad}_W(E)$$

## SEPARATION LINEAIRE PAR UN SEUL NEURONE





==> SEPARATION LINEAIRE

loi d'apprentissage :

$$W_{k+1} = W_k + dX \quad \text{si } X \text{ mal classé (d : sortie désirée)}$$

$$W_{k+1} = W_k \quad \text{si } X \text{ bien classé}$$

- Convergence de l'algorithme dans le cas séparable
- Impossible de savoir si un problème est non séparable.

## REGLES DELTA

### AUTRES REGLES D'APPRENTISSAGE POUR LA SEPARATION LINEAIRE PAR UN NEURONE

- La règle Delta ou de Widrow-Hoff :

$$E(W) = \sum_p (\sigma_p - d_p)^2 \quad \text{où } \sigma: \text{ avant fonction d'activation}$$

Le gradient partiel calculé donne  $\Delta W = \lambda (d_p - W \cdot X_p) X_p$

**CONVERGE VERS SOLUTION MOINDRES CARRÉS :**

$$W = (XX^T)^{-1} Xd \quad \text{où } X: \text{ matrice } (X_1, \dots, X_n) \text{ et } d: \text{ vecteur } (d_1, \dots, d_n)$$

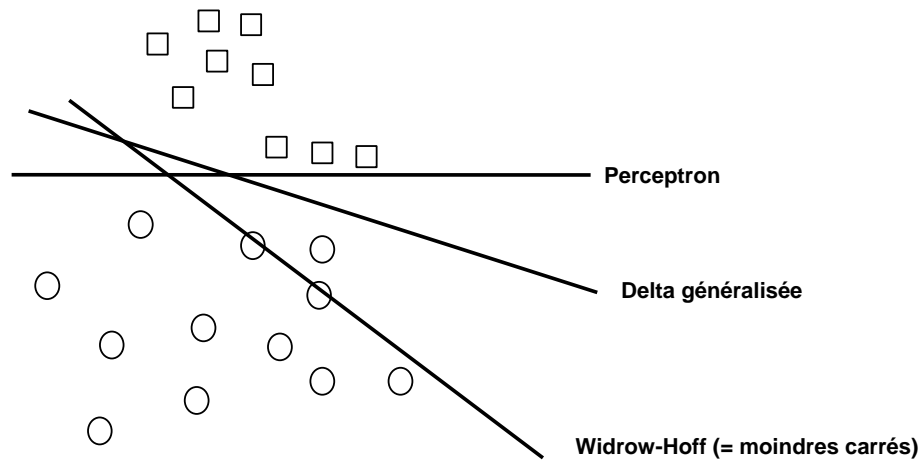
- La règle Delta généralisée :

$$E(W) = \sum_p (y_p - d_p)^2 \quad y: \text{ après fonction d'activation}$$

$$\implies \Delta W = \lambda (d_p - f(W \cdot X_p)) f'(W \cdot X_p) X_p$$

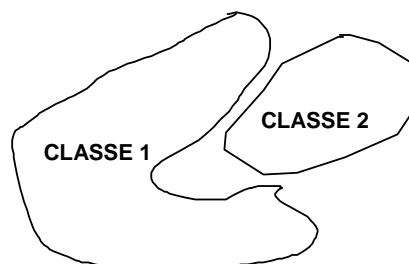
(la règle Delta est le cas où f est l'identité)

## COMPARAISON DES SOLUTIONS DES 3 ALGORITHMES



## Nécessité de neurones “cachés”

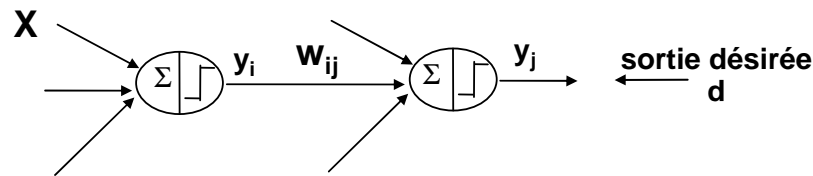
### L'INSUFFISANCE DE LA SEPARATION LINEAIRE



- Nécessité de plusieurs couches pour un réseau de neurones
- Nécessité d'un nouvel algorithme pour ces réseaux

# LE PROBLEME DE "CREDIT ASSIGNMENT"

Comment connaître la sortie désirée pour un neurone caché ?



- Problème non résolu par Minsky
- Cause de la désaffection du domaine jusqu'en 80

UTILISER DES NEURONES DERIVABLES

+  
METHODE DE GRADIENT



LA RETROPROPAGATION DU GRADIENT  
(en anglais BACKPROPAGATION)

## La rétropropagation du gradient (1)

Méthode astucieuse de calcul du gradient de la fonction de coût en utilisant la dérivation composée pour "rétropropager" l'erreur.

La fonction de coût est  $E(t) = \sum_p (Y_p - D_p)^2$   
où  $p$  parcourt l'ensemble d'apprentissage

Gradient total :  $W(t+1) = W(t) - \lambda(t) \text{grad}_W(E(t))$

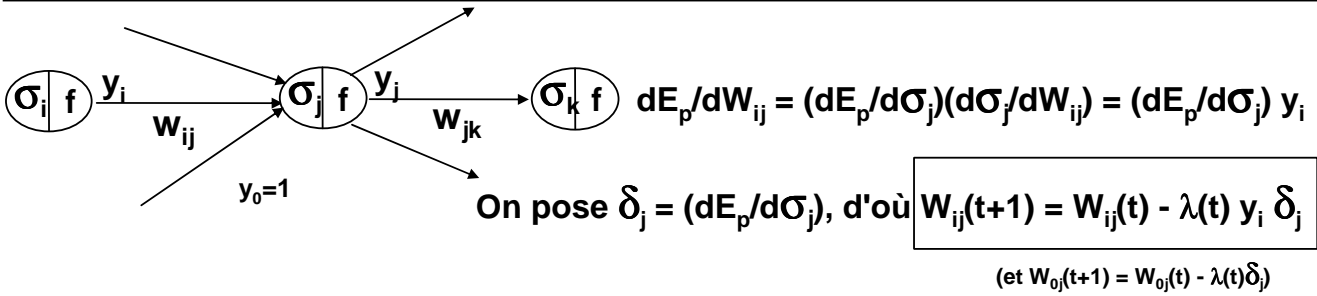
Gradient stochastique :  $W(t+1) = W(t) - \lambda(t) \text{grad}_W(E_p(t))$

où  $E_p = (Y_p - D_p)^2$ , erreur calculée sur un seul exemple que l'on tire au hasard à chaque  $t$

$\lambda(t)$  = pas de gradient (fixe, décroissant, ou adaptatif)

Reste à calculer  $dE_p/dW_{ij}$

# La rétropropagation du gradient (2)



Mais  $\delta_j = (dE_p/d\sigma_j) = \sum_k (dE_p/d\sigma_k)(d\sigma_k/d\sigma_j) = \sum_k \delta_k (d\sigma_k/d\sigma_j) = \sum_k \delta_k W_{jk} (dy_j/d\sigma_j)$

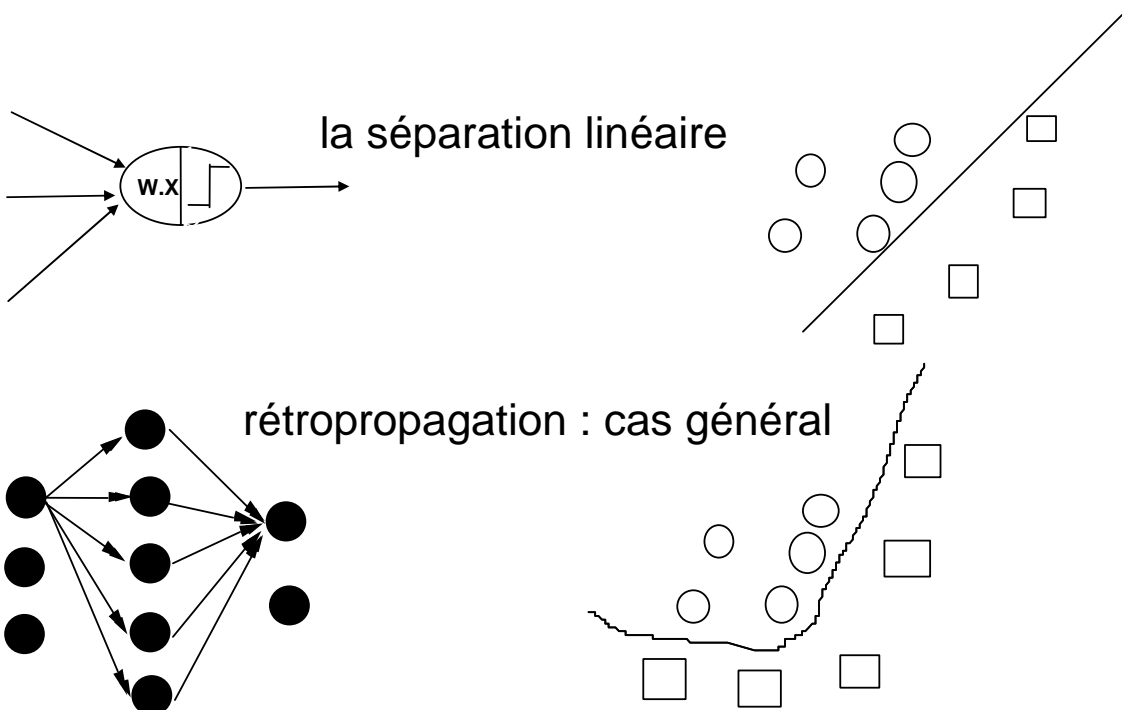
d'où  $\delta_j = (\sum_k W_{jk} \delta_k) f'(\sigma_j)$  si neurone  $j$  est caché

Et  $\delta_j = (dE_p/d\sigma_j) = (dE_p/dy_j)(dy_j/d\sigma_j)$

d'où  $\delta_j = 2(y_j - D_j) f'(\sigma_j)$  si neurone  $j$  en sortie

**→  $\delta_j$  se calcule de proche en proche par "rétropropagation de l'erreur"**

## Rétropropagation v.s. réseau monocouche



# LE THEOREME D'APPROXIMATION UNIVERSELLE

---

## Cybenko 89

- Pour toute fonction  $F$  continue définie et bornée sur un ensemble borné, et pour tout  $\varepsilon$ , il existe un réseau à 1 couche cachée de neurones sigmoïdes qui approxime  $F$  à  $\varepsilon$  près.

...Mais on ne sait pas comment le trouver à coup sûr, et ce réseau a peut-être énormément de neurones cachés...

## Sussman 92

- Les réseaux à une couche cachée forment une famille d'approximateurs parcimonieux : à nombre égal de paramètres on approxime correctement plus de fonctions qu'avec des polynômes

# Caractéristiques des réseaux neuronaux multicouches

---

## AVANTAGES

- Approximateur universel parcimonieux (et classifieur universel)
- Rapidité d'exécution (++) si multi-processeurs ou chip)
- Robustesse des solutions, résistance au bruit des données
- Facilité de développement

## INCONVENIENTS

- *Le choix de l'architecture est critique*
- Le temps d'apprentissage peut être long
- *Présence de minima locaux de la fonction de coût*
- Difficultés d'interprétation des résultats en terme de connaissance

# METHODOLOGIE POUR L'APPRENTISSAGE SUPERVISE DE RESEAUX NEURONAUX A COUCHES

## Bases d'apprentissage, de test, et de validation

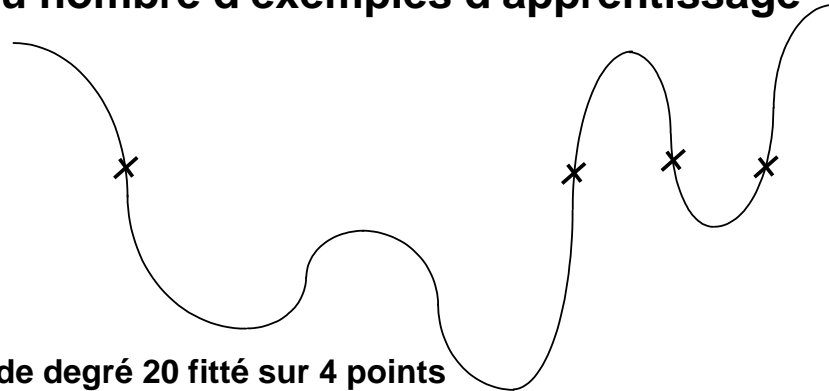
- Espace des entrées possibles infini (si valeurs continues), ensemble d'apprentissage = ss-ensemble ( $\approx$  échantillonnage)
- Erreur nulle sur tous les exemples  $\neq$  bons résultats sur tout l'espace des entrées possibles (cf erreur de généralisation  $\neq$  erreur empirique...)



- collecter suffisamment d'exemples représentatifs
- mettre de côté un ensemble d'exemples qui ne servira que de "base de test" pour estimer le taux de généralisation final (ie quand apprentissage terminé)
- séparer les exemples restants en une base d'apprentissage et une "base de validation" cette dernière servira pour arrêter l'apprentissage quand l'erreur cesse de baisser en test (éviter le "sur-apprentissage")

# METHODOLOGIE : Favoriser bonne généralisation

- éviter d'avoir trop de paramètres libres (nombre de poids) par rapport au nombre d'exemples d'apprentissage



polynôme de degré 20 fitté sur 4 points

- OU SINON, ajouter à la fonction de coût un terme de “régularisation” qui pénalisera les combinaisons de paramètres donnant un réseau “trop complexe” (typiquement, coût  $K = \text{Erreur} + \sum_{ij} W_{ij}^2 \equiv \text{“weight decay”}$ )

## Initialisation, pas de gradient, sélection de modèle

- initialiser les poids à des valeurs petites et inversement proportionnelles au nombre de connections entrant dans le même neurone (→ état initial presque linéaire)
- prendre un pas de gradient plutôt constant et petit, ou adaptatif

⇒ favorise réseaux de complexité “minimale”

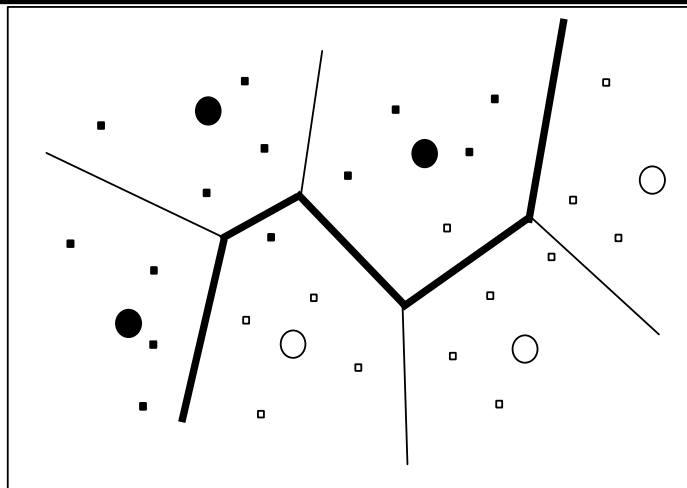
- **SELECTION DE MODELE:**  
essayer successivement différentes architectures avec un nombre croissant de neurones cachés (ou bien adopter un algorithme "constructif")

- **importance des prétraitements de entrées : en classification, extraction de paramètres discriminants pour diminuer la dimension des vecteurs d'entrée**
- **normaliser les valeurs de sorties désirées des exemples selon la fonction d'activation des neurones de sortie (a priori entre 0 et +1 ou entre -1 et +1)**
- **normaliser les valeurs d'entrée des exemples (typiquement entre -1 et +1)**
- **pour classification, choisir de préférence le codage "grand-mère" :**  
1 neurone par classe, et classe 1 = (1,0,0,...,0), classe 2 = (0,1,0,...,0),  
...
- **diviser pour régner : en classification, essayer de scinder le pb en sous-problèmes plus simples**

- **Travaux très récents montre grand intérêt de initialisation NON-aléatoire des poids, par algo non-supervisé type « machine de Boltzman »**

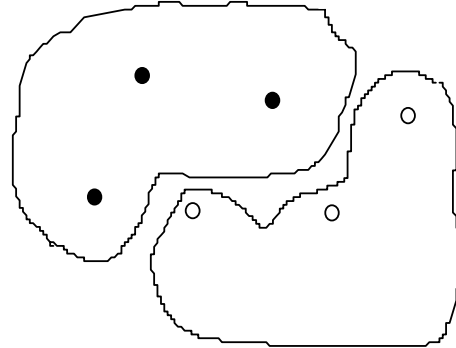
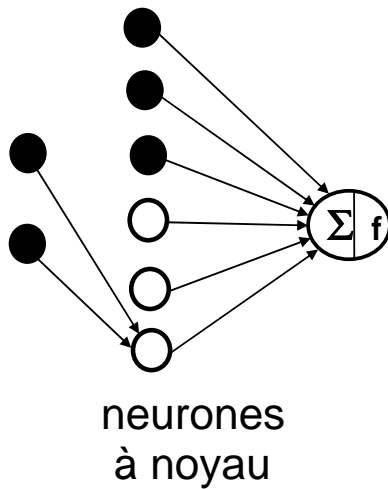
# AUTRES TECHNIQUES NEURONALES SUPERVISEES

## RESEAU LVQ



- On associe plusieurs références à chaque classe
- Les frontières de classification sont déterminées par le diagramme de Voronoï
- L'apprentissage déplace les références pour optimiser la frontière

## METHODES A NOYAUX : (RBF, Potentiels, ondelettes, SVMs)



# L'APPRENTISSAGE NON SUPERVISE

## Motivation

- Pour “organiser”/analyser/catégoriser un grand volume de données inexploitable tel quel
- Indispensable en classification lorsqu'on possède beaucoup d'exemples dont une trop faible proportion est labellée.

- **K-means**
- **cartes topologiques de Kohonen  
(Self-Organizing Feature Maps, SOFM)**
- **Adaptive Resonance Theory (ART)**

## LES CARTES TOPOLOGIQUES (1)

---

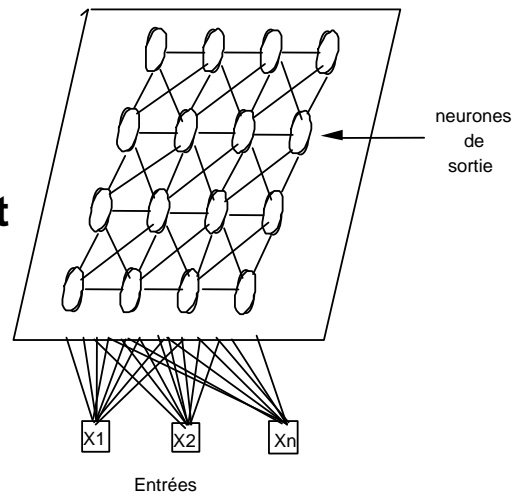
**L'inspiration initiale est biologique :  
auto-organisation des régions du système nerveux.**

### MOTIVATIONS EN CLASSIFICATION

- **Faire du clustering, i.e. regrouper les exemples en paquets "similaires" pour définir des classes**
- **Construire une représentation visualisable (1D ou 2D en général) des entrées par une sorte de "projection non-linéaire" de l'espace des entrées (en général de grande dimension) qui respecte la topologie initiale (les "projections" de points proches restent proches).**

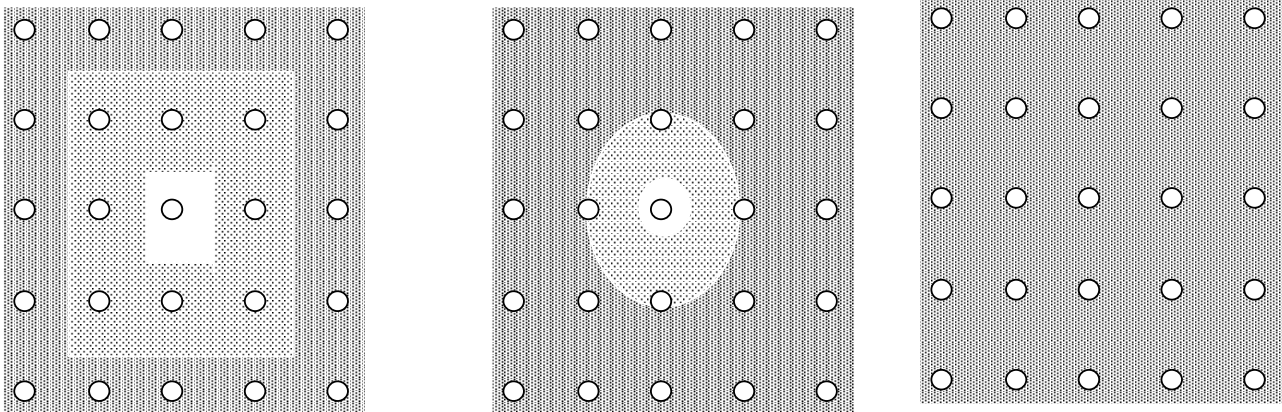
## L'ALGORITHME DE KOHONEN

- neurones distance en sortie
- connexion totale
- neurone de sortie  $\in$  espace départ
- système de voisinages sur la carte de sortie
- "projection non linéaire"



# LES CARTES TOPOLOGIQUES (3)

- définition de voisinages sur la carte :



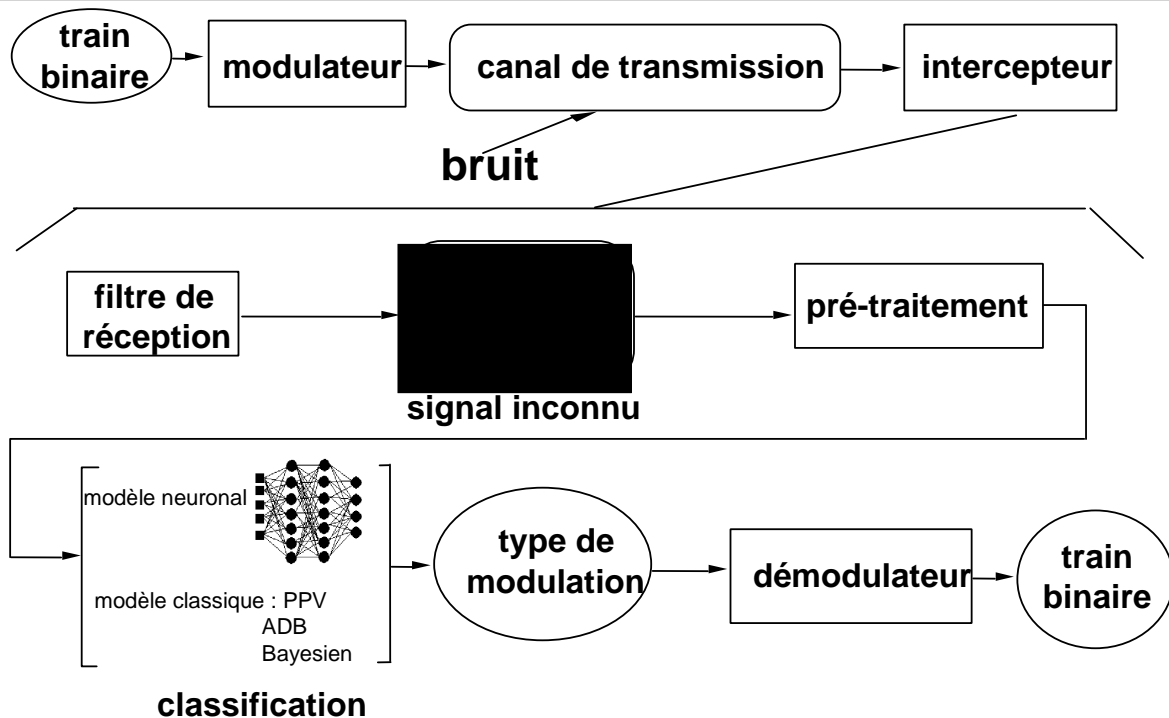
$V_i(t)$  voisinage décroissant avec  $t$

- La réponse d'une cellule  $i$  de poids  $W_i = (w_{i1}, \dots, w_{in})$  à une forme  $X = (x_1, \dots, x_n)$  est la distance euclidienne de  $X$  à  $W_i$ .
- l'apprentissage :
  - repérer la cellule la plus active (la plus proche)
  - essayer de la rendre de plus en plus active
  - EN MEME TEMPS QUE SON VOISINAGE.**
- 2 paramètres : la taille du voisinage (rayon)
  - le pas  $\alpha(t)$  de la modification des poids qui diminuent avec le temps

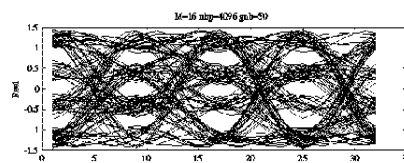
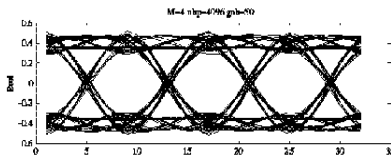
- $t=0$ , initialiser les poids (hasard ?)
- date  $t$ , présenter l'exemple  $X$ 
  - déterminer le neurone de poids le plus proche
  - déterminer le voisinage  $V_i(t)$  et le pas  $\alpha(t)$
  - modifier les poids :
$$W_j(t+1) = W_j(t) + \alpha(t) (X - W_j(t)) \quad \text{si } j \in V_i(t)$$
$$W_j(t+1) = W_j(t) \quad \text{sinon.}$$
- $t = t+1$
- Convergence de l'algorithme :
  - conditions sur  $\alpha(t)$  ( $1/t$  convient)
- Amélioration de la gaussienne
  - (réduction au seul paramètre  $\alpha$ )

## RECONNAISSANCE DE MODULATION

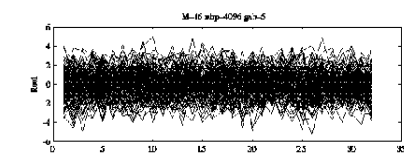
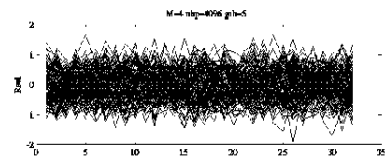
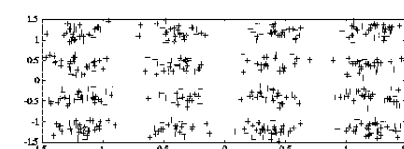
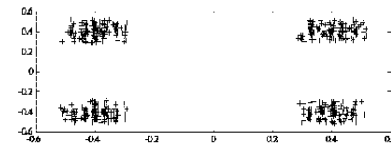
## RECONNAISSANCE DE MODULATION (1)



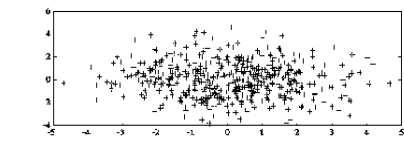
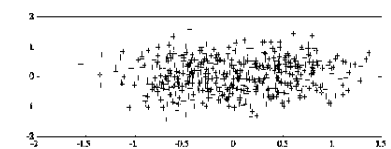
# RECONNAISSANCE DE MODULATION (2)



- fort S/N  
reconnaissance visuelle possible



- faible S/N  
reconnaissance visuelle impossible



- Nécessité d'un classifieur

# RECONNAISSANCE DE MODULATION (3)

## MODULATIONS A RECONNAÎTRE

- FSK2, FSK4, FSK8
- MDP2, MDP4, MDP8     10 types de modulation
- MAQ 16, MAQ64
- OQPSK, MSK
- DONNEES BRUTEES (canal de transmission) :  
rapport signal/bruit en dB/bit de 0 à 50 dB
- Filtre d'interception (bande réaliste)
- 2048 échantillons temporels pour chaque exemple

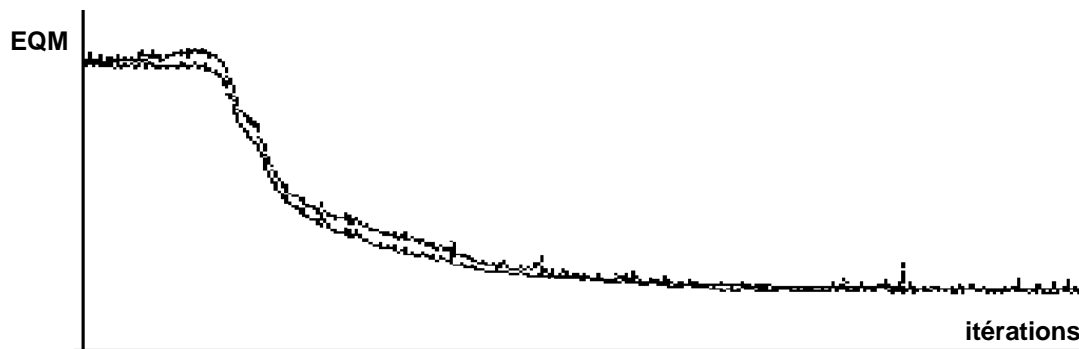
## • CHOIX DES ECHANTILLONS

- 1 générateur
- 3000 exemples, équitablement répartis entre les classes de modulation (300 chacune) et les niveaux de signal/bruit
- 11 paramètres discriminants (moments statistiques) calculés sur les exemples (==> espace entrée classifieur dimension 11 au lieu 2048)
- apprentissage = 1500 exemples, validation = 1500 exemples

## RECONNAISSANCE DE MODULATION (4)

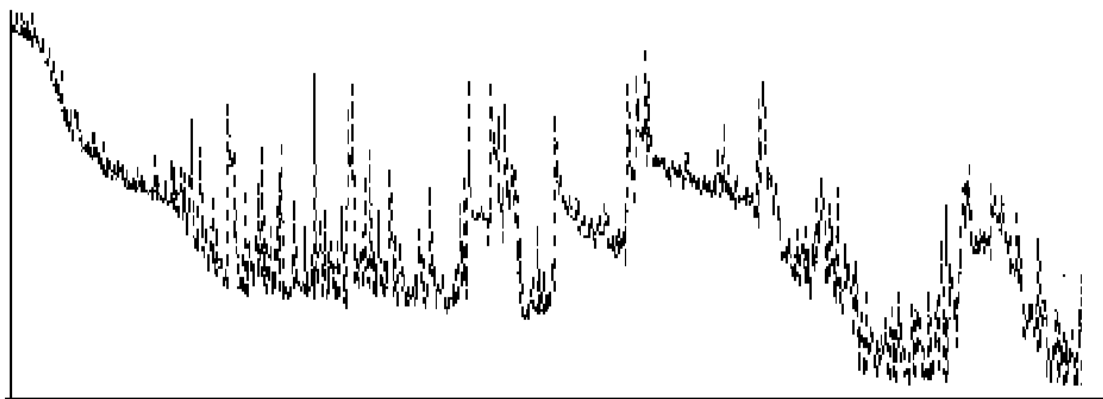
- regroupement des modulations en 3 "méta-classes" naturelles
  - FSK2, FSK4, FSK8, OQPSK, MSK
  - PSK2, PSK4, PSK8 ==> 4 problèmes plus simples
  - QAM 16, QAM 64

- courbe d'apprentissage typique



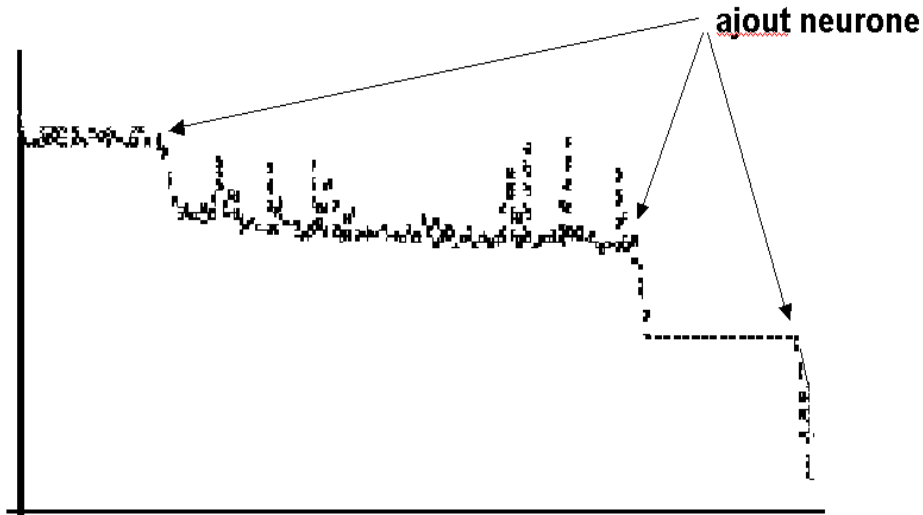
## RECONNAISSANCE DE MODULATION (5)

- cas d'un apprentissage avec pas de gradient un peu trop grand

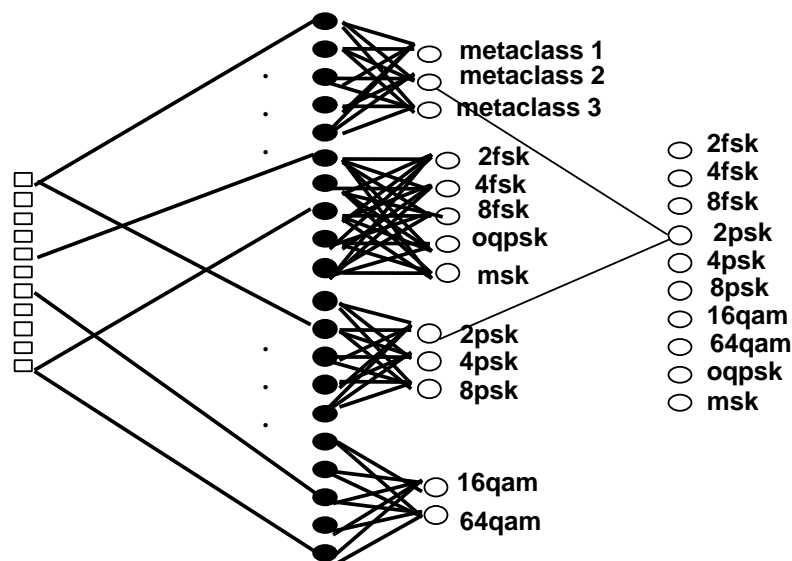


# RECONNAISSANCE DE MODULATION (6)

- cas d'un apprentissage constructif (Adaptive BackPropagation)



# RECONNAISSANCE DE MODULATION (7)

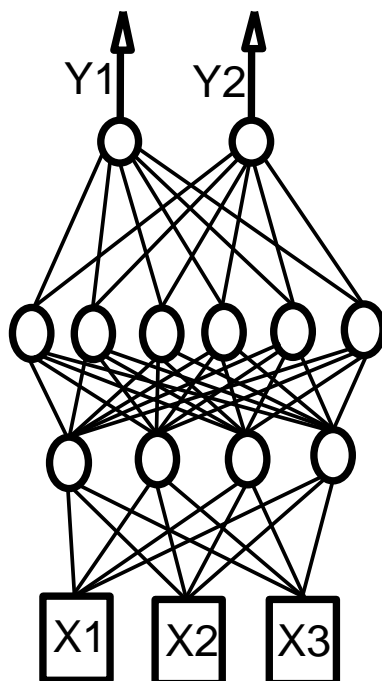


- 0-50 dB  $\rightarrow$  90% de bon classement (5-50 dB  $\rightarrow$  95%)
- erreurs < 2% si le classifieur rejette les cas litigieux ( $\rightarrow$  81% de bon classement et 17% de rejet)

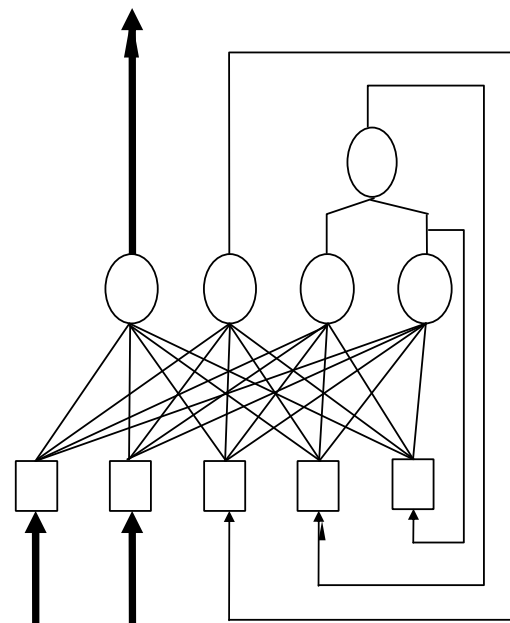
- 1) Les réseaux bouclés (ou récurrents)
- 2) L'apprentissage des réseaux récurrents
- 3) Identification par réseaux de neurones
- 4) Commande neuronale
- 5) Applications et exemples

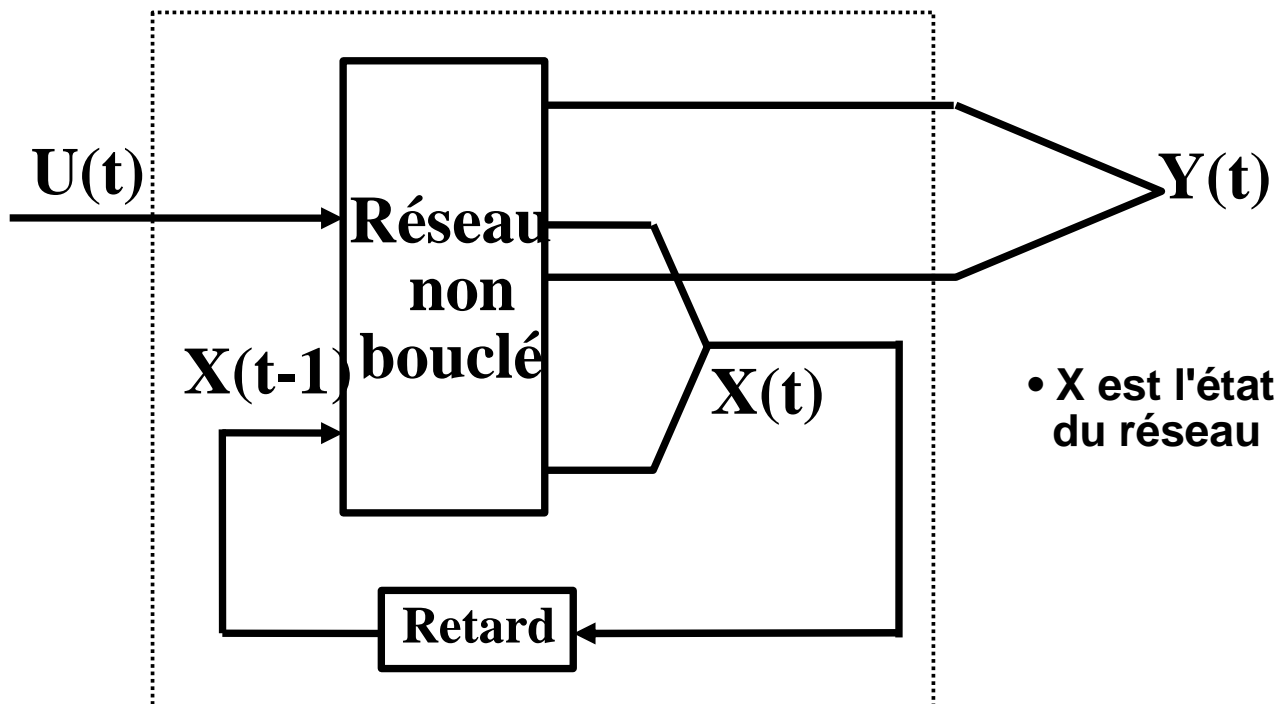
## LES RESEAUX BOUCLES

RESEAU STATIQUE



RESEAU BOUCLE  
(ou RECURRENT)





## ETAT ET ORDRE D'UN RESEAU BOUCLE

- Pour calculer la sortie à l'instant  $t$ , on doit connaître :
  - les valeurs d'entrée du réseau à  $t$
  - un nombre minimum de sorties de neurones à  $t-1$
- Ces sorties sont appelées **VARIABLES D'ETAT** (pas uniques)
- Leur nombre est l'**ORDRE DU RESEAU**
- L'ensemble des variables d'état est l'**ETAT DU RESEAU**

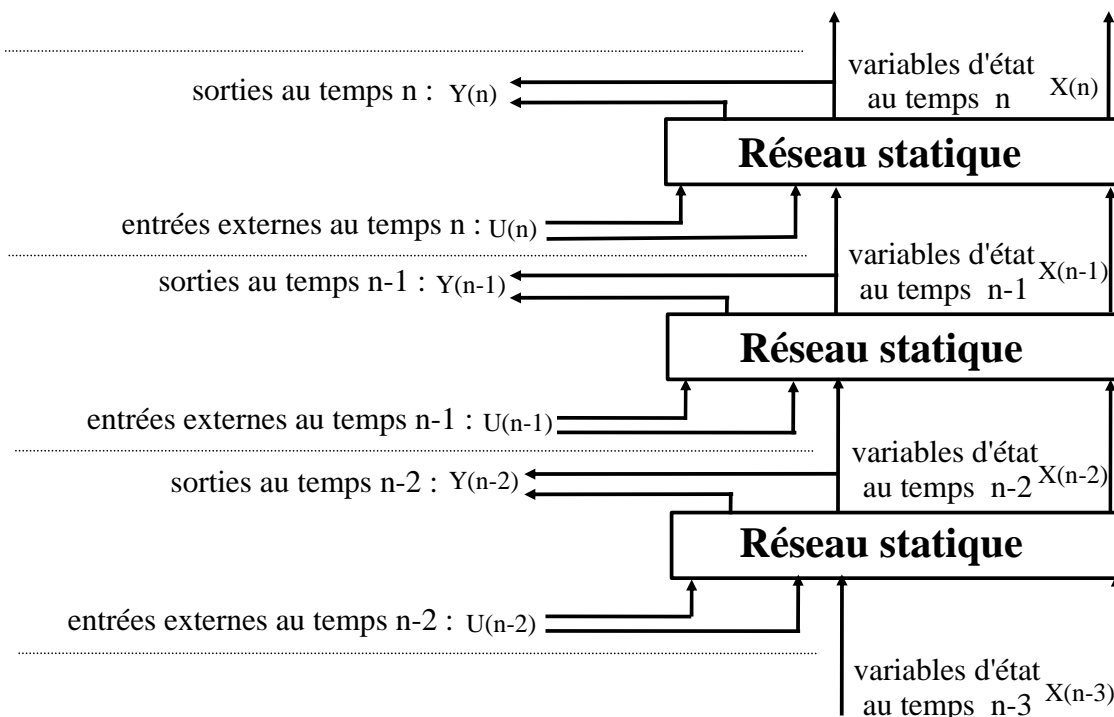
- Un réseau bouclé à temps discret d'ordre  $K$  peut être représenté par  $K$  équations aux différences, couplées, d'ordre 1, mettant en jeu  $K$  variables (état) en plus des  $N$  variables d'entrée.
- Soit le vecteur d'entrée  $U(t) = [u_1(t), \dots, u_N(t)]$  et le vecteur d'état  $X(t) = [x_1(t), \dots, x_K(t)]$

Comportement dynamique du réseau :

$$X(t) = \phi[U(t), X(t-1)]$$

$$Y(t) = \pi[U(t), X(t-1)]$$

## FONCTIONNEMENT DES RESEAUX BOUCLES



L'apprentissage des réseaux bouclés  
se fait sur des séquences d'apprentissage

Plusieurs techniques d'apprentissage proposées :

- Rétropropagation récurrente
- Rétropropagation dans le temps (BPTT)
- Apprentissage temporel récurrent en temps réel (Real-Time Recurrent Learning, RTRL)

différences = ce que ça fait, ou la façon de calculer, voire simplement le nom de l'auteur...

## PARAMETRES SPECIFIQUES : DEFINITIONS

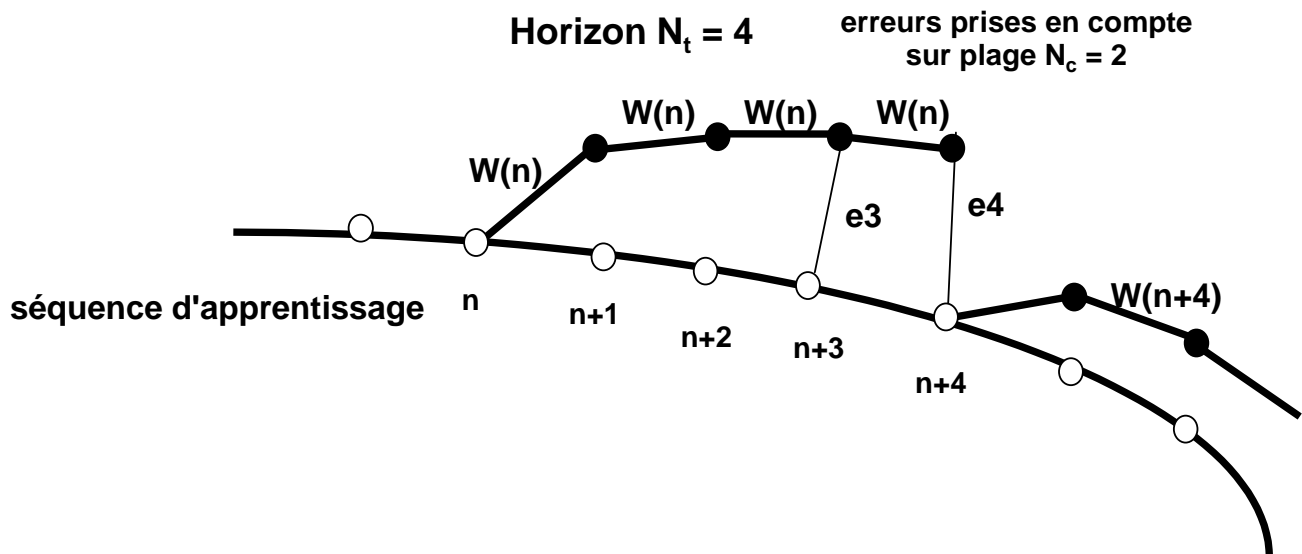
---

• **Horizon :**  
nombre  $N_t$  d'itérations entre 2 modifications des poids  
(donc nb de "copies" du réseau statique mises bout à bout)

- **Fonction de coût** calculée sur une plage  $N_c \leq N_t$
- **Fréquence de recalage** du vecteur d'état  
sur des valeurs désirées

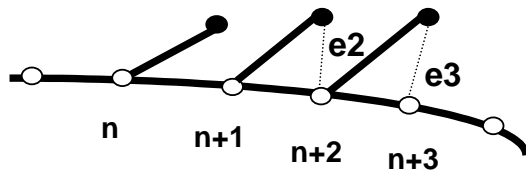


**Ce dernier paramètre détermine  
plusieurs grands types d'algorithmes**

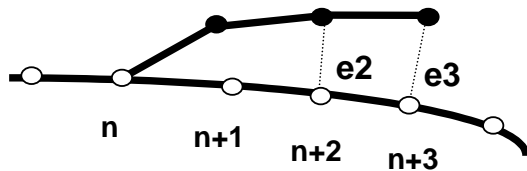


## CATEGORIES D'ALGORITHMES

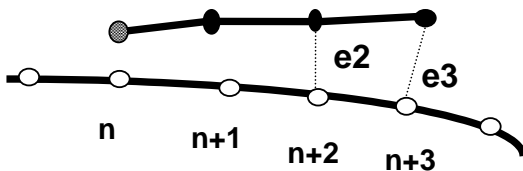
- **ALGORITHMES DIRIGES :**  
état entrant dans chaque bloc = valeurs désirées.
- **ALGORITHMES SEMI-DIRIGES :**  
état entrant dans premier bloc de l'horizon = valeurs désirées,  
état entrant dans les suivants = état sortant du précédent.
- **ALGORITHMES NON DIRIGES :**  
l'état n'est jamais recalé sur des valeurs désirées.



algorithme dirigé



algorithme semi-dirigé



algorithme non dirigé

(Horizon  $N_t = 3$ , erreurs prises en compte sur plage  $N_c = 2$ )

- Si certaines variables d'état n'ont pas de valeurs désirées contenues dans la base d'apprentissage (cas des états purement internes), alors on ne peut faire que du non dirigé.
- Plus l'algorithme est dirigé, plus il est stable, avec erreur faible en apprentissage. Mais le comportement dynamique n'est pas assuré en généralisation.
- Un algorithme non dirigé est difficile à apprendre. Il offre plus de garantie en dynamique.

$$W(t+N_t) = W(t) - \lambda \text{grad}_W(E) \text{ avec } E = \sum_{\tau} (Y_{\tau} - D_{\tau})^2$$

$$\begin{pmatrix} y_n \\ x_n \end{pmatrix} = F \begin{pmatrix} u_n \\ x_{n-1} \end{pmatrix} \quad \begin{aligned} \frac{dy}{dw} &= \frac{\partial F_y}{\partial w} + \frac{\partial F_y}{\partial x} \cdot \frac{dx_{n-1}}{dw} \\ \frac{dx_n}{dw} &= \frac{\partial F_x}{\partial w} + \frac{\partial F_x}{\partial x} \cdot \frac{dx_{n-1}}{dw} \end{aligned}$$

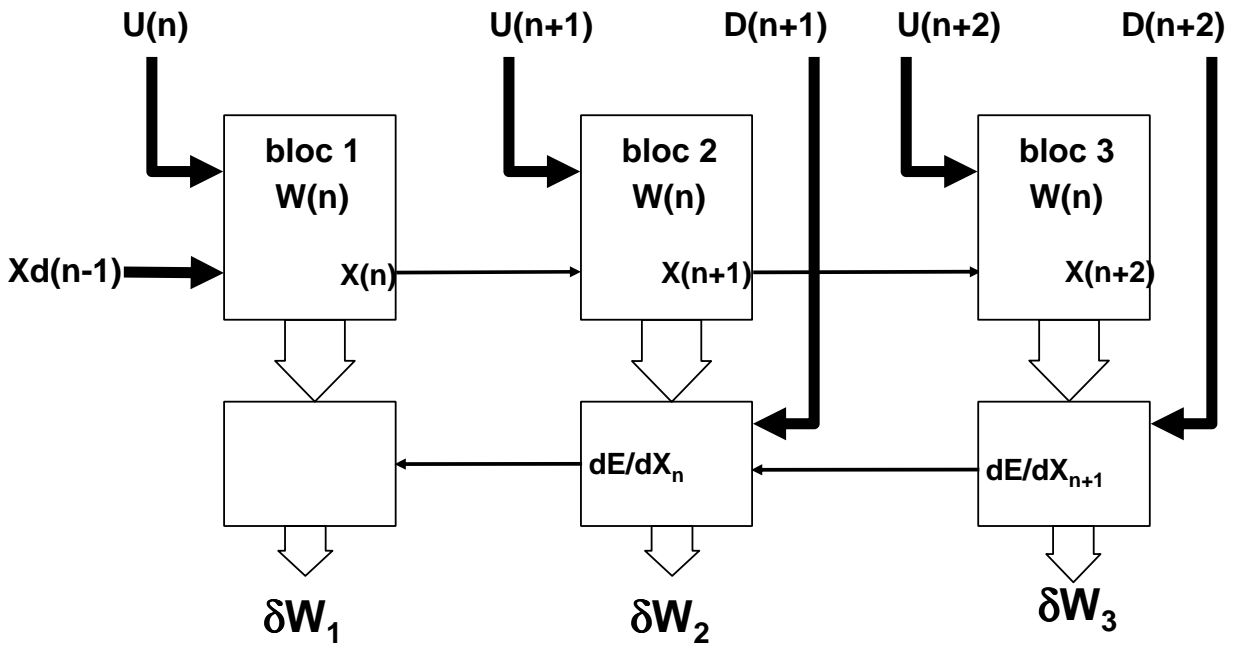
Si dirigé,  $\frac{dx_{n-1}}{dw} = 0 \implies$  formule usuelle, calculable par rétropropagation

Si semi-dirigé,  $\frac{dx_{n-1}}{dw} = 0$  au début de chaque horizon  $\implies$  on peut rétropropager à travers  $N_t$  répliques successives du réseau statique

Si non dirigé, il faut faire une propagation avant du gradient (mais on peut quand même calculer la Jacobienne du réseau par technique type rétropropagation, avec une passe par sortie)

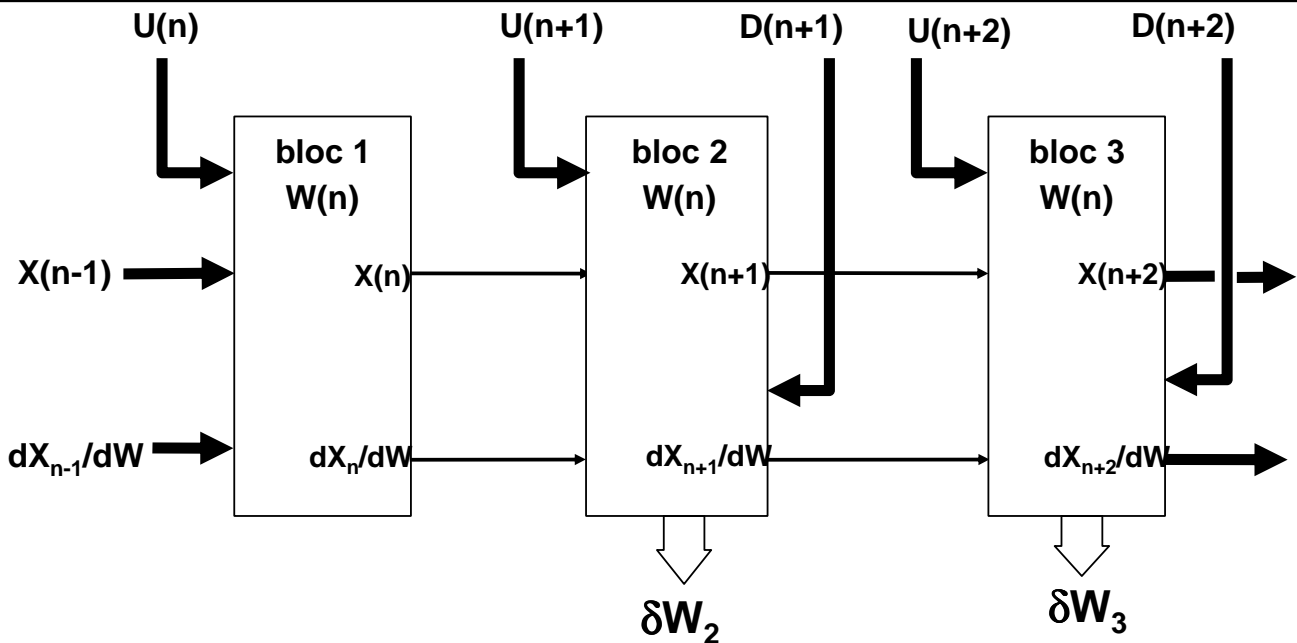
- Pour algo dirigé, BP à chaque pas pour lequel on prend erreur en compte, mais modification des poids que tous les  $N_t$  pas.
- démultiplication du réseau statique + BP globale  $\implies$  calculs rapides, MAIS :
  - pb de mémoire si horizon grand,
  - ne convient pas pour algo non dirigé
- analyse des équations et reformulation matricielle globale  $\implies$  compliqué
- propagation avant

# RETROPROPAGATION SUR UN HORIZON



$$\delta W = \delta W_1 + \delta W_2 + \delta W_3$$

# PROPAGATION AVANT



Le calcul du  $\delta W$  et du  $dX_{k+1}/dW$  dans chaque bloc peut se faire :

- soit par calcul direct,
- soit par plusieurs rétropropagations

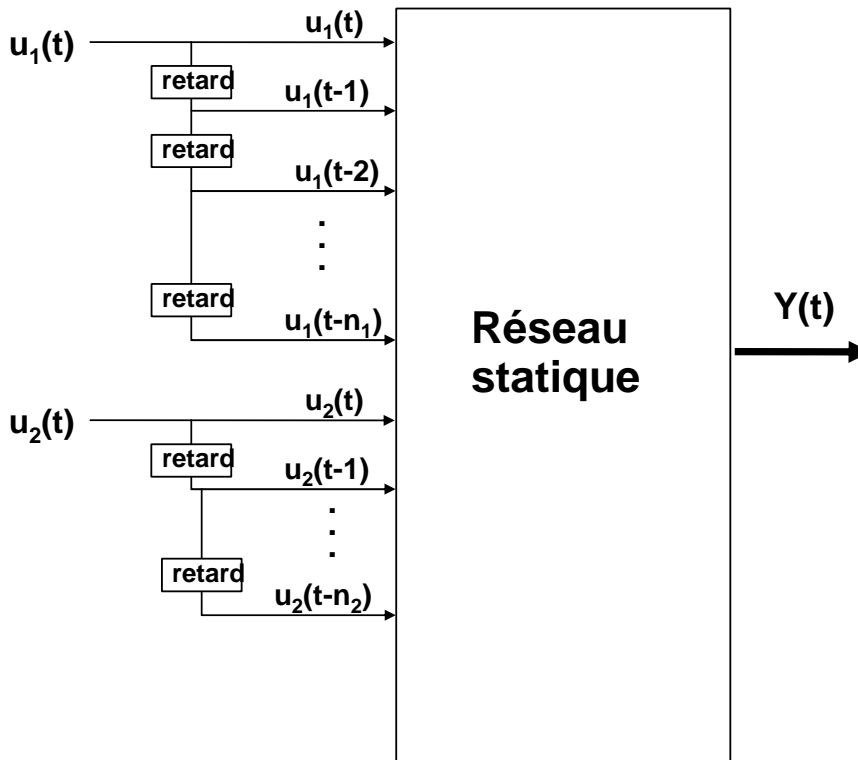
## GENERALITES SUR L'IDENTIFICATION

- **Identification = obtention d'un modèle mathématique pour représenter le comportement dynamique d'un processus.**
- **Types de modèle :**
  - **parfois, modèle de connaissances (équations différentielles) mais 1) souvent trop idéalisé par rapport au système réel, 2) difficulté à connaître la valeur des paramètres physiques, et 3) lourdeur des calculs pour résolution des equa. diff.**
  - **le plus souvent, modèle mathématique simple (en général linéaire) avec des paramètres sans signification physique : c'est l'identification "boîte noire".**

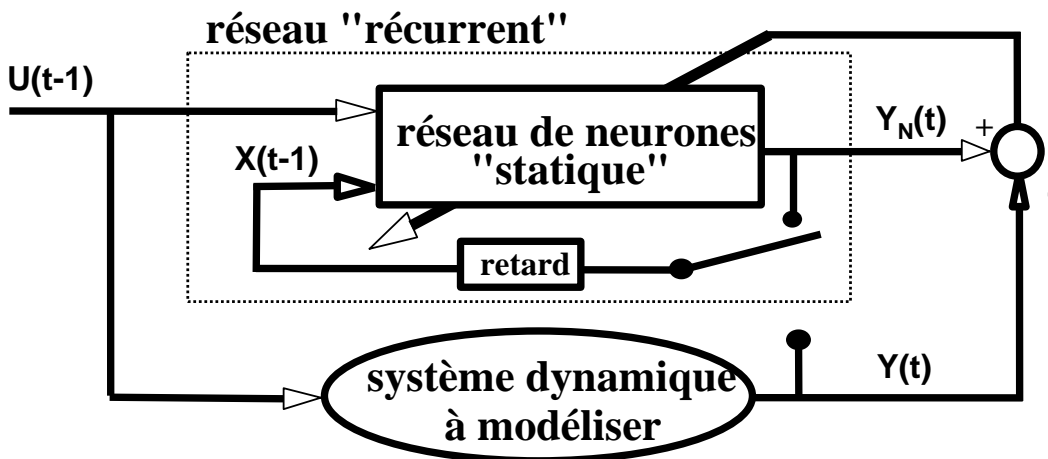
## INTERET DU NEURONAL EN IDENTIFICATION ?

---

- **propriété d'approximateurs universels parcimonieux ==> modèles NON-LINEAIRES quelconques**
- **rapidité de calcul en exécution**
- **capacité naturelle à réaliser un modèle adaptatif**



## TYPES D'IDENTIFICATION NEURONALE



- série-parallèle (equation error, NARX) = algorithme dirigé (valeur état prise sur système à modéliser)
- parallèle (output error) = algorithme non dirigé (réseau rebouclé pendant l'apprentissage)

- **equation error (ARX)**

$$y_m(k) = a_1 y(k-1) + a_2 y(k-2) + \dots + a_n y(k-n) + b_0 u(k) + b_1 u(k-1) + \dots + b_n u(k-n)$$

→ c'est un neurone linéaire...

- **output error**

$$y_m(k) = a_1 y_m(k-1) + \dots + a_n y_m(k-n) + b_0 u(k) + b_1 u(k-1) + \dots + b_n u(k-n)$$

→ réseau récurrent avec sortie rebouclée n fois, et 1 neurone linéaire.

- **Si on connaît certaines équations explicites du système, s'en servir soit simplement pour contraindre l'architecture du réseau, soit comme une partie du modèle**
- **On peut utiliser des neurones "ad hoc" (sin, cos, ...)**

# CONSEILS PRATIQUES POUR L'IDENTIFICATION NEURONALE

---

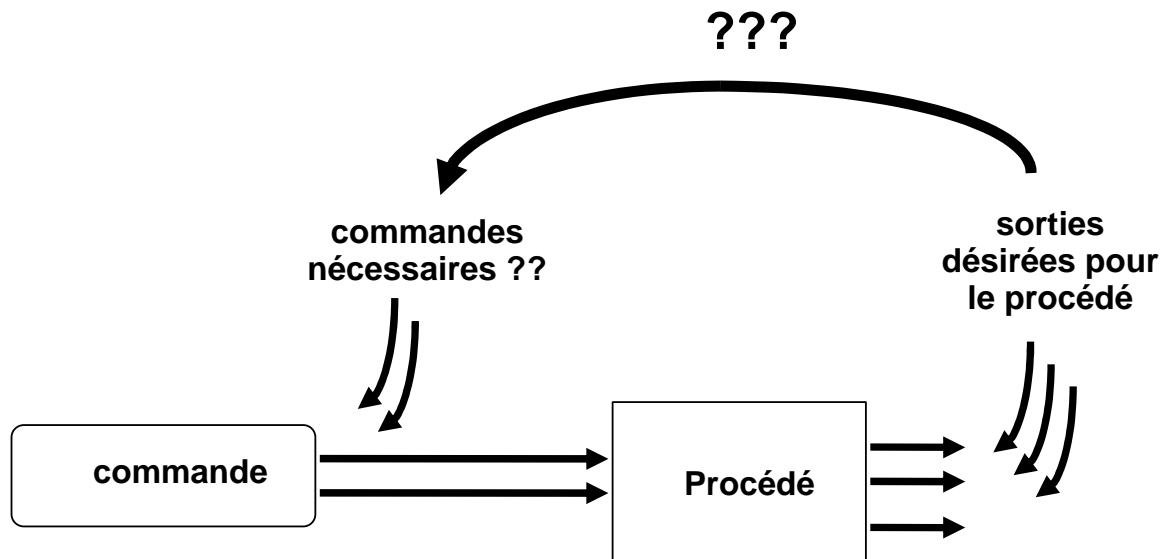
- non-linéaire ==> pas de théorème de superposition  
==> les séquences d'apprentissage doivent bien couvrir  
le domaine de fonctionnement
- si possible faire d'abord identification linéaire avec  
laquelle on initialise le réseau, ou bien à laquelle la  
contribution du réseau est ajoutée
- Pour de la prédiction à court terme, préférer filtre  
transverse
- Pour modèle dynamique hors ligne, préférer réseau  
récurrent + apprentissage parallèle

# COMMANDE NEURONALE

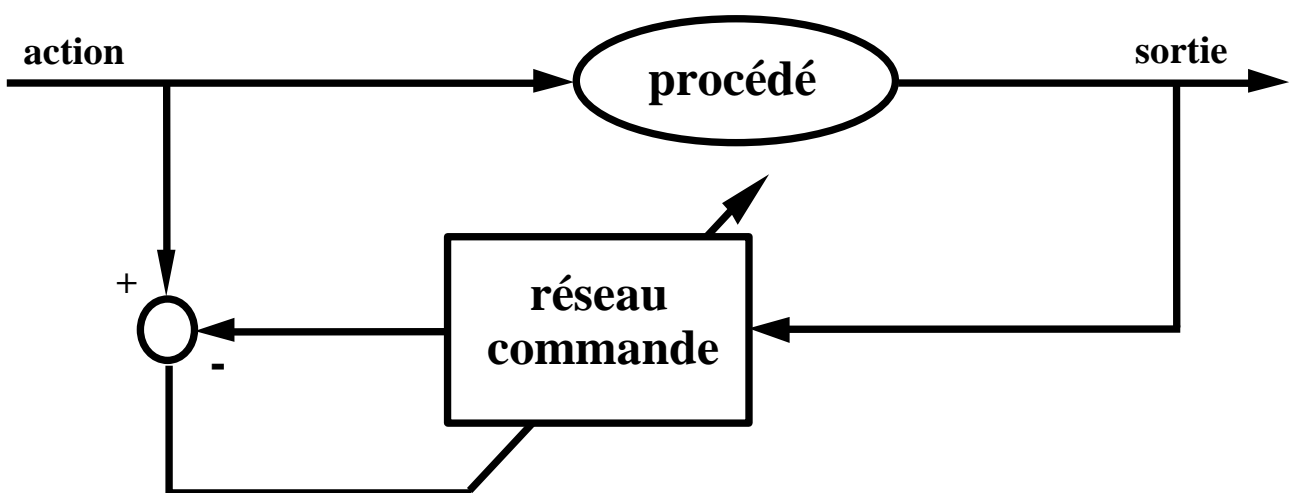
---

- **Commande = élaboration de signaux d'entrée d'un processus qui permette d'obtenir un comportement spécifié d'avance**  
(régulation : maintenir autour pt de fonctionnement)
- **Intérêts essentiels des réseaux neuronaux :**
  - 1) **capacité à synthétiser des modèles inverses non-linéaires**
  - 2) **capacité naturelle à réaliser de la commande adaptative**

# OU L'ON RETROUVE LE "CREDIT ASSIGNMENT"

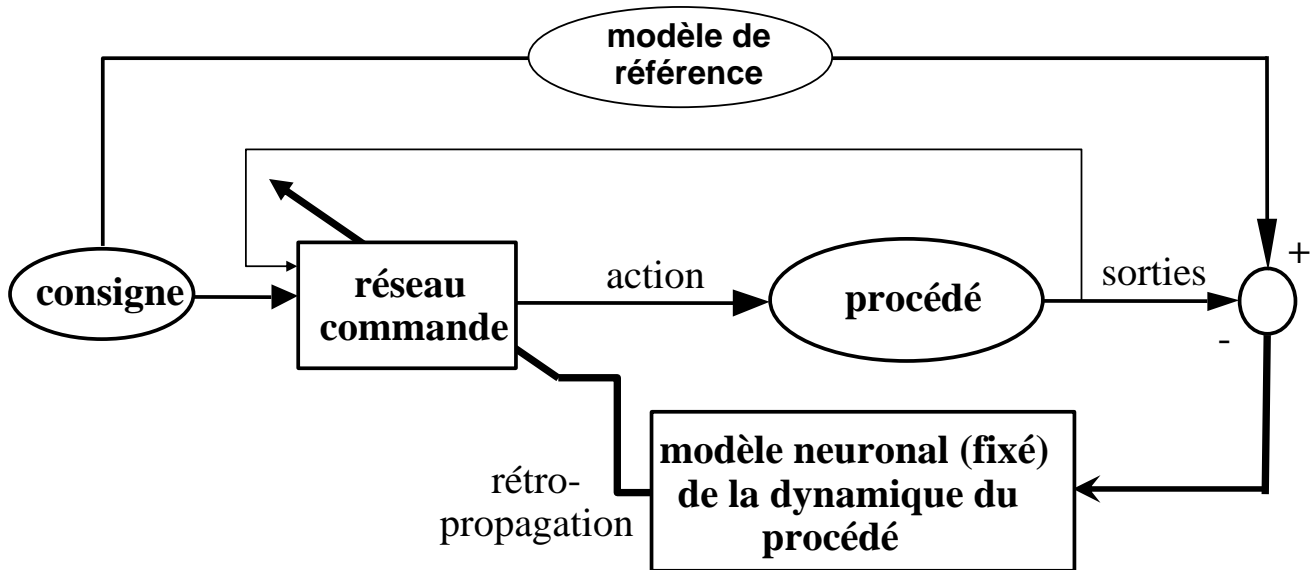


# APPRENTISSAGE DIRECT DE MODELE INVERSE



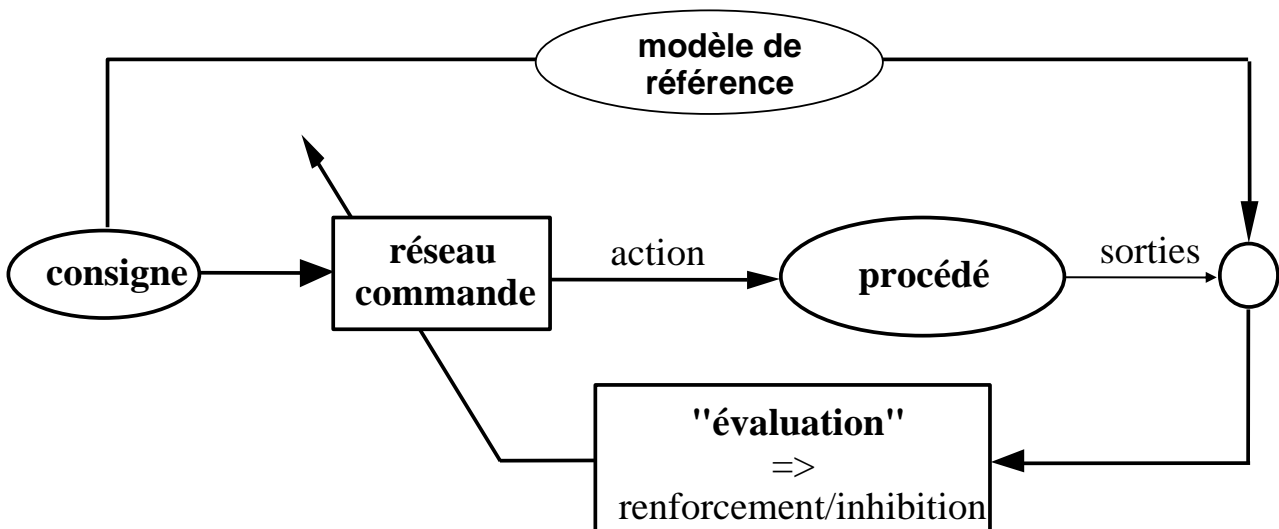
- Problème si plusieurs actions peuvent donner la même sortie...

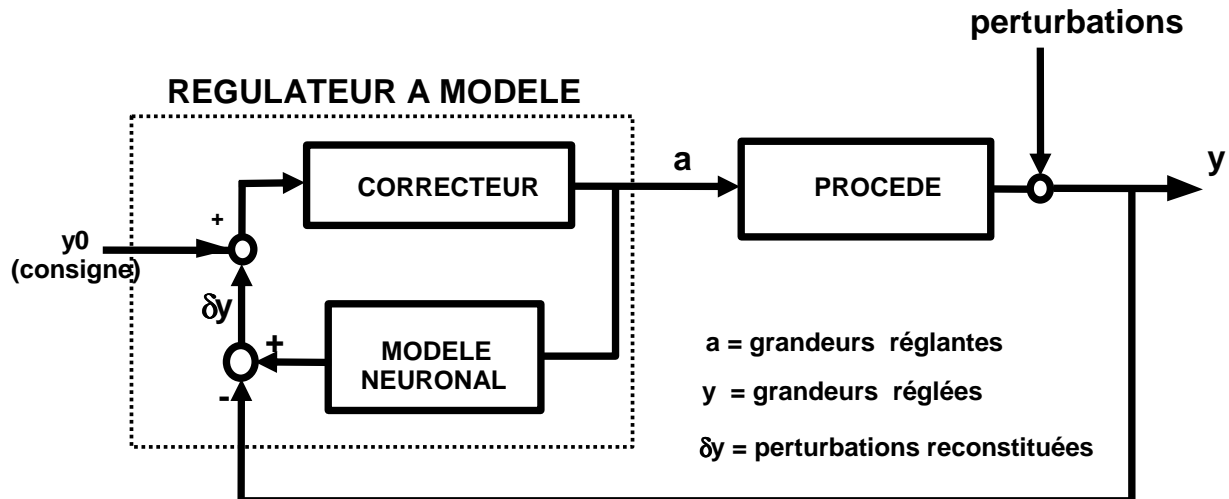
# APPRENTISSAGE INDIRECT DE MODELE INVERSE



Technique la plus couramment utilisée

# APPRENTISSAGE PAR RENFORCEMENT





- Modèle neuronal dans un "régulateur à modèle" classique

- Intérêt de travailler d'abord sur un modèle du procédé, et de ne faire que recalage/raffinement sur le vrai système...
- Comme pour l'identification non-linéaire, nécessité de séquences d'apprentissage couvrant bien le domaine de fonctionnement
- Difficultés liées à l'absence de preuve de stabilité (comme toute commande non-linéaire)

## DOMAINES DE PREDILECTION

- Prédiction empirique (consommation d'eau, d'électricité, trafic routier, nb de passagers, ...)
- modélisation de procédés physico-chimiques, thermiques, voire biologiques (car équations complexes, difficiles à résoudre, voire inconnues)
- commande non-linéaire de système mécanique (prise en compte saturations, frottements, ...)

# APPLICATIONS

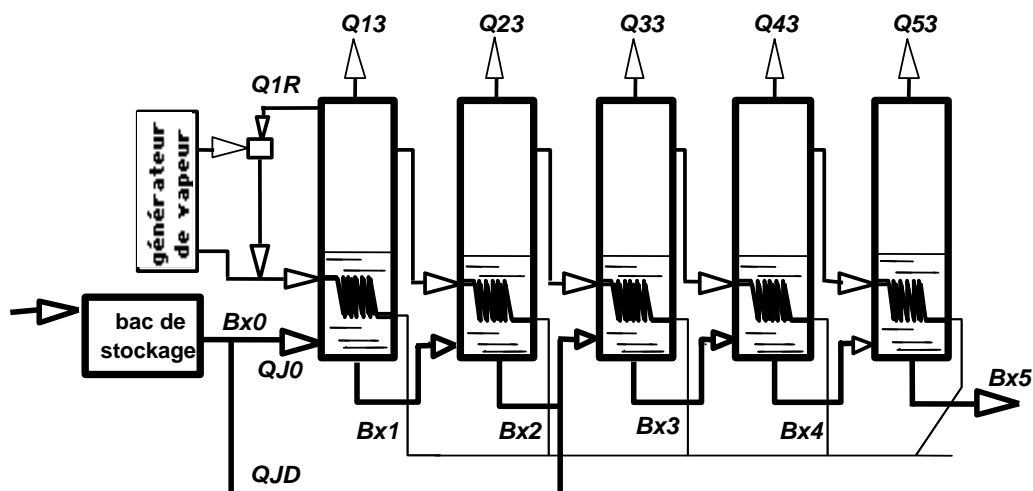
---

- prévision de consommation d'eau (Lyonnaise, CGE), d'électricité (EDF), de trafic routier (Cofiroute), d'affluence de voyageurs (SNCF), de cours boursiers...
- identification de procédé industriel (Air liquide, Elf Atochem, ciments Lafarge...), modèle de dynamique thermique de bâtiment, ...
- commande de véhicule (Sagem), de robot autonome, de station de pompage (Mitsubishi)

- Identification boite noire
- Identification hybride
- Commande

## EXEMPLE D'IDENTIFICATION "BOITE NOIRE"

**Evaporateur de sucrerie**  
(étude réalisée pour Cegelec par Alcatel-Alsthom Recherche)



## 3 entrées externes :

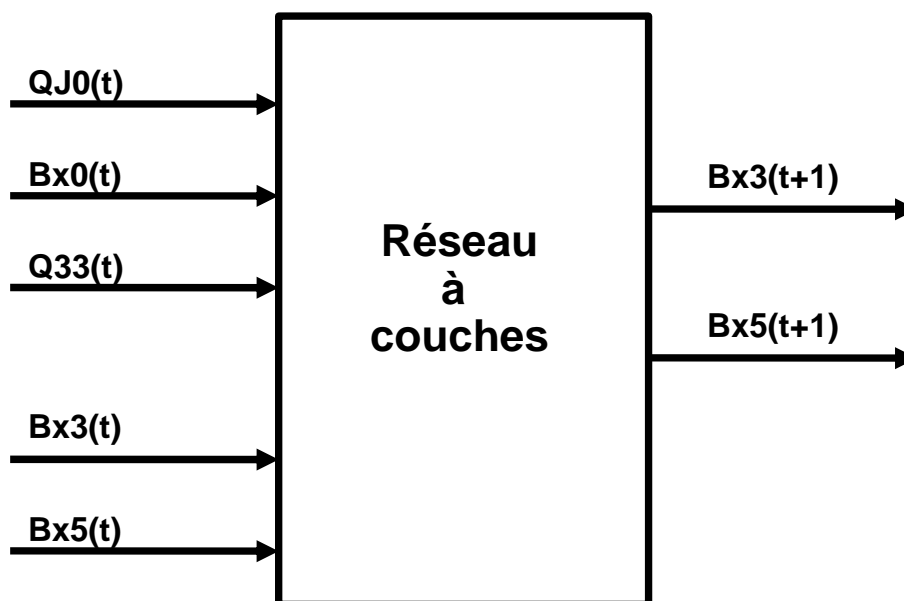
- QJ0 (commande)
- Bx0, Q33 (perturbations mesurées)

## 2 sorties :

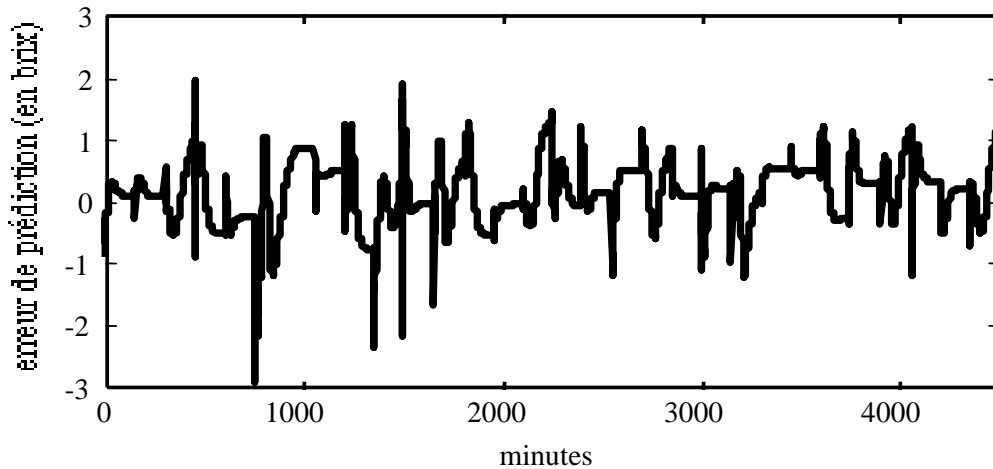
- Bx3 et Bx5  
(concentrations effectivement mesurées)

## séquence d'apprentissage :

série de 30 échelons 3D aléatoires sur les entrées  
150 pas de temps entre 2 échelons

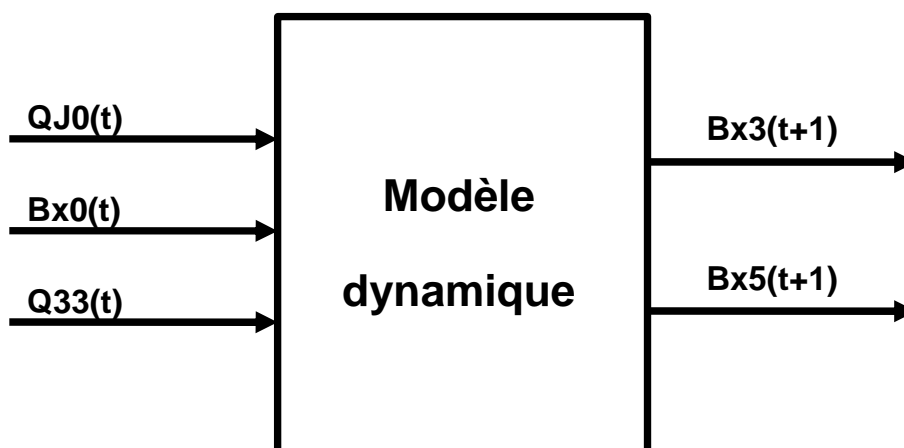


## Prédiction à 1 pas de temps en avance



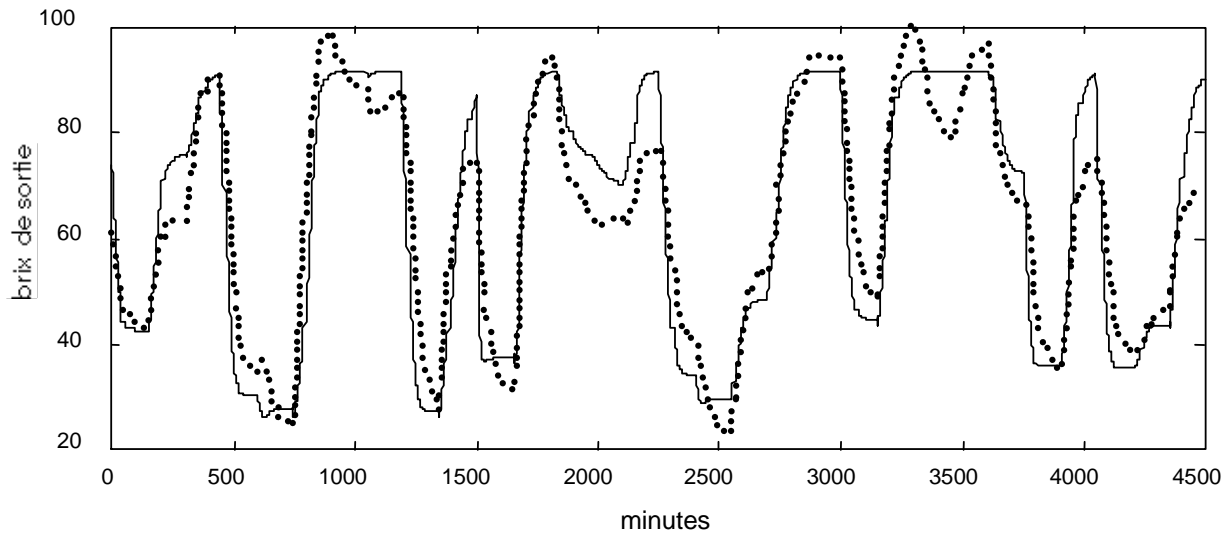
- **erreur moyenne - 0,8 % (base de test)**
  - réseau 5-15-2
  - (ordre 1 sur les sorties précédentes et les entrées)

## MODELE PARALLELE



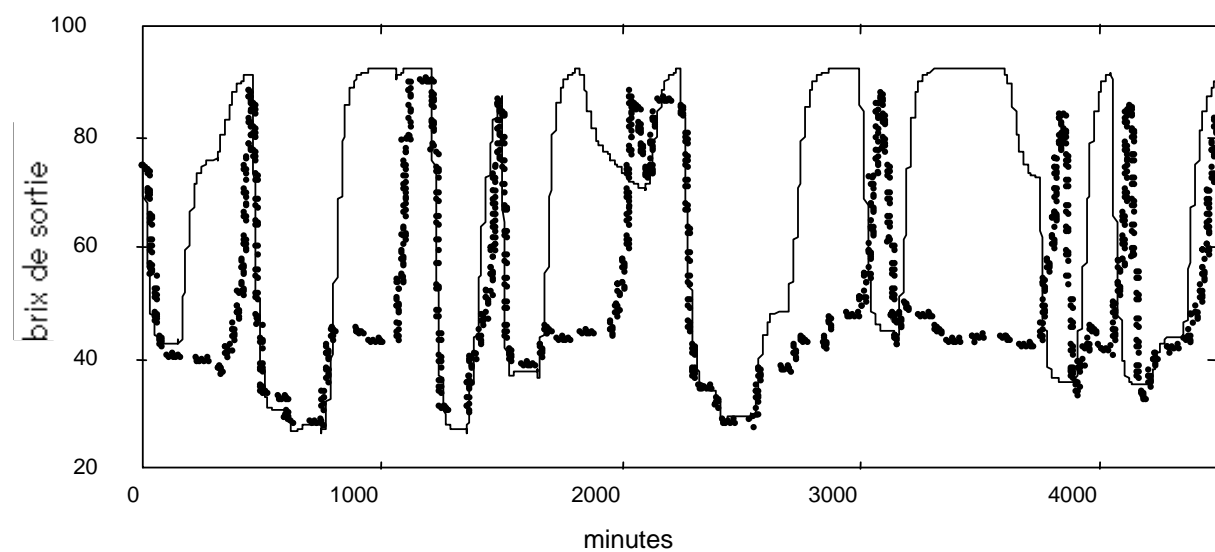
- **Pour prédire évolution à long terme,  
ou faire simulation “off-line”**

# MODELE PARALLELE LINEAIRE

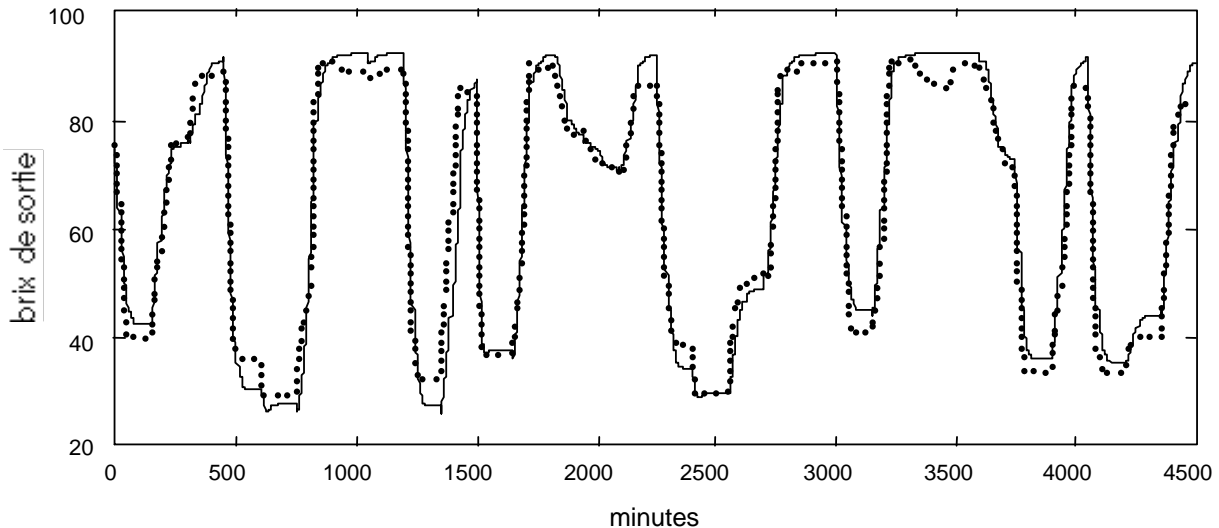


- erreur moyenne - 8,7 % (base de test)  
(ordre 4 sur la sortie et 2 sur les entrées)
- saturations non reproduites

# MODELE PARALLELE NEURONAL (1)



- **PREDICTEUR REBOUCLE**
- **un bon prédicteur peut avoir un mauvais comportement en mode parallèle**



- **APPRENTISSAGE RECURRENT**
- **erreur moyenne - 6,2 % (base de test)**
  - réseau 14-40-2
  - (ordre 4 sur les sorties et 2 sur les entrées)

## CONCLUSIONS ET REFERENCES

- **Réseau statique + rétropropagation**  
==> bon prédicteur
- **Réseau bouclé + APPRENTISSAGE RECURRENT**  
==> modèle parallèle meilleur que modèles linéaires

### Références sur cette application :

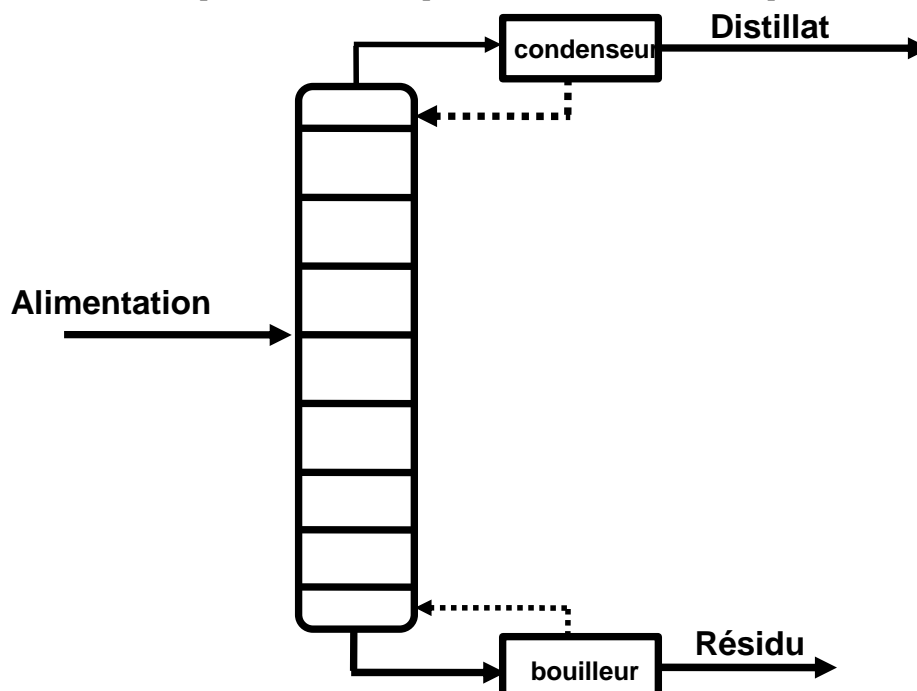
- "Réseaux de neurones pour l'identification de procédés industriels", F. Moutarde, P. Delesalle & M. Menahem, proceedings of Neuro-Nimes 92, Nîmes, 2-6 novembre 1992 (pages 443-454).
- "Potentialités d'applications des techniques neuronales en diagnostic et contrôle de procédés industriels", F. Moutarde, A. Benveniste & Q. Zhang, rapport final de l'étude MRT numéro 90S0866, décembre 1992.

- séquence d'apprentissage couvrant bien l'espace  $(x,u)$  ?
- influence du bruit sur l'apprentissage ?
- apprentissage délicat
  - modèles non-linéaires à éléments dynamiques linéaires
- ajout composante linéaire pour cas faiblement non-linéaires

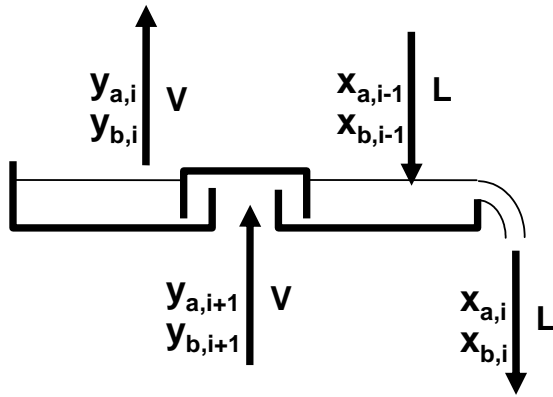
## EXEMPLE DE MODELE NEURONAL HYBRIDE

### • Colonne de distillation

(application réalisée pour Elf-Atochem par le laboratoire d'électronique de l'ESPCI et la société Netral)



# MODELE DE CONNAISSANCE D'UN PLATEAU



**a, b : constituants**  
**x, y : fractions molaires**  
 (resp. ds le liquide et dans la vapeur)  
**L : flux de liquide**  
**V : flux de vapeur**

$$M_i (dx_{a,i}/dt) = Lx_{a,i-1} + Vy_{a,i+1} - Lx_{a,i} - Vy_{a,i}$$

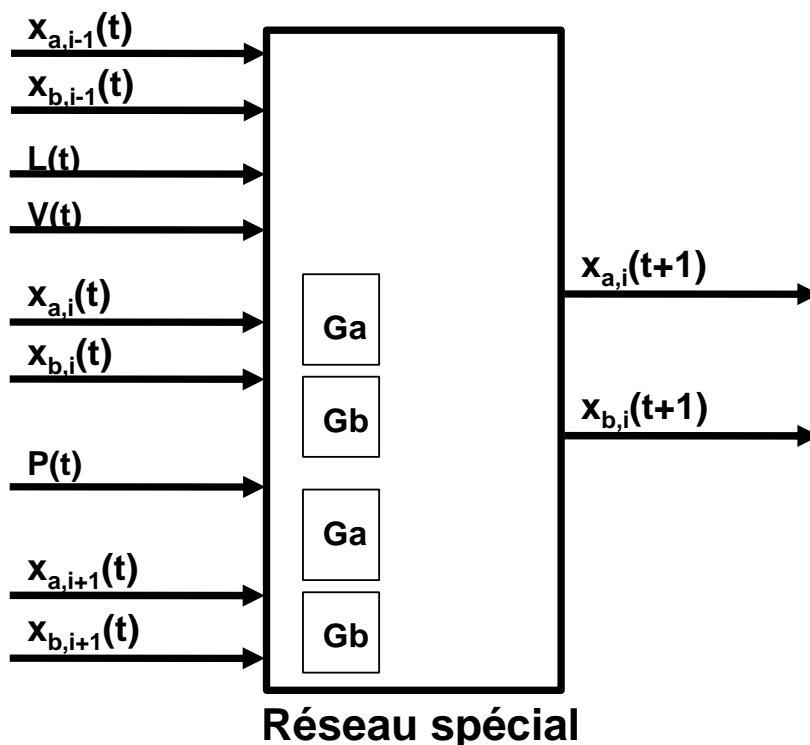
$$M_i (dx_{b,i}/dt) = Lx_{b,i-1} + Vy_{b,i+1} - Lx_{b,i} - Vy_{b,i}$$

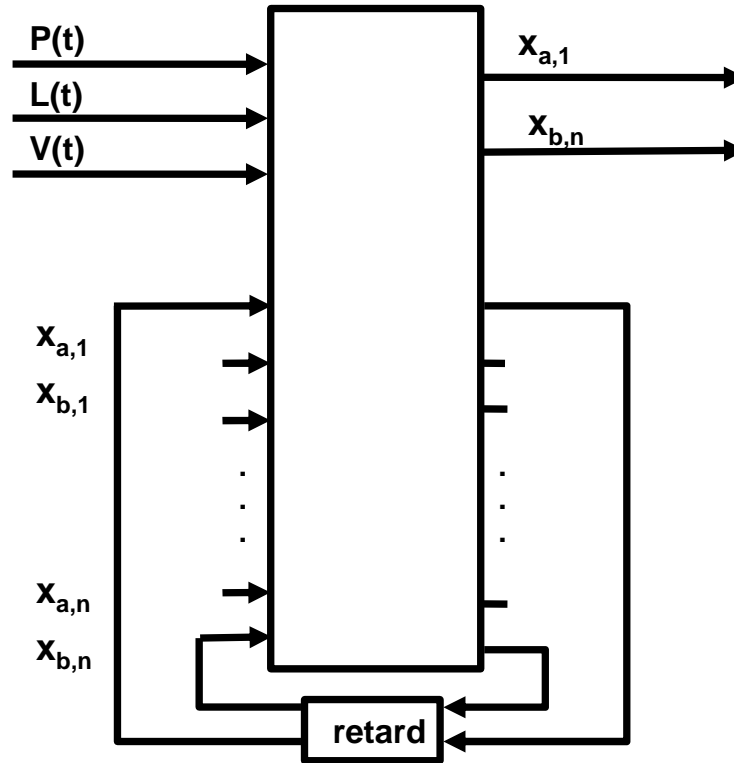
et  $y_{a,i} = G_a(P, x_{a,i}, x_{b,i})$   
 $y_{b,i} = G_b(P, x_{a,i}, x_{b,i})$  donc on peut éliminer les  $y_a, y_b$

$$\Rightarrow x_{a,i}(t+1) = LK_i x_{a,i-1}(t) - LK_i x_{a,i}(t) + VK_i G_a(P, x_{a,i+1}(t), x_{b,i+1}(t)) - VK_i G_a(P, x_{a,i}(t), x_{b,i}(t))$$

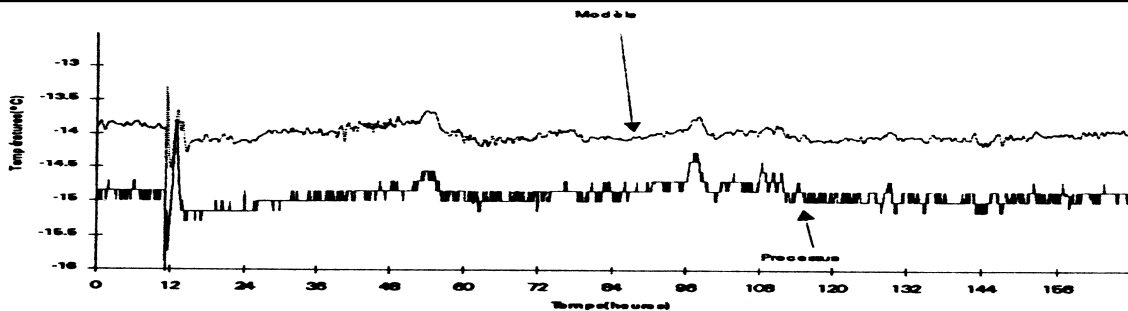
et idem pour les  $x_b$

# RESEAU DE PLATEAU

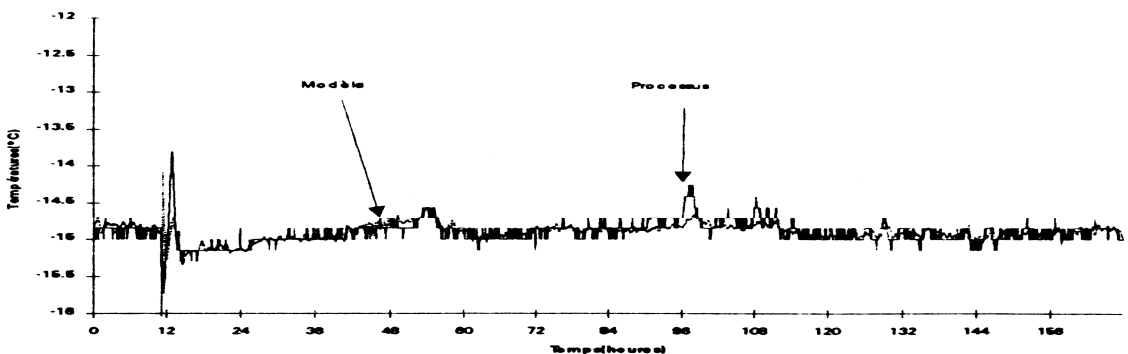




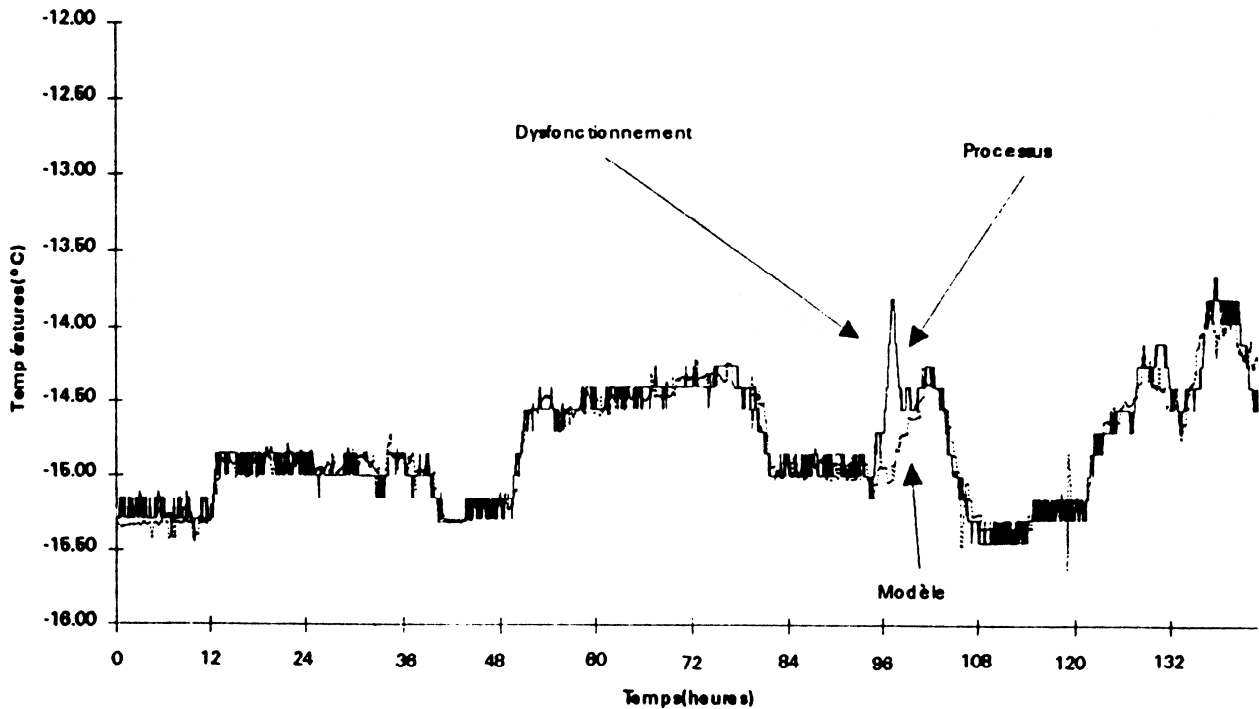
## RESULTATS D'APPRENTISSAGE



réseau entraîné sur simulateur v.s. colonne réelle



réseau après recalage par apprentissage sur données réelles



## CONCLUSIONS ET REFERENCE

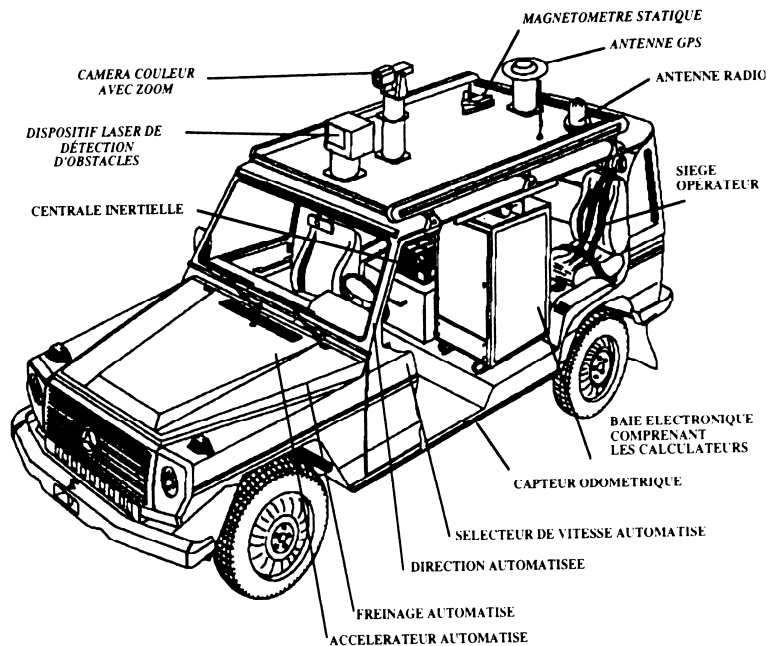
- **La connaissance sur le procédé permet de définir une architecture adaptée, donc un réseau plus simple**
- **L'existence d'un simulateur basé sur des équations physique permet de réaliser un bon apprentissage initial**
- **Le modèle neuronal obtenu a les avantages suivants :**
  - **recalage facile sur la colonne réelle**
  - **exécution beaucoup plus rapide que le simulateur physique**

Référence sur cette application :

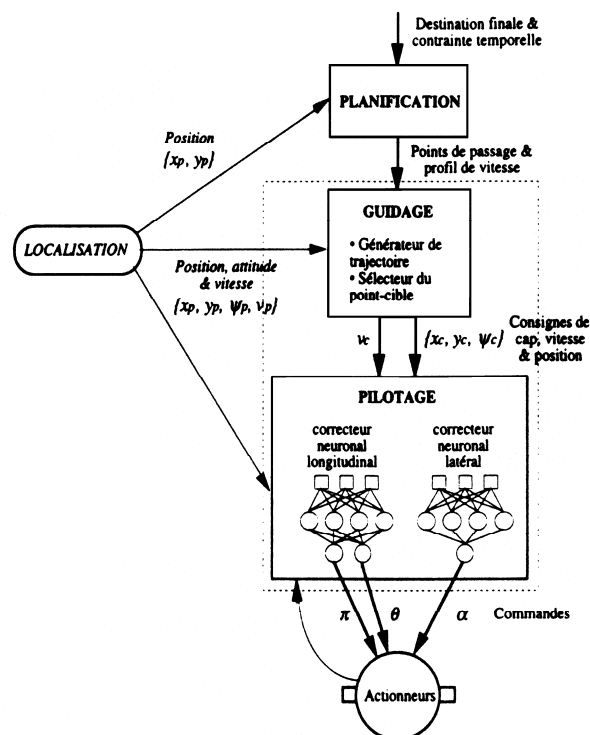
"From Knowledge-based Models to Recurrent Networks: an Application to an Industrial Distillation Process", Ploix J.-L., Dreyfus G., Corriou J.-P. & Pascal D., NeuroNîmes '94 (1994).

# EXEMPLE DE COMMANDE NEURONALE

- **Commande du volant d'un véhicule 4x4 autonome**  
(application réalisée par la société Sagem et le laboratoire d'électronique de l'ESPCI)



# ARCHITECTURE GLOBALE DU SYSTEME



The overall model, with state  $S(k) = [x(k), y(k), \psi(k), \beta(k), v(k)]$  is described by the following state equations :

$$x(k+1) = x(k) + v(k) \Delta T \cos \psi(k)$$

$$y(k+1) = y(k) + v(k) \Delta T \sin \psi(k)$$

$$\beta(k+1) = f_{NN\beta}[\beta(k), v_p(n), \alpha(k)]$$

$$\psi(k+1) = \psi(k) + v_p(h) f_{NN\psi}[\beta(k)]$$

$$v(k+1) = v(k) + f_{Uv}(S(k), \theta(k), B(k))$$

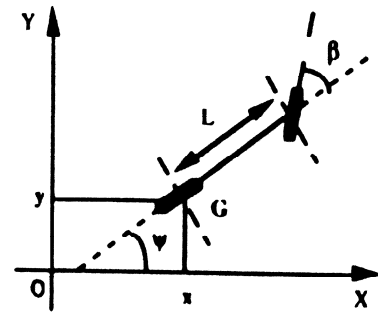
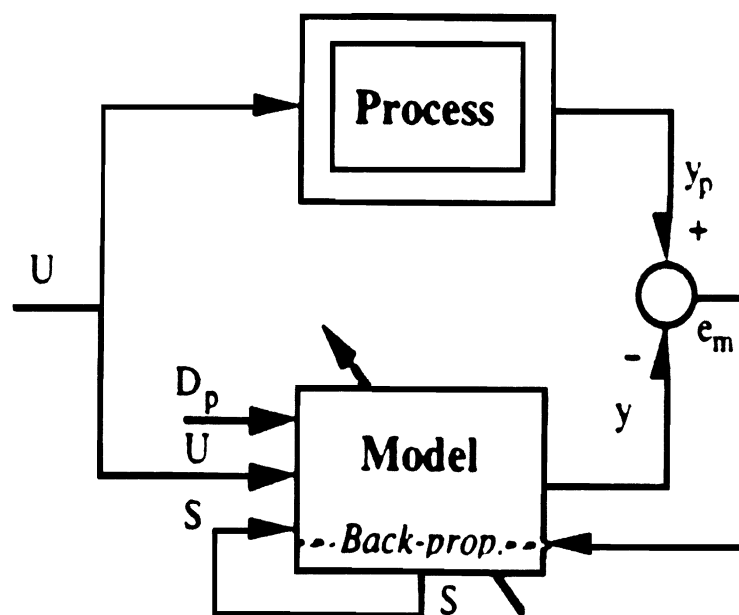
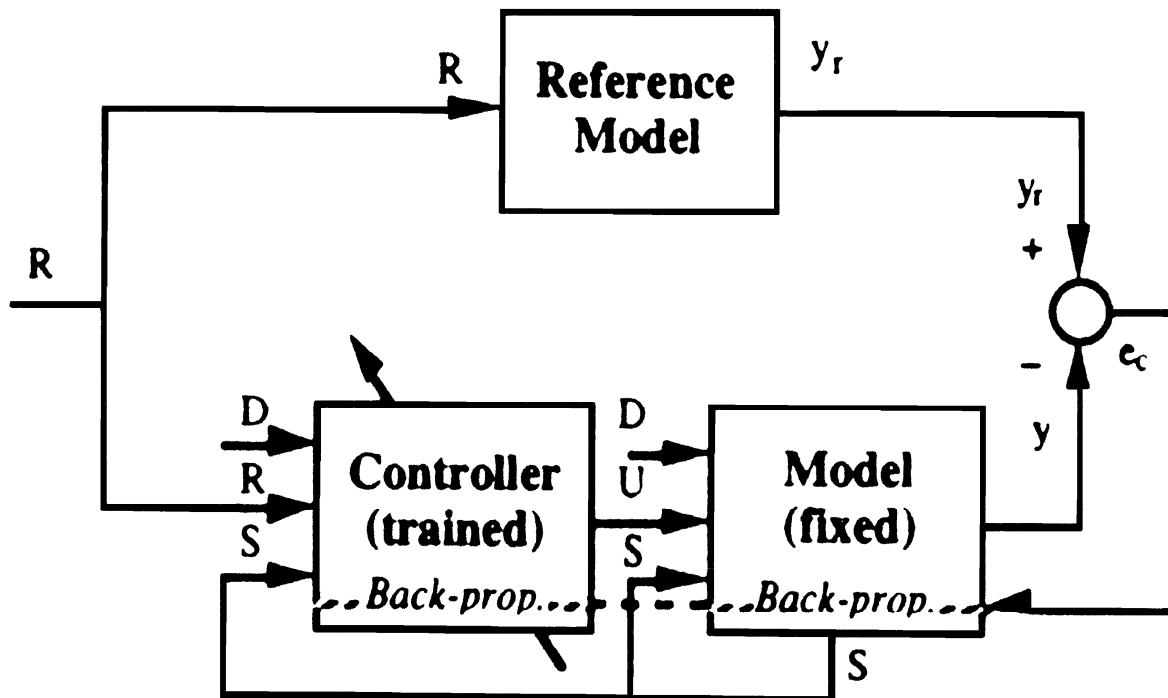


FIGURE 4  
The variables of the model.

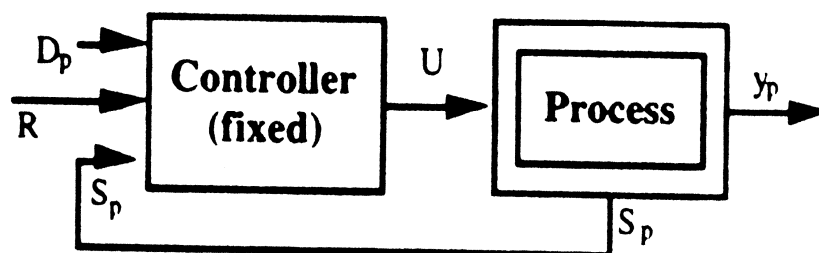
## IDENTIFICATION



- Algorithme semi-dirigé



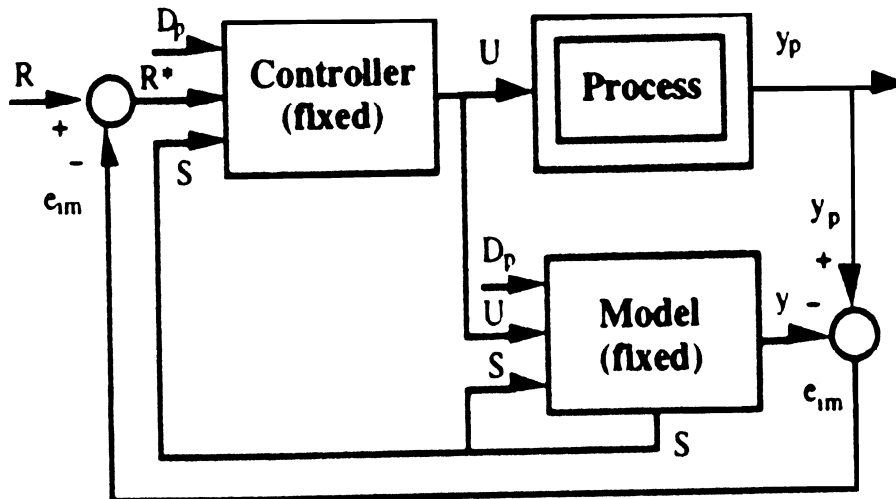
$$U(n) = h_{NN} [U(n-1), S_p(n), D_p(n), R(n)]$$



Operating phase : SFC system.

For this control system to be stable and show good tracking and regulation properties, the neural model used for the training of the controller has to be quite accurate.

$$U(n) = h_{NN} \left[ U(n-1), S(n), D_p(n), R^*(n) \right]$$

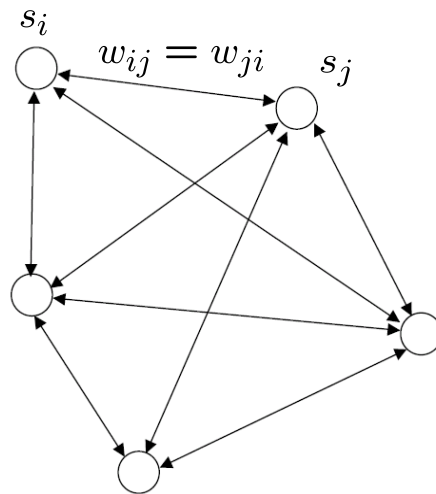


## REFERENCES SUR CETTE APPLICATION

- "Real-time control of an autonomous vehicle: a neural network approach to the path-following problem", Rivals I., Personnaz L., Dreyfus G. & Canas D., 5th International Conference on Neural Networks and their Applications (NeuroNîmes 93), p. 219-229 (1993).
- "Modeling and control of mobile robots and intelligent vehicles by neural networks", Rivals I., Canas D., Personnaz L. & Dreyfus G., IEEE Conference on Intelligent Vehicles, 24-26 octobre 1994, Paris, pp. 137-142 (1994).
- "Modélisation et commande de processus par réseaux de neurones ; application au pilotage d'un véhicule autonome", Rivals I., Thèse de Doctorat de l'Université Paris 6, (1995).

# Un réseau bouclé très particulier : le réseau de Hopfield

## Réseau totalement connecté et sans aucune entrée

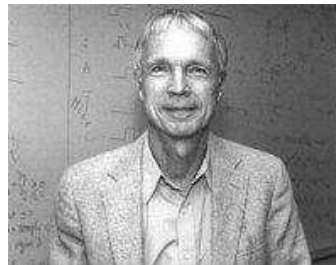


N neurones (ici N=5)  
à sorties binaires (+1 ; -1)

- **Connections symétriques**
- **Pas d'auto-activation :  $W_{ii} = 0$**

## Origine historique du réseau de Hopfield

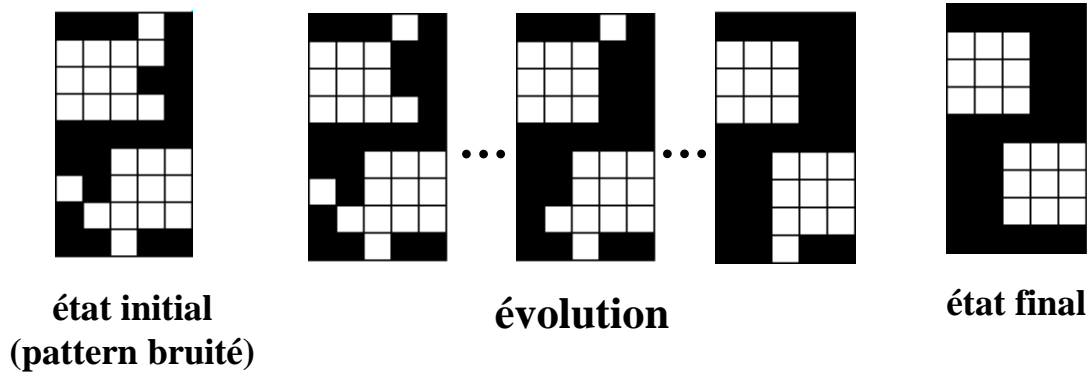
*Neural networks and physical systems with emergent collective computational properties, Hopfield and Tank, PNAS, 1982.*



J.J. Hopfield

- **Inspiration de la physique statistique : modèle des verres de spin de Ising, à savoir pour N atomes avec spins  $S_i \in \{-1 ; +1\}$ , chacun contribue au champ global par  $h_i = \sum_j (J_{ij} S_j)$  avec  $J_{ij}$  = influence de atome i sur atome j**

- **Mémoire auto-associative :**
  - initialisation des sorties
  - évolution dynamique → stabilisation (dans un attracteur)



**Plusieurs modes évolutions possibles :  
synchrone vs asynchrone(s)...**

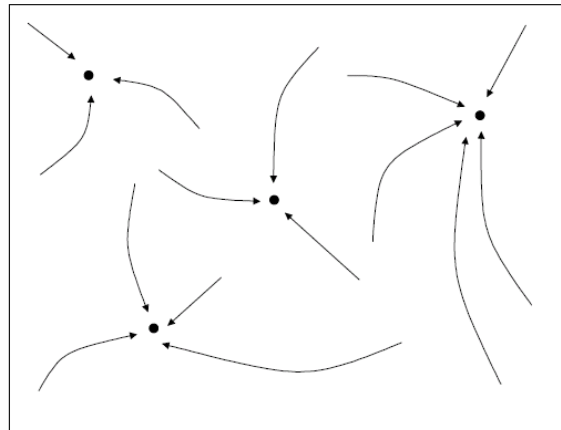
- En mode asynchrone, la convergence est garantie par l'existence (si  $W_{ij}=W_{ji}$ ) d'une fonction de Lyapunov qui ne peut que décroître au fil des itérations :

$$E = -\frac{1}{2} \sum_{i < j} (W_{ij} S_i S_j)$$

En effet, un neurone ne change de sortie que si  $S_i(t)$  et  $\sum(W_{ij}S_j(t))$  sont de signes opposés. Si le neurone  $i$  est inchangé,  $E=Cte$ . Sinon :

$$\Delta E = -\frac{1}{2} \left( \sum_{i < j} (W_{ij} S_i(t+1) S_j(t+1)) - \sum_{i < j} (W_{ij} S_i(t) S_j(t)) \right) = -\frac{1}{2} (S_i(t+1) - S_i(t)) \sum_j (W_{ij} S_j(t))$$

$\Delta E$  forcément  $<0$  si chgt état, alors que  $E$  borné par  $E > -\sum_{i < j} W_{ij}$   
→ convergence vers un point fixe (minimum local  $\equiv$  attracteur)



**Un Hopfield efficace est un système dynamique dont on a « choisi » les attracteurs, et fait en sorte que leurs bassins d'attraction soient les plus larges possible**

## Apprentissage Hopfield

- La mémorisation comme attracteurs d'un ensemble de motifs (patterns)  $\{(P_i^k, k=1, \dots, K)\}$  est TRES SIMPLE :

$$W_{ij} = \sum_k P_i^k P_j^k \quad (\sim \text{r\`e}gle \text{ Hebb})$$

→ tous les  $P^m$  stables *sous certaines conditions* :

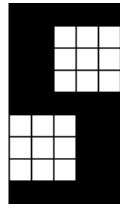
$$\begin{aligned} \sum_j (W_{ij} P_j^m) &= \sum_j \sum_k (P_i^k P_j^k P_j^m) = \sum_j (P_i^m + \sum_{k \neq m} (P_i^k P_j^k P_j^m)) \\ &= N P_i^m + \sum_{k \neq m} (P_i^k \sum_j (P_j^k P_j^m)) \end{aligned}$$

- En pratique ça ne marche bien que si le « taux de chargement »  $K/N < 0.138$  !!  
(voire moins si motifs corréllés entre eux)

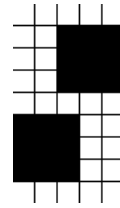
- De plus, il se forme des attracteurs « parasites »

- Par symétrie, tous les motifs « inverses » sont aussi mémorisés :

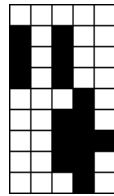
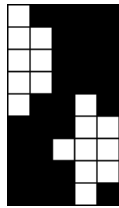
stockage de



→ mémorisation de



- D'autres sont des combinaisons linéaires des patterns mémorisés, et autres « états poubelles »



→ Éventuellement les « désapprendre » par  $\Delta W_{ij} = -\eta S_i S_j$  avec  $\eta \ll 1$  ou mieux : « orthogonaliser » les motifs

- Du fait de la contrainte  $\text{nbMotifs} \ll \text{nbNeurones}$ , et des pb d'attracteurs parasites, assez peu d'applications pratiques de type mémoire auto-associative...
- Par contre, en faisant en sorte que la fonction de Lyapunov du Hopfield corresponde à qqch que l'on veut minimiser, quelques applications :
  - En optimisation combinatoire (pb NP-complets) :
    - type « voyageur de commerce »
    - affectation de ressources à des tâche, ordonnancement,...
  - En amélioration d'images
  - ...

# QUELQUES REFERENCES SUR LES RESEAUX NEURONAUX

---

- Des livres :

- *Réseaux de neurones : méthodologie et applications*,  
**G. Dreyfus et al., Eyrolles, 2002.**
- *Réseaux de neurones formels pour la modélisation, la  
commande, et la classification*, **L. Personnaz et I. Rivals, CNRS  
éditions, collection Sciences et Techniques de l'Ingénieur, 2003.**
- *Réseaux de neurones : de la physique à la psychologie*,  
**J.-P. Nadal, Armand Colin, 1993.**

- Sur le Web :

- La « FAQ » sur les réseaux neuronaux :  
<http://www.faqs.org/faqs/ai-faq/neural-nets/part1/>
- Les pages du groupe «neuronal » de l'UTC de Compiègne :  
<http://www.hds.utc.fr/~grandval/rna/>
- Les pages du cours «neuronal » de l'INSA de Rouen :  
<http://asi.insa-rouen.fr/~scanu/coursRNA/>
- Voir aussi les liens depuis ma homepage :  
[http://www.ensmp.fr/~moutarde/rech\\_neuronal.html](http://www.ensmp.fr/~moutarde/rech_neuronal.html)