

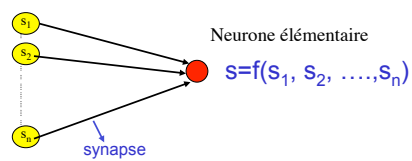
# Réseaux de neurones Multicouches

## Partie 1

## Neurone

### Caractérisé par:

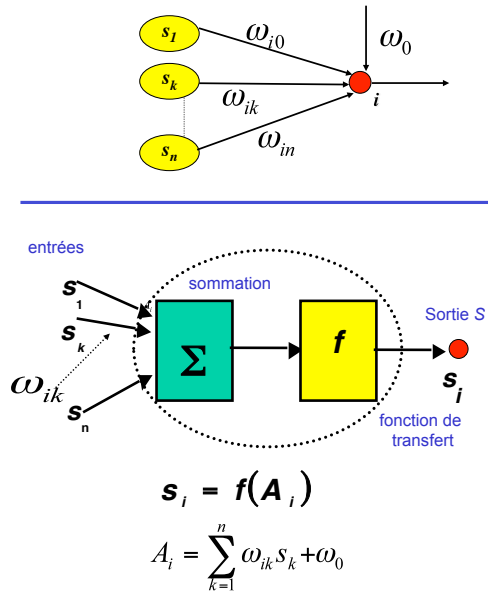
- état interne  $s \in S$
- voisinage  $s_1, \dots, s_n$
- fonction de transfert  $f$
- changement d'état  $s = f(s_1, s_2, \dots, s_n)$



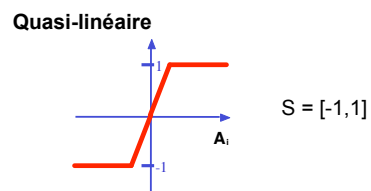
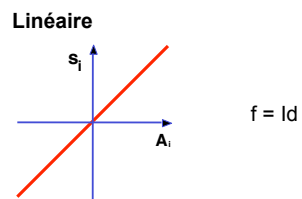
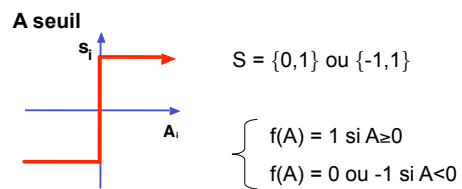
### Exemples

- $S = \{0, 1\}$  ou  $\{-1, 1\}$  **neurone binaire**  
1 : actif  
0/-1 : inactif
- $S = \{0, 1, 2, \dots, k\}$  **neurone discret**  
*Si un neurone représente un pixel dans une image,  $i \in S$  représente le niveau de gris utilisé dans le codage.*
- $S = [a, b]$  **neurone continu**

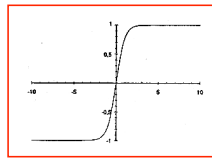
## Neurone produit scalaire



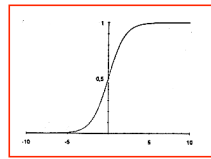
## Fonctions de transfert



### Fonctions sigmoïde

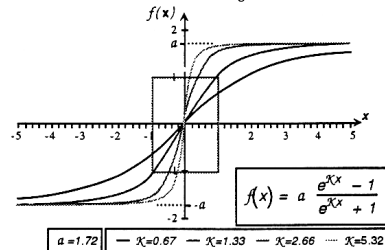


$$a \frac{1 - e^{-kx}}{1 + e^{-kx}}$$

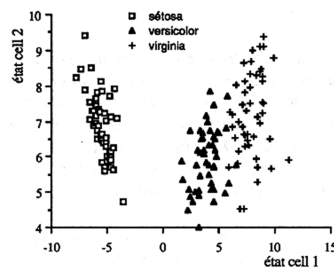


$$a \frac{1}{1 + e^{-kx}}$$

Famille de fonctions Sigmoides



### Classification



- Connaissant la position d'un point, déterminer automatiquement sa classe.
- D'un point de vue géométrique: Déterminer les surfaces séparatrices entre les classes.



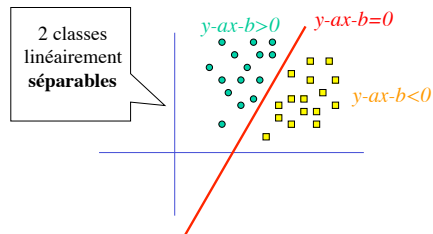
#### Solution:

Rechercher des surfaces séparatrices parmi une famille paramétrée donnée.

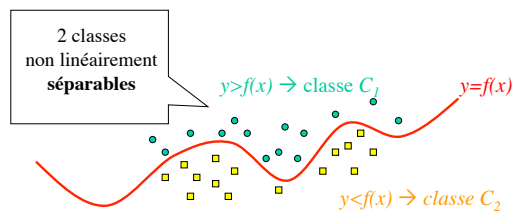
#### Apprentissage:

Déterminer les bons paramètres pour réaliser «au mieux» la tâche de séparation.

## Quelle famille de surfaces choisir?



Cas simple : une séparation linéaire



Les surfaces séparatrices peuvent être complexes

## Exemple : 2 classes Séparation par hyperplans

Notations :  $x \in \mathbb{R}^n$  (input)

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

$C_1$  : ensemble des formes de la classe 1  
 $C_2$  : ensemble des formes de la classe 2

Hyperplan dans  $\mathbb{R}^n$  défini par l'équation :

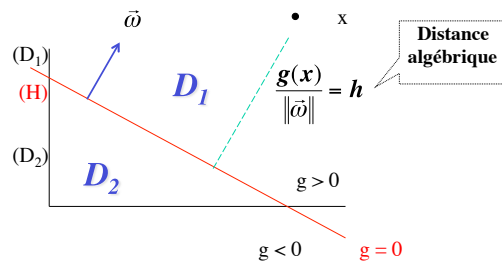
$$(H) \quad \sum_{i=1}^n \omega_i x_i + \omega_0 = 0$$

$$\vec{\omega} = \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_n \end{pmatrix}$$

Vecteur orthogonal à  $H$

### Distance algébrique $h$ de $x$ à l'hyperplan

$$h = \frac{\sum_{i=1}^n \omega_i x_i + \omega_0}{\|\vec{\omega}\|} \quad \text{avec} \quad \|\vec{\omega}\|^2 = \sum_{i=1}^n \omega_i^2$$



• Si on pose :  $g(x) = \sum_{i=1}^n \omega_i x_i + \omega_0$

$g(x) = 0 \Leftrightarrow x \in (H)$

• L'hyperplan (H) sépare l'espace en 2 demi-espaces qui correspondent à :

$$D_1 = \{x / g(x) > 0\} \quad D_2 = \{x / g(x) < 0\}$$

### $g$ : fonction de décision

#### Décider

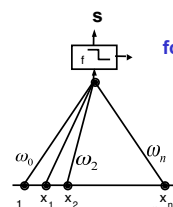
- $C_1$  si  $g(x) = w^t x > 0$
- $C_2$  si  $g(x) = w^t x \leq 0$

**Remarque :** Si  $\omega_0 = 0$  Alors H passe par l'origine

### Formulation neuronale :

Cas de 2 classes

#### Neurone à seuil



$$S = f(A) = 1 \text{ si } A \geq 0$$

$$S = f(A) = -1 \text{ si } A < 0$$

$$s = 1 \text{ si } \sum_{i=1}^n \omega_i x_i + \omega_0 \geq 0$$

$$s = -1 \text{ si } \sum_{i=1}^n \omega_i x_i + \omega_0 < 0$$

# Réseaux Multicouches

**MLP: Multi Layer Perceptron**

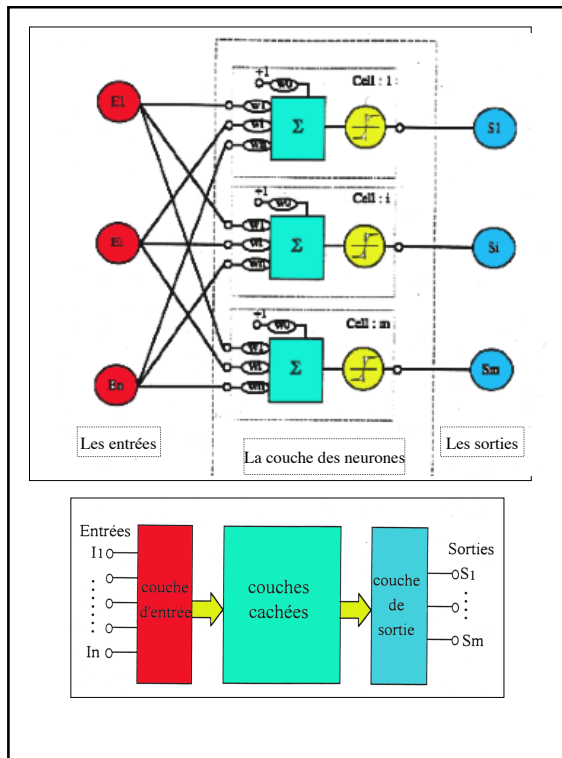
**PMC: Perceptron Multi-couche**

## Définition

- Les neurones sont réparties dans des couches successives  $(C_k)_{k=0, \dots, C+1}$
  - La couche  $C_0$  : « **Couche d'entrée** », contient  $n$  neurones imposées de l'extérieur.
  - La couche  $C_{C+1}$  : « **Couche de sortie** », contient  $p$  neurones.
  - Les couches  $(C_k)_{1 \leq k \leq C}$  sont les « **Couches cachées** ».
- La couche  $C_k$  contient  $n_k$  neurones ( $1 \leq k \leq C$ )
- Les états des neurones de  $C_0$  sont dans  $\mathcal{R}$ .
  - Les états des autres neurones sont en général dans  $[-a, a]$ .
- Les seules **connexions autorisées** sont d'un neurone de  $C_k$  à un neurone de  $C_l$  avec  $k < l$ .

### Les fonctions de transferts des neurones

- Les neurones d'entrées n'ont pas de fonctions de transferts, leurs états étant imposés par l'extérieurs.
- Les neurones cachés ont des fonctions de transferts sigmoïdes.
- Les neurones de sorties, suivant les applications, ont des fonctions de transfert : sigmoïdes, linéaires, exponentielles ou autres



### DYNAMIQUE - PROPAGATION AVANT DES ETATS

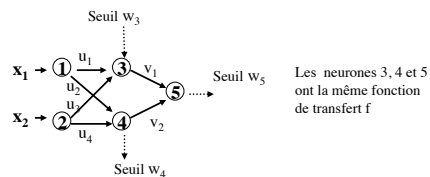
On présente un vecteur input  $x = (x_1, x_2, \dots, x_n)$  à la couche d'entrée qui sera propagé d'une couche à une autre vers la couche de sortie.

$y$  : étant le vecteur de sortie « output » calculé.

$G$  : fonction définie par le réseau :  $y = G(x, W)$

$W$  représente l'ensemble des poids Synaptiques et des seuils

#### Exemple de propagation avant



#### Etat des neurones :

$$3 : s_3 = f(u_1x_1 + u_3x_2 + w_3)$$

$$4 : s_4 = f(u_2x_1 + u_4x_2 + w_4)$$

$$5 : s_5 = f(v_1s_3 + v_2s_4 + w_5)$$

Ainsi dans ce cas :

$$Y = G(X, W) = f[v_1 f(u_1x_1 + u_3x_2 + w_3) + v_2 f(u_2x_1 + u_4x_2 + w_4) + w_5]$$

## Position du problème

Choisir l'architecture du réseau (ayant  $n$  entrées et  $p$  sorties).

On note par  $W$  l'ensemble des poids et des seuils.

Calculer les états : propagation de couche en couche.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{Entrée du réseau} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} \quad \text{Sortie calculée}$$

Le réseau définit une famille de fonctions paramétrée par  $W$  :

$$G : \Re^n \rightarrow \Re^p$$

$$\text{Avec } \mathbf{y} = G(\mathbf{x}, W)$$

On note cette famille de fonctions par :

$$G(., W)$$

## Apprentissage

On dispose d'un ensemble d'apprentissage :

$$\text{App} = \{(\mathbf{x}^k, \mathbf{d}^k) ; k=1, \dots, N\}$$

Où  $\mathbf{x}^k$  est la représentation d'un individu et  $\mathbf{d}^k$  est la réponse qui lui est associée.

Pour une architecture fixée et un système de poids donnés  $W$ , le réseau définit une fonction  $G(., W)$ .

Pour un individu  $\mathbf{x}^k$  le réseau calcule la sortie  $\mathbf{y}^k$  :

$$\mathbf{y}^k = G(\mathbf{x}^k, W)$$

L'apprentissage consiste à trouver les poids  $W^*$  de façon que pour tout  $\mathbf{x}^k$  :  $\mathbf{y}^k \cong \mathbf{d}^k$



**Minimiser une fonction erreur  $J(\text{App}, W)$**



## Erreur Quadratique

L'erreur entre  $d^k$  et  $y^k$  sera mesurée par la distance euclidienne :

$$J_k(W) = \|d^k - y^k\|^2 = \sum_{i=1}^p (d_i^k - y_i^k)^2$$

Où  $y_i^k$  et  $d_i^k$  sont les  $i^{\text{ème}}$  composantes de  $y^k$  et  $d^k$

La fonction « **erreur quadratique** » est alors définie par :

$$\mathfrak{J}(W) = \sum_{k=1}^N J_k(W) = \sum_{k=1}^N \|d_k - G(x_k, W)\|^2$$

**Il s'agit d'une erreur globale**  
**L'apprentissage consiste à minimiser cette fonction :**

$$W^* = \underset{W}{\text{ArgMin}} \mathfrak{J}(W, App)$$

**La minimisation se fait par une méthode de gradient**

**Qui nécessite le calcul des :**  $\frac{\partial \mathfrak{J}(W)}{\partial w_{ij}}$

## Algorithme de la rétro-propagation du gradient

➡ Détermination de

$$\mathfrak{J}(W, App) = \sum_k J_k(W)$$

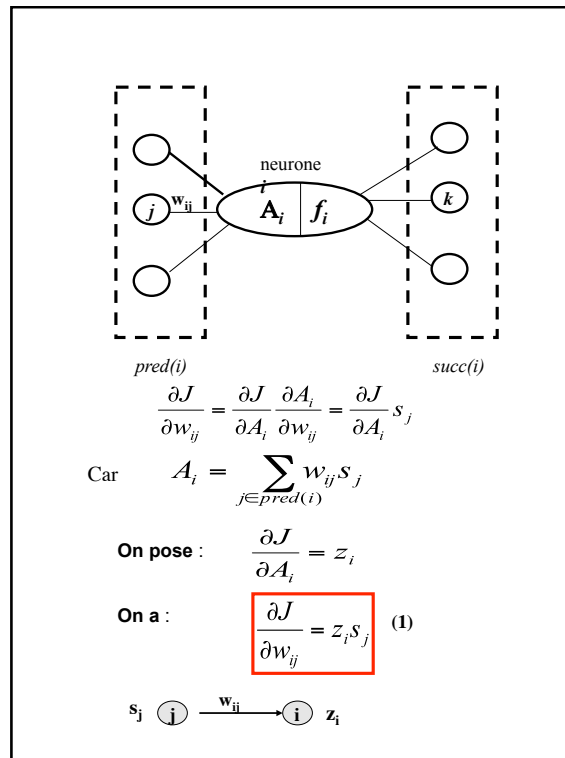
Erreur sur la  $k^{\text{ème}}$  forme

➡ Calcul la dérivée de  $\mathfrak{J}$  par rapport  $w_{ij}$  de :

$$\frac{\partial \mathfrak{J}(W)}{\partial w_{ij}} = \sum_{k=1}^N \frac{\partial J_k(W)}{\partial w_{ij}}$$

Il suffit de savoir calculer :  $\frac{\partial J_k(W)}{\partial w_{ij}}$

Afin de simplifier les notations, on note par la suite ce terme par  $\frac{\partial J(W)}{\partial w_{ij}}$



### Pour $i$ : neurone de la couche de sortie

- La quantité  $A_i$  intervient directement dans  $J$  :

$$J = \sum_{i=1}^p (f(A_i) - d_i)^2$$

et

$$z_i = \frac{\partial J}{\partial A_i} = \frac{\partial J}{\partial f(A_i)} \frac{\partial f(A_i)}{\partial A_i}$$

$$z_i = 2(f(A_i) - d_i) f'(A_i) \quad (2)$$

(1) et (2)



$$\frac{\partial J}{\partial w_{ij}} = 2(f(A_i) - d_i) f'(A_i) s_j \quad (3)$$


### Pour i : neurone sur une couche cachée

La quantité  $A_i$  intervient dans la fonction  $J$  par le biais du calcul des états de l'ensemble des neurones successeurs de  $i$   $\text{succ}(i)$ .

$$z_i = \frac{\partial J}{\partial A_i} = \sum_{k \in \text{succ}(i)} \frac{\partial J}{\partial A_k} \frac{\partial A_k}{\partial A_i}$$

Comme

$$A_k = \sum_{l \in \text{pred}(k)} f(A_l) w_{kl}$$


$$\frac{\partial A_k}{\partial A_i} = f'(A_i) w_{ki}$$

On tire alors :

$$z_i = f'(A_i) \sum_{k \in \text{succ}(i)} z_k w_{ki} \quad (4)$$

### Calcul des $z_i$

- (2) permet de calculer  $z_i$  pour les neurones de sortie.
- (4) permet (par récurrence) de calculer  $z_i$  pour les autres neurones (cachées).

*On commence le calcul pour les neurones de sortie, puis pour les neurones de l'avant dernière couche et ainsi de suite jusqu'aux neurones de la seconde couche*



### Algorithme de la Rétro-propagation du gradient

## Modification des poids

### À l'itération $h$

$$w_{ij}^{(h)} = w_{ij}^{(h-1)} + \Delta w_{ij}$$

$$\Delta w_{ij} = -\varepsilon \frac{\partial J}{\partial w_{ij}}$$

Tenant compte de :  $\frac{\partial J}{\partial w_{ij}} = z_i s_i$

$$\Delta w_{ij} = -\varepsilon_h z_i s_i$$



$$w_{ij}^{(h)} = w_{ij}^{(h-1)} - \varepsilon_h z_i s_{ij}$$

## Conclusion

$J$  étant l'erreur relative à un couple  $(x, d)$  de la base d'apprentissage.

Le calcul du gradient  $\frac{\partial J}{\partial W}$  se présente ainsi :

- Présenter  $x$  aux neurones de la couche d'entrée et faire un **propagation avant** afin de calculer les états des neurones du réseau MLP
- Initialiser les neurones de sorties par (2) et appliquer l'algorithme de la **rétro-propagation du gradient** afin de calculer les  $z_i$  des neurones.
- Pour chaque poids synaptique  $w_{ij}$  appliquer la formule :

$$\frac{\partial J}{\partial w_{ij}} = z_i s_j$$

**Remarque :** Le seuil  $\omega_i$  d'un neurone  $i$  peut être considéré comme un poids synaptique particulier. Il suffit d'ajouter un neurone fictif à la couche d'entrée ayant un état constant égale à 1 et d'interpréter  $\omega_i$  comme étant son poids synaptique.

## Cas particulier

### :Classification

p classes  $\omega_1, \omega_2, \dots, \omega_p$

$$x_k \in \omega_k \Rightarrow d_k = (0, 0, \dots, 1, 0, \dots, 0)$$

$$\text{Ou } d_k = (-1, -1, \dots, 1, -1, \dots, -1)$$

Codage  
des classes

Classification

$$x \in \omega_i \Leftrightarrow g_i(x) \geq g_j(x) \quad \forall j \neq i$$

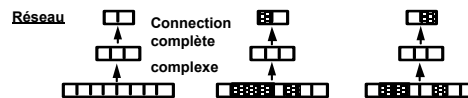
Détermination de frontières de décision complexes

STRUCTURE REVELÉE	TYPES OF DISCRETE RELATIONS	EXPLANATION OF RELATIONS	CLASSICAL ARTI- FICIAL RELATIONS	NEURAL NETWORK ARTIFICIAL RELATIONS
	Simple relations between input and output	Simple relations between input and output	Simple relations between input and output	Simple relations between input and output
	Complex relations between input and output	Complex relations between input and output	Complex relations between input and output	Complex relations between input and output
	Complex relations between input and output	Complex relations between input and output	Complex relations between input and output	Complex relations between input and output

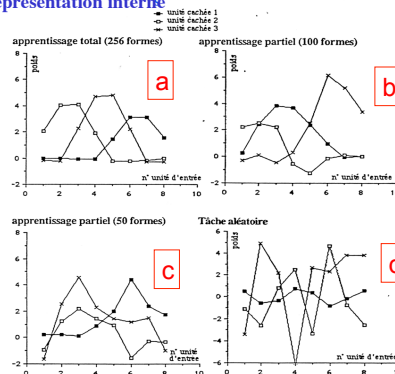
## Mise au point pratique d'un réseau multicouche

Exemple 1 Problème: détecter un événement dans un signal "3 dans 8"

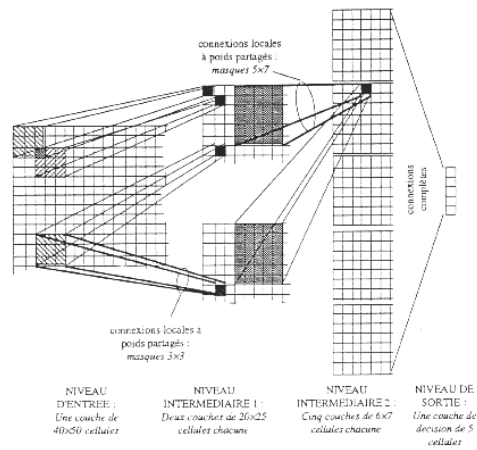
Exemple Contre exemple 256 exemples



Représentation interne



## Architecture à masque



•Exemple d'une architecture multicouche pour la reconnaissance de visages [Viennet 91].

## Reconnaissance de locuteur [Y. Bennani]

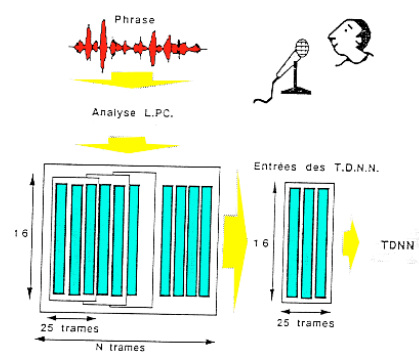
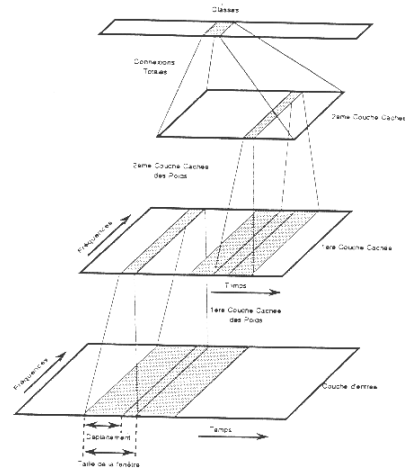


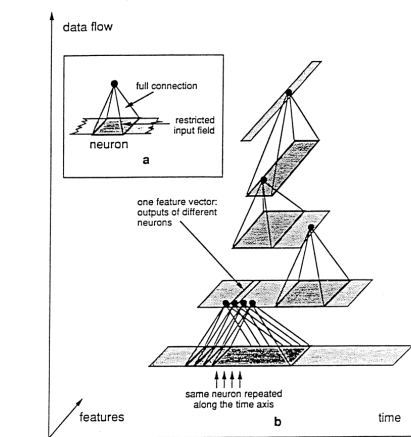
Figure 10.4 : Mécanisme du fenêtrage sur les phrases

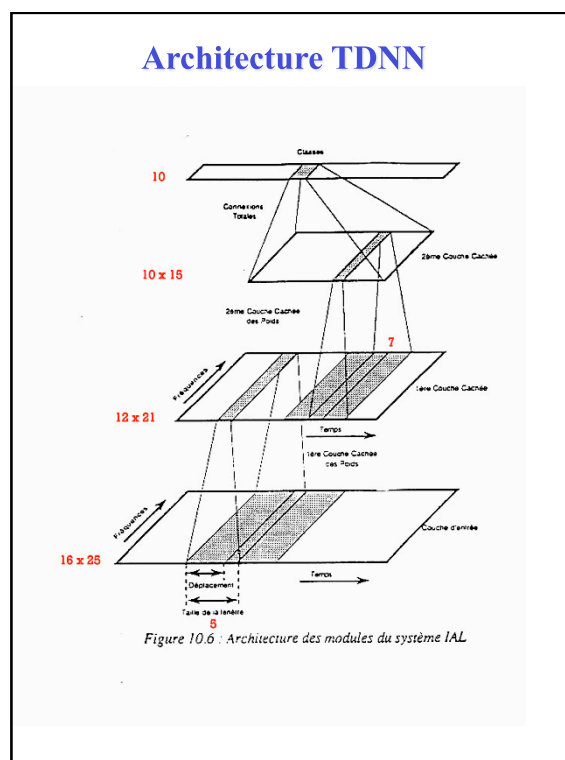
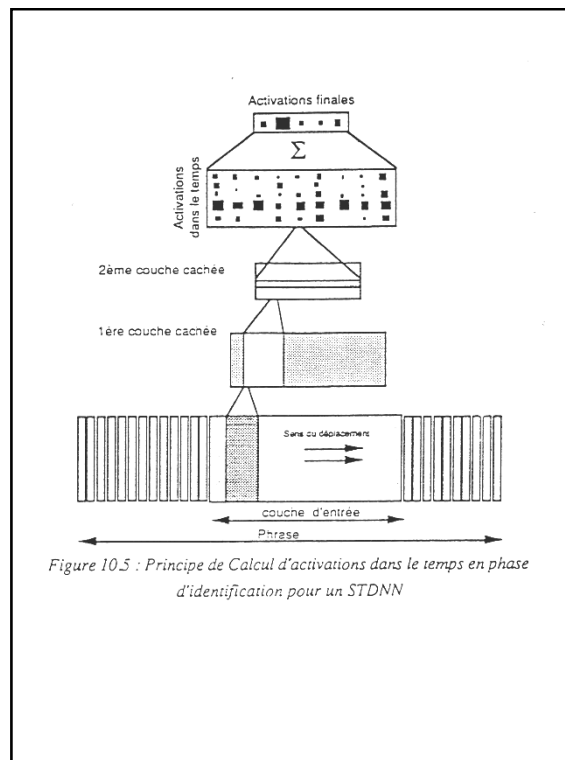
## Architecture TDNN (Time-Delay Neural Networks)



### •Exemple d'un réseau de neurones TDNN.

•Architecture utilisée pour l'identification vocale du locuteur [Bennani 92].







### ACP des couches successives

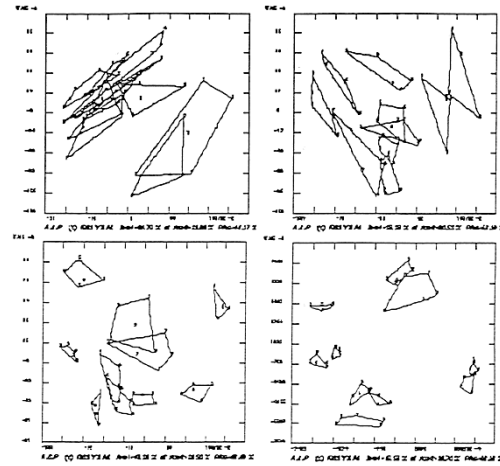


Figure 10.10 : ACP des couches successives du module M3

### Apprentissage : Un exemple de la régression

On dispose d'une base d'apprentissage :

$$\{(x^k, d^k) ; k = 1, \dots, N\}$$

Où  $x^k$  appartient à  $\mathcal{R}$  et  $d^k$  à  $\mathcal{R}$ .

L'apprentissage consiste à déterminer une fonction  $G(., w)$  de  $\mathcal{R}$  dans  $\mathcal{R}$  et qui minimise l'erreur quadratique :

$$\sum_{k=1}^N (F(x^k, w) - d^k)^2$$

Il s'agit donc d'un problème de **régression simple**.

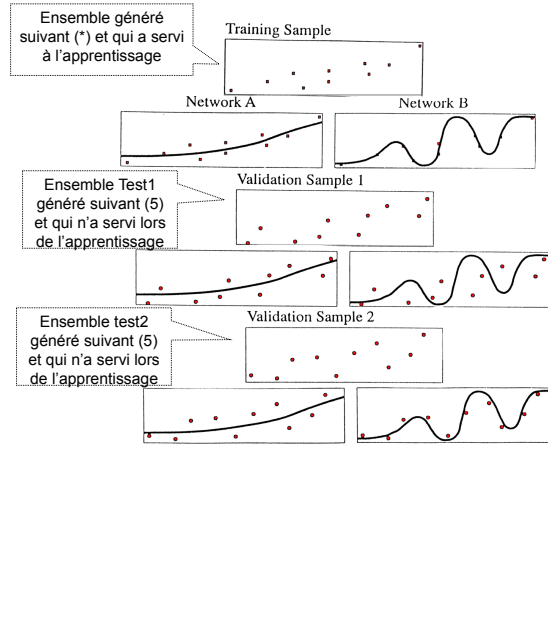
Si nous faisons l'hypothèse que les réponses désirées  $d^k$  sont générées de la manière suivante :

$$d^k = g(x^k) + \epsilon_k \quad (5)$$

Où  $g$  est une fonction de  $\mathcal{R}$  dans  $\mathcal{R}$  et  $\epsilon_k$  est un bruit blanc tiré suivant la loi normale  $N(0, \sigma)$ .

**Normalement, le but de l'apprentissage, dans ce cas, est de retrouver  $g$  qui est la fonction sous-jacente aux données.**

### Un exemple simple de la régression :



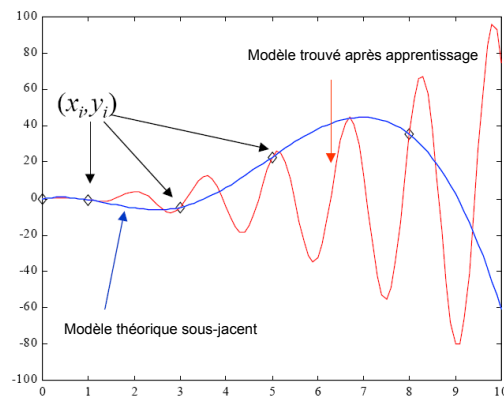
### Remarques concernant l'exemple

- Les réseaux A et B sont formés d'une couche cachée ayant respectivement 2 et 5 neurones.
- Ainsi, lors de l'apprentissage, le réseau B arrive à générer une courbe plus complexe et qui passe par les points de l'ensemble d'apprentissage (donc il annule l'erreur quadratique).
- Par contre, le réseau A arrive à mieux reproduire la fonction ( $g$ ) qui est sous-jacente aux données générées.
- Quand on visualise le comportement ces deux réseaux sur les deux bases TEST1 et TEST2 on remarque que :
  - Le réseau A un comportement plus robuste sur les deux bases Test1 et Test2.
  - Le réseau B un comportement beaucoup moins robuste puisqu'il est très mauvais relativement à test1 et beaucoup mieux relativement à Test2

**Remarque :** L'apprentissage ne consiste pas à trouver l'architecture qui minimise jusqu'au bout la fonction erreur définie sur l'ensemble d'apprentissage

## Le sur-apprentissage

*Apprentissage "par coeur" : le modèle ne "connaît" que les points utilisés pour l'apprentissage et fait n'importe quoi ailleurs.*



## Apprentissage et Généralisation

- Un système est décrit comme un vecteur aléatoire  $\mathbf{x}$  dont les valeurs sont régies par la densité de probabilité  $p(\mathbf{x})$ .
- A chaque vecteur  $\mathbf{x}$  est associée un vecteur  $\mathbf{y}$  qui se réalise suivant la loi conditionnelle  $p(\mathbf{y}/\mathbf{x})$ .
- L'apprentissage consiste à ajuster les paramètres  $\mathbf{W}$ , pour une famille de fonctions  $\{F(\mathbf{x}, \mathbf{W})\}_{\mathbf{W}}$ , en minimisant l'erreur d'apprentissage calculée sur un ensemble de  $N$  exemples:

$$App = \{(\mathbf{x}^k, \mathbf{d}^k) ; k=1, \dots, N\}$$

$$E_{App} = \frac{1}{N} \sum_{k=1}^N \|\mathbf{d}^k - F(\mathbf{x}^k, \mathbf{W})\|^2$$

**Mais la fonction  $F(\mathbf{x}, \mathbf{W}^*)$  ne minimise pas nécessairement l'erreur en généralisation :**

$$E_{gene} = \int \|\mathbf{y} - F(\mathbf{x}, \mathbf{W})\|^2 p(\mathbf{x}) p(\mathbf{y}/\mathbf{x}) d\mathbf{x} d\mathbf{y}$$

En effet, l'objectif de l'apprentissage est de proposer un modèle capable d'avoir de « bonnes performances » sur l'ensemble des exemples possibles.

## Estimation, de l'erreur en généralisation

- L'erreur en généralisation  $E_{\text{gene}}$  est théorique, on ne peut pas la calculer. Il faut donc lui définir un estimateur.
- Si l'on dispose de suffisamment de données, on pourra alors retenir une partie pour l'ensemble d'apprentissage ( $App$ ) et utiliser une partie de ce qui reste afin de constituer une sorte base appelée base de teste :  $Test = \{(x^i, y^i) ; i=1, \dots, p\}$
- On définit alors l'erreur moyenne quadratique sur cet ensemble :  

$$E_{Test} = \frac{1}{p} \sum_{i=1}^p \|y^i - F(x^i, w)\|^2$$
- $E_{\text{test}}$  dépend du choix aléatoire de l'ensemble de teste, elle représente donc une variable aléatoire.

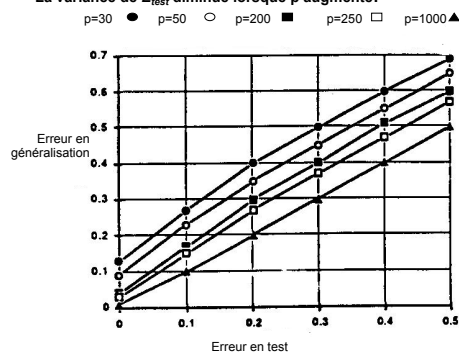
**On démontre que la moyenne de  $E_{\text{test}}$  relativement à tous les ensembles d'apprentissages possibles et de taille  $p$  est égale à  $E_{\text{gene}}$ .**

**$E_{\text{test}}$  est un estimateur non biaisé de  $E_{\text{gene}}$**

## Quelques remarques

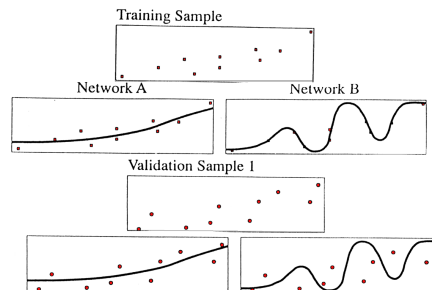
- $E_{\text{gene}}$  représente la valeur moyenne de  $E_{\text{test}}$ .
- La précision avec laquelle  $E_{\text{test}}$  approche  $E_{\text{gene}}$  dépend de la variance de  $E_{\text{test}}$ . Plus la variance est petite plus on a la garantie que l'estimation est meilleure.
- Il faut donc :
  - Choisir les exemples de l'ensemble Test en respectant au mieux la distribution des données  $p(x,y)$ .
  - Prendre le nombre  $p$  suffisamment élevé.

La variance de  $E_{\text{test}}$  diminue lorsque  $p$  augmente.



## Estimation des performances

- Comparaison de  $J_{app}(W, App)$  et  $J_{val}(W, Test)$  pour 2 architectures différentes et deux ensembles de tests différents



- Le réseau B apprend par cœur, il a de mauvaises performances sur la base de validation
- Le réseau A montre qu'il y a une relation presque linéaire entre les données et réalise de meilleures performances sur la base de validation.

## Choix du meilleur réseau

Unités cachées	$E_{app}$	$E_{test}$
1	0,012	0,0204
2	<b>0,0093</b>	<b>0,0147</b>
3	0,0077	0,0382
4	0,0043	0,0784
5	0,0019	0,1569

