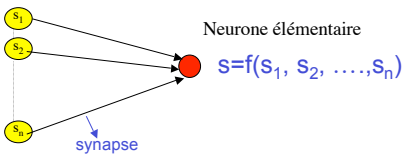
Réseaux de neurones
Multicouches

Partie 1

Neurone

Caractérisé par:

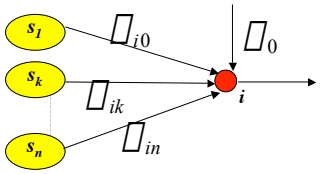
- état interne $s \in S$
- voisinage $s_1, \ldots, s_n$
- fonction de transfert f
- changement d'état $s=f(s_1, s_2, \ldots, s_n)$

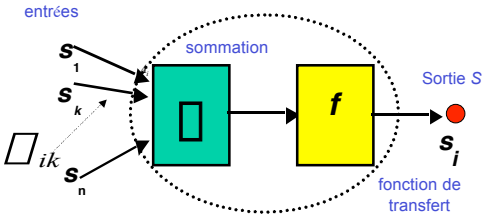


Exemples

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

Neurone produit scalaire

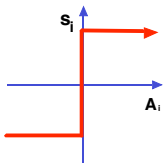




$$s_i = f(A_i)$$

$$A_i = \sum_{k=1}^{n} \theta_{ik} s_k + \theta_0$$

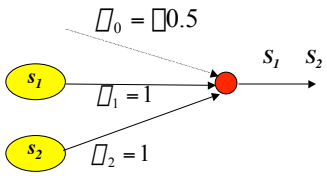
Le perceptron linéaire à seuil



S = {0,1} ou {-1,1}

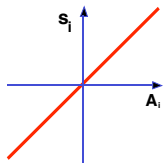
f(A) = 1 si A≥0
f(A) = 0 ou -1 si A<0

Exemple: **Perceptron qui calcule le OU**



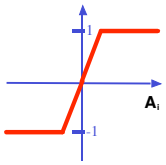
Autre fonctions de transfert

Linéaire



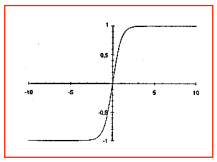
f = Id

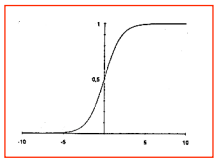
Quasi-linéaire



S = [-1,1]

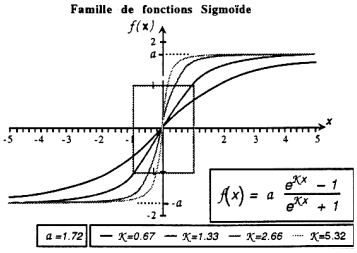
Fonction sigmoïde



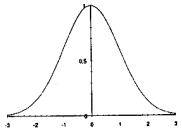


$$a\frac{1-e^{-kA}}{1+e^{-kA}} \qquad a\frac{1}{1+e^{-kA}}$$

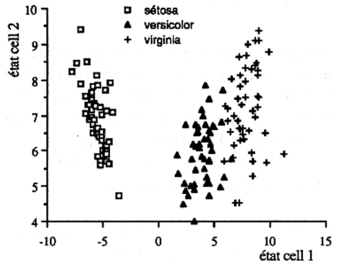
Famille de fonctions Sigmoïde



Fonction gaussienne (noyau)



Classification



- Connaissant la position d'un point, déterminer automatiquement sa classe.

•D'un point de vue géométrique: Déterminer les surfaces séparatrices entre les classes.

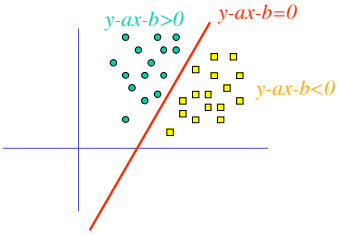
Solution:

Rechercher des surfaces séparatrices parmi une famille paramétrée donnée.

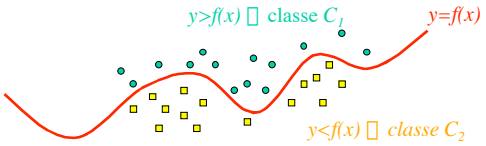
Apprentissage:

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

Quelle famille de surfaces choisir?



Cas simple : une séparation linéaire



Les surfaces séparatrices peuvent être complexes

Pourquoi des hyperplans?

•Souvent pas d'indication sur la forme des classes et donc sur celle des surfaces séparatrices.

• Calculs plus simples algorithmes facilement implémentables.

• Formalisation possible à l 'aide de l'algèbre linéaire

Si le problème n'est pas linéairement séparable les performances de la classification seront mauvaises

Exemple : 2 classes

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

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

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 :

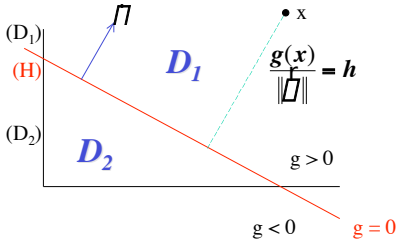
$$(H) \quad \sum_{i=1}^{n} w_i x_i + w_0 = 0$$

$$\vec{w} = \begin{pmatrix} w_1 \\ \\ \\ \\ w_n \end{pmatrix}$$

Vecteur orthogonal à H

Distance algébrique *h* de *x* à l'hyperplan

$$h = \frac{\sum_{i=1}^{n} \alpha_i x_i + \alpha_0}{\|\vec{\alpha}\|} \quad avec \|\vec{\alpha}\|^2 = \sum_{i=1}^{n} \alpha_i^2$$



•Si on pose : $g(x) = \sum_{i=1}^{n} \alpha_i x_i + \alpha_0$

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

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

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

g : fonction de décision

Notation : Vecteurs étendus $x \in \mathbb{R}^{n+1}$ $w \in \mathbb{R}^{n+1}$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix} \qquad \alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ \alpha_0 \end{pmatrix}$$

La fonction de décision s'écrit:

$$\mathbf{g(x) = x^T \alpha}$$

Décider

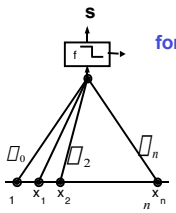
•C_1 *si* $g(x) = w^t x > 0$

•C_2 *si* $g(x) = w^t x <= 0$

Remarque : Si $\alpha_0 = 0$ Alors H passe par l'origine

Formulation neuronale :
Cas de 2 classes

Neurone à seuil



S = *f*(A) = 1 si A ≥ 0

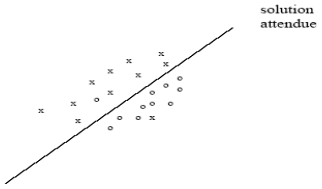
S = *f*(A) = -1 si A < 0

s = 1 si $\sum_{i+1}^{n} \theta_i x_i + \theta_0 \geq 0$

s = -1 si $\sum_{i+1}^{n} \theta_i x_i + \theta_0 < 0$

Problème :

Apprendre (Déterminer-Estimer) les paramètres (θ_0, θ_1 , …., θ_n) à partir d'un ensemble d'apprentissage : App



Réseaux Multicouches

MLP: Multi Layer Perceptron

PMC: Perceptron Multi-couche

Définition

•Les neurones sont réparties dans des couche successives (C_K) k=0, …, 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»**.

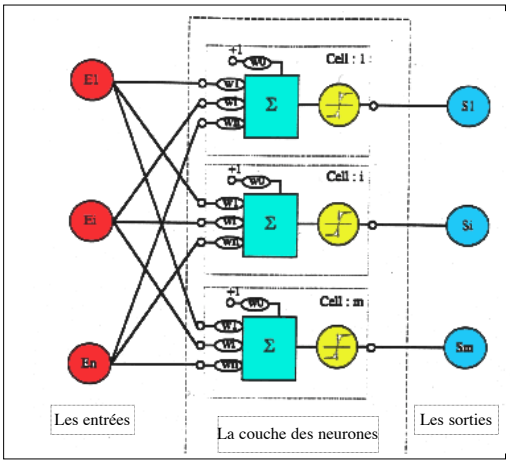
La couche C_k contient n_k neurones ($1 \leq k \leq C$)

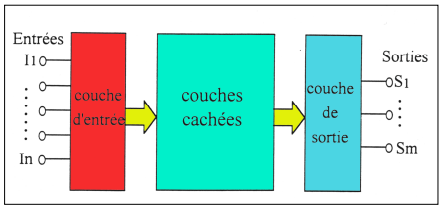
- Les états des neurones de C_0 sont dans .
- Les états des autres neurones sont en général dans [-*a*, *a*].

Les seules **connexions autorisées** sont d'une neurone de C_k à une 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 cachées ont des fonctions de transferts sigmoïdes.
• Les neurones de sorties, suivant les applications , ont des fonctions de transfert : sigmoïdes, linéaires, exponentielles.





DYNAMIQUE - PROPAGATION AVANT DES ETATS

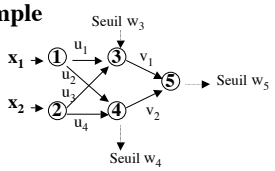
On présente un vecteur input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ à la couche d'entrée qui sera propagée d'une couche à une autre vers la couche de sortie.

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



Les neurones 3, 4 et 5 ont la même fonction de transfert f

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 $\Rightarrow G(.,W)$

$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{pmatrix}$ Entrée du réseau

Calcul les états : propagation de couche en couche.

$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \cdot \\ \cdot \\ y_p \end{pmatrix}$ Sortie calculée

•Fonction définie par un réseau

•Famille de fonction paramétrée par W

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

$G : \mathbb{R}^n \rightarrow \mathbb{R}^p$

Les fonctions de transferts

• Les neurones d'entrées n'ont pas de fonctions de transferts, leurs états étant imposés par l'extérieurs.

• Les neurones des couches cachées ont des fonctions de transferts sigmoïdes.

• Les neurones de sorties, suivant les applications , ont des fonctions de transfert : sigmoïdes, linéaires, exponentielles, …..

Apprentissage

On dispose d'un ensemble d'apprentissage :

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

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

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

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

$$\boldsymbol{y^K = G(x^K, W)}$$

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

⇩

Minimiser une fonction erreur $J(A_{pp}, W)$

Erreur Quadratique

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

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

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

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

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

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

$$W^* = \underset{W}{ArgMin}\ \varepsilon(W, App)$$

La minimisation se fait par une méthode de gradient

Algorithme de la rétro-propagation du gradient

➡ Détermination de

$$\varepsilon(W, App) = \sum_k J_k(W)$$

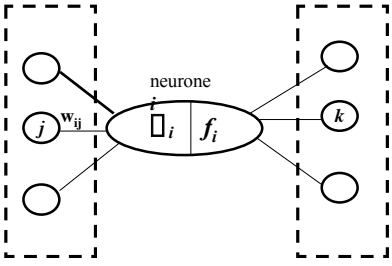
Erreur sur la k[ième] forme

➡ Calcul la dérivée de ε par rapport w_{ij} de :

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

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 $\frac{\partial J(W)}{\partial w_{ij}}$



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

Car $A_i = \sum_{j \in pred(i)} w_{ij} s_j$

On pose : $\frac{\partial J}{\partial A_i} = z_i$

On a : $\frac{\partial J}{\partial w_{ij}} = z_i s_j$ **(1)**

s_j (j) $\xrightarrow{w_{ij}}$ (i) z_i

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

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

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

(1) et (2)

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

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 *succ(i)*.

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

Comme

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

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

On tire alors :

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

Calcule 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 calcule 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*

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

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

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

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

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

Conclusion

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

Le calcule 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-propgation du gradient** afin de calculer les z_i des neurones.
- Pour chaque poids synaptique w_{ij} appliquer la formule (1).

Remarque : Le seuil θ_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 θ_i comme étant son poids synaptique.

Cas particulier :Classification

p classes $\Omega_1, \Omega_2, \ldots, \Omega_p$

$\underline{x}_k \in \Omega_k$ $d_k = (0,0,\ldots,1,0,\ldots 0)$

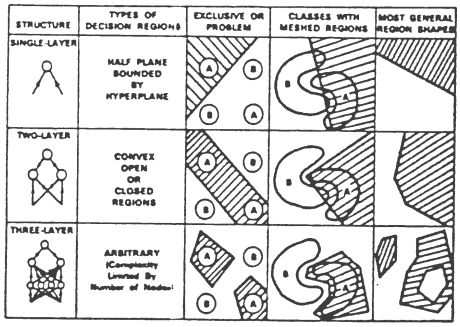
Ou $d_k = (\pm 1, \pm 1, \ldots, 1, \pm 1, \ldots \pm 1)$

Codage des classes

Classification

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

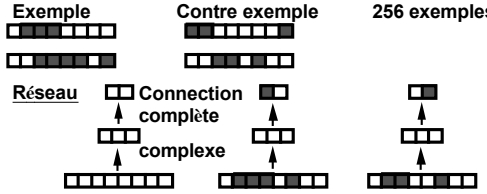
Détermination de frontières de décision complexes



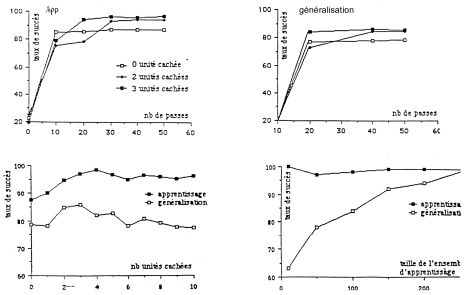
Mise au point pratique d'un réseau multicouche

Exemple 1 **Problème: détecter un évènement dans un signal "3 dans 8"**

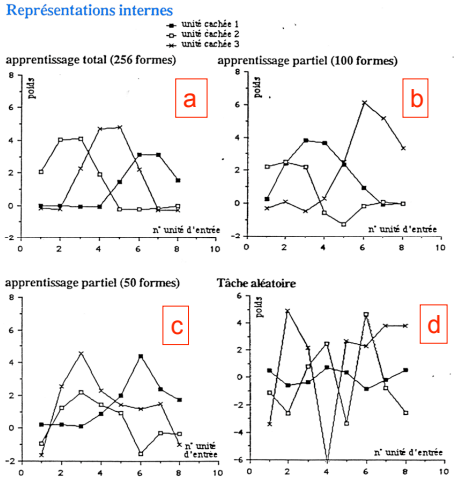
Exemple **Contre exemple** **256 exemples**



Courbes d'apprentissage en fonction du temps



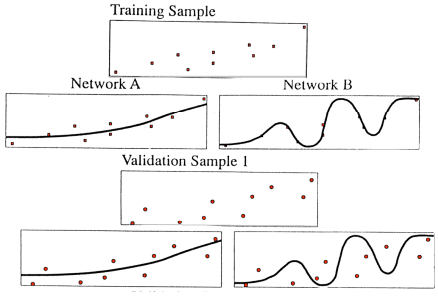
Représentations internes



a : Affecte à chaque neurone cachée une portion du signal.

Estimation des performances

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

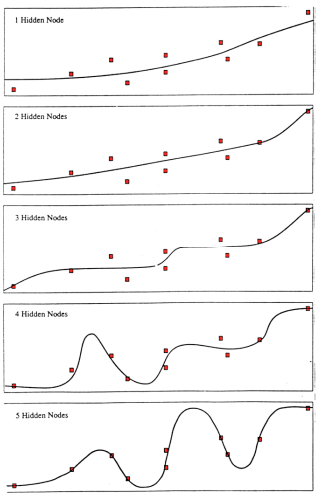


• Le réseau B apprend par cœur, il a de mauvaises performances sur la base de validation

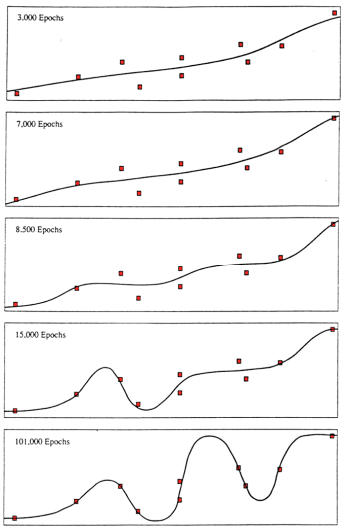
• Le réseau A montre qu'il y a une relation presque linéaire entre les données et réalise de meilleurs performances sur la base de validation.

Choix du meilleur réseau

Unités cachées	J(App,□)	J(Test,□)
1	0,0120	0,0240
2	0,0093	0,0147
3	0,0077	0,0382
4	0,0043	0,0784
5	0,0019	0,1569



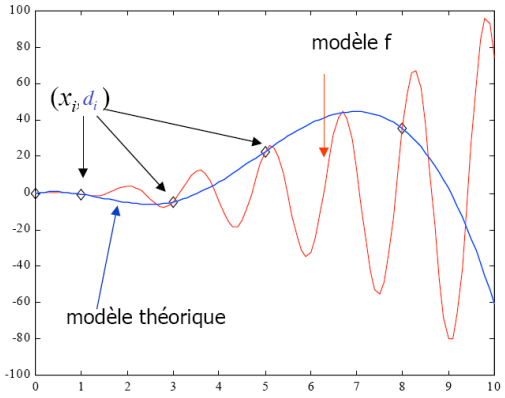
Évolution de la fonction pendant l'apprentissage



- Durant l'apprentissage, la « complexité » de la fonction augmente.
- Problème : Stopper l'apprentissage de manière à minimiser J(Test, □).

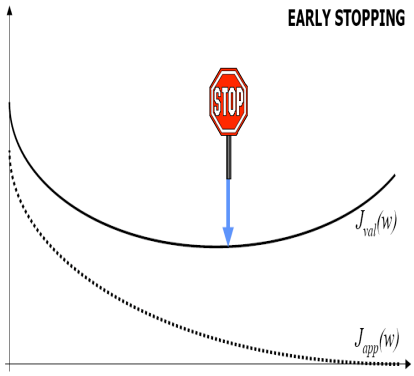
Le sur-apprentissage

Apprentissage "par coeur" : le modèle ne "connaît" que les points utilisés pour l'apprentissage et a un mauvais comportement ailleurs.

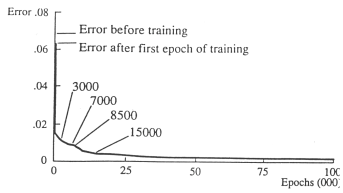


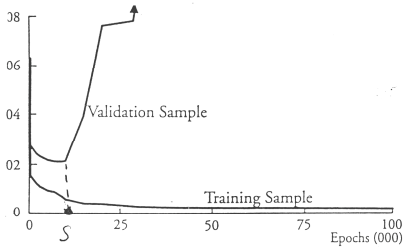
Validation

- ***On peut s'en servir pour arrêter l'apprentissage***

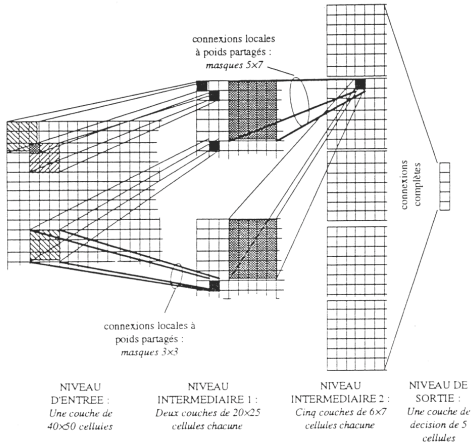


Arrêt de l'apprentissage





Architecture à masque



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

Reconnaissance de locuteur [Y. Bennani]

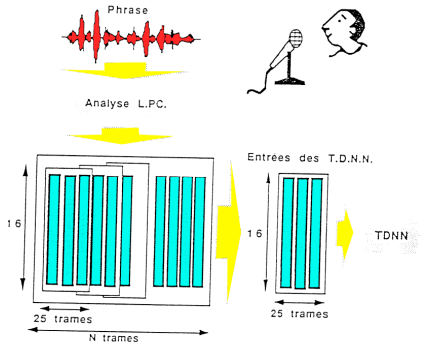
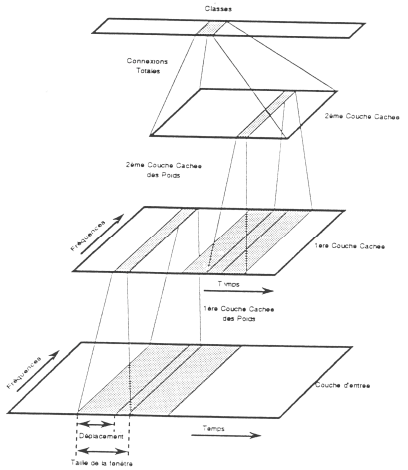


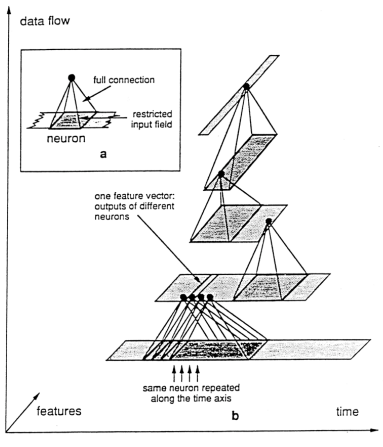
Figure 10.4 : Mécanisme du fenêtrage sur les phrases

Architecture TDNN (Time-Delay Neural Networks)



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

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



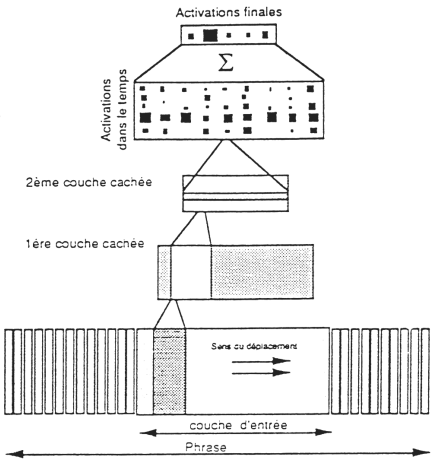


Figure 10.5 : Principe de Calcul d'activations dans le temps en phase d'identification pour un STDNN

Architecture TDNN

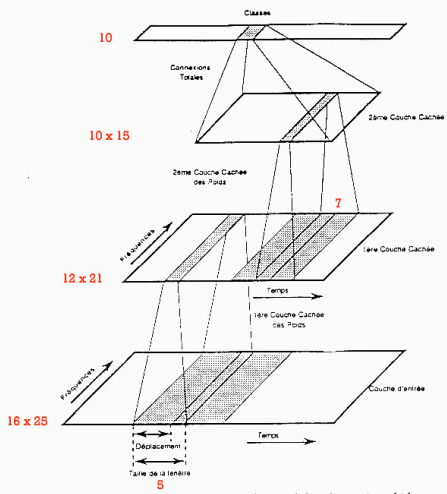


Figure 10.6 : Architecture des modules du système IAL

ACP des couches successives

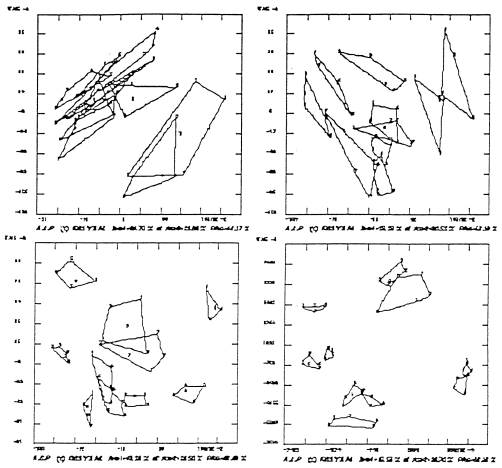


Figure 10.10 : ACP des couches successives du module M3