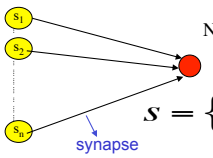
Réseaux de neurones
Multicouches

RAPPEL

Neurone

Caractérisé par:

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



Neurone élémentaire

$s=f(s_1, s_2, \ldots, s_n)$

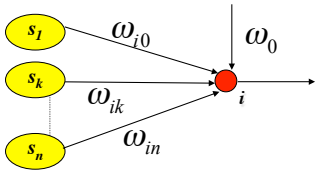
$s = \{0,1\}$

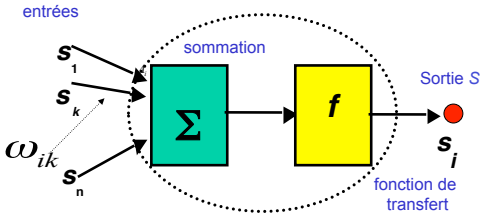
Exemples

- 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



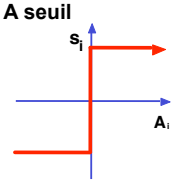


$$s_i = f(A_i)$$

$$A_i = \sum_{k=1}^{n} \omega_{ik} s_k + \omega_0$$

Fonctions de transfert

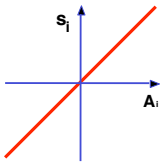
A seuil



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

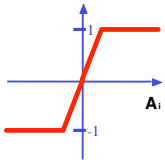
f(A) = 1 si A≥0
f(A) = 0 ou -1 si A<0

Linéaire



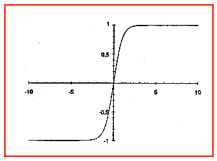
f = Id

Quasi-linéaire

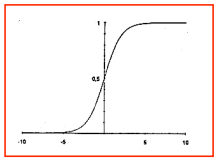


S = [-1,1]

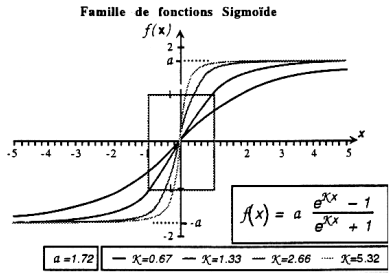
Fonctions sigmoïde



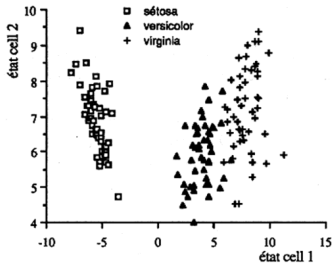
$$a\frac{1-e^{-kA}}{1+e^{-kA}}$$



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



Classification



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

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

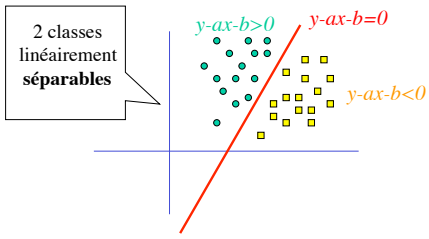
Solution:

Rechercher des surfaces séparatrices parmi une famille paramétrée donnée.

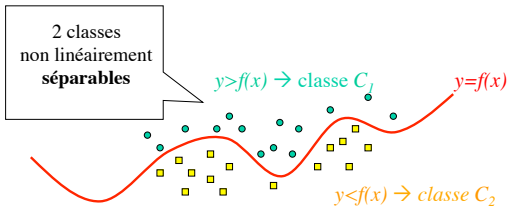
Apprentissage:

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

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»**.

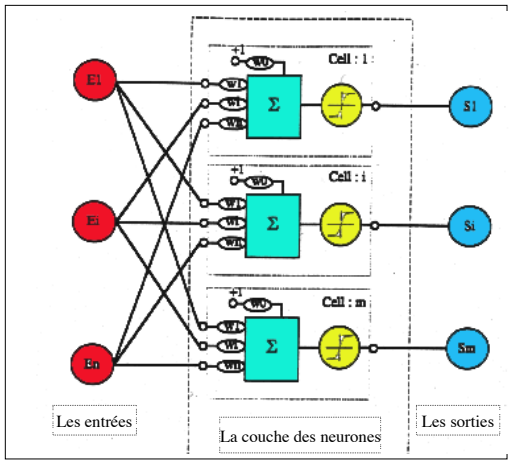
La couche C_k contient n_k neurones ($1 \leq k \leq C$)

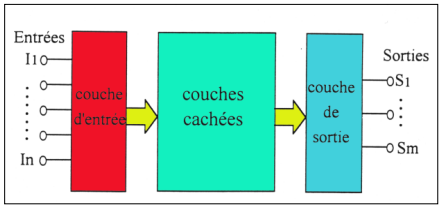
- Les états des neurones de C_0 sont dans $\Re$.
- Les états des autres neurones sont en général dans $[-a, a]$.

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 cachées ont des fonctions de transferts sigmoïdes.
• Les neurones de sorties, suivant les applications , ont des fonctions de transfert : sigmoïdes, linéaires, exponentielles ou autres





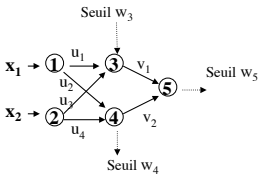
DYNAMIQUE - PROPAGATION AVANT DES ETATS

On présente un vecteur input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ 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



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 $\boldsymbol{W}$ l'ensemble des poids et des seuils.

Calculer les états : propagation de couche en couche.

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

Le réseau définit une famille de fonctions paramétrée par $\boldsymbol{W}$:

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

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

On note cette famille de fonctions par :

$$\mathbf{G}(.,\mathbf{W})$$

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.

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 $\boldsymbol{y}^k$:

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

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

⇩

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 :

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

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

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

La minimisation se fait par une méthode de gradient

Qui nécessite le calcule des : $\dfrac{\partial J(W)}{\partial w_{ij}}$

Algorithme de la rétro-propagation du gradient

➡ Détermination de

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

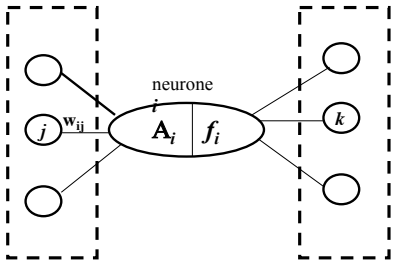
Erreur sur la k[ième] forme

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

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

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 : $\boxed{\frac{\partial J}{\partial w_{ij}} = z_i s_j}$ **(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* :**

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

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 :

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

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*

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

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

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 ($\boldsymbol{x}$, $\boldsymbol{d}$) de la base d'apprentissage.

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

- Présenter $\boldsymbol{x}$ 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 :

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

Remarque : Le seuil ω_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 ω_i comme étant son poids synaptique.