Chapitre II : Neurones Simples et Apprentissage

2. Neurone simple et problème de classification

Dans ce chapitre, nous allons, tout d'abord, présenter les types de neurones simples ainsi que quelques règles d'apprentissage de base, soutenus par des exemples. Une fois cette introduction effectuée, un aperçu sur la classification en utilisant les neurones présentés sera introduite avec quelques exemples de modélisation des fonctions logiques AND et OR.

2.1 Type de neurones simples

2.1.1 Le neurone de Mc Culloch-Pitt

Le neurone de Mc Culloch-Pitt est donné dans la Figure 2.1, il se caractérise principalement par une fonction d'activation de type limitation 0 ou 1 (s=1 ou 0).

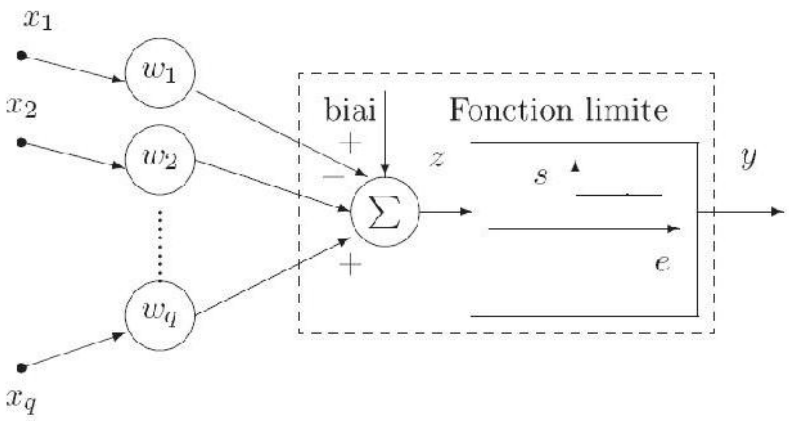


Figure 2.1 – Neurone de Mc Culloch-Pitt

2.1.2 Le Perceptron

Le perceptron, Figure 2.2, se caractérise par :

- Une fonction d'activation continue de type : $y = \frac{2}{1+e^{-\beta x}} - 1$.
- Fonction d'activation de type squashing, ou $-1 < y < 1$.
- La fonction d'activation est différentiable.

2.2 La règle d'apprentissage de Hebb

Hebb (1949) :

Quand l'axone de la cellule A est proche d'exciter la cellule B ceci se traduit par une décharge (fire), un changement métabolique ou évolution se produit dans l'une ou l'autre des cellules. En d'autres termes, l'e_cacité de déchargement de A sur B est augmentée.

Algorithme :

$$y = f(w^T x) \quad (2.1)$$
$$\Delta w = \mu y x = \mu f(w^T x) x \quad (2.2)$$

Et, pour le neurone *i* et l'entrée *j*, on a :

$$\Delta w_{ij} = \mu y x = \mu f(w_i^T x_j) x_j \quad (2.3)$$

Avec :

- w_0 est initialisé avec des valeurs aléatoires
- L'apprentissage est purement a propagation avant (feedforward) et non supervisé.
- Les poids w_i, seront fortement influencés par des entrées fréquentes
- Une évolution sans contraintes des poids w_i.
- Une règle de mise a jour basé sur la corrélation.

2.2.1 La règle de Hebb appliqué a un neurone de type signe

Soit le neurone de type signe suivant. Il est à noter que ceci est une version du perceptron avec $\beta \to \infty \Rightarrow f(w^T x) = sgn(w^T x)$.

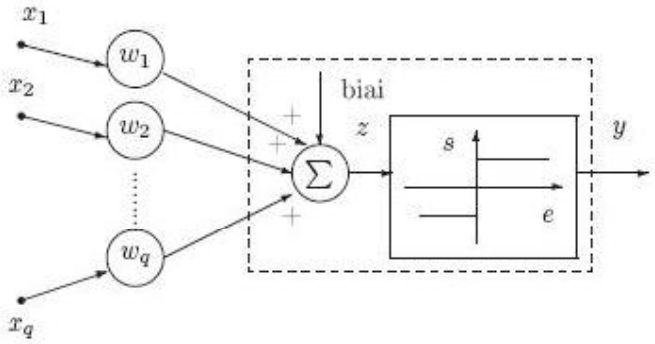


Figure 2.3 – Neurone de type signe

Soit : $w_0 = \begin{bmatrix} 1.0 \\ -1.0 \\ 0.0 \\ 0.5 \end{bmatrix}$ et $x_0 = \begin{bmatrix} 1.0 \\ -2.0 \\ 1.5 \\ 0.0 \end{bmatrix}$, $x_1 = \begin{bmatrix} 1.0 \\ -0.5 \\ -2.0 \\ -1.5 \end{bmatrix}$, $x_2 = \begin{bmatrix} 0.0 \\ 1.0 \\ -1.0 \\ 1.5 \end{bmatrix}$

Si on prend $\mu = 1$, pour des raisons simplificatives, on obtient :

$$z_0 = w_0^T x_0 = [1.0 \ \ -1.0 \ \ 0.0 \ \ 0.5] \begin{bmatrix} 1.0 \\ -2.0 \\ 1.5 \\ 0.0 \end{bmatrix} = 3$$

La mise a jour des poids ($w_{k+1} = w_k + \Delta w$) se fait comme suit :

$$w_{k+1} = w_k + sgn(z_k)x_k \tag{2.4}$$

D'ou :

$$w_1 = \begin{bmatrix} 1.0 \\ -1.0 \\ 0.0 \\ 0.5 \end{bmatrix} + 1. \begin{bmatrix} 1.0 \\ -2.0 \\ 1.5 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3.0 \\ 1.5 \\ 0.5 \end{bmatrix}$$

Pour les deux prochaine itérations :

$$z_1 = -0.25,\ w_2 = w_1 + sgn(z_1)x_1 = w_1 - x_1 = \begin{bmatrix} 1.0 \\ -2.5 \\ 3.5 \\ 2.0 \end{bmatrix}$$

$$z_2 = -3.0,\ w_3 = w_2 + sgn(z_2)x_2 = w_2 - x_2 = \begin{bmatrix} 1.0 \\ -3.5 \\ 4.5 \\ 0.5 \end{bmatrix}$$

La règle de Hebb utilisé avec un neurone signe, revient à rajouter ou soustraire l'entrée et le vecteur poids. Si bien sur $\mu = 1$.

2.2.2 Règle de Hebb et neurone de type Perceptron

Soit le neurone de type perceptron Figure 2.2, avec $\beta = 1, => \ f(w^T x) = y = 2/(\ 1 + e^z\ )\ - 1.$

Soit : $w_0 = \begin{bmatrix} 1.0 \\ -1.0 \\ 0.0 \\ 0.5 \end{bmatrix}$ et $x_0 = \begin{bmatrix} 1.0 \\ -2.0 \\ 1.5 \\ 0.0 \end{bmatrix}$, $x_1 = \begin{bmatrix} 1.0 \\ -0.5 \\ -2.0 \\ -1.5 \end{bmatrix}$, $x_2 = \begin{bmatrix} 0.0 \\ 1.0 \\ -1.5 \\ 1.5 \end{bmatrix}$

Si on prend $\mu = 1$, pour des raisons simplificatives, on obtient :

$$z_0 = w_0^T x_0 = [1.0 \ \ -1.0 \ \ 0.0 \ \ 0.5] \begin{bmatrix} 1.0 \\ -2.0 \\ 1.5 \\ 0.0 \end{bmatrix} = 3$$

La mise a jour des poids se fait comme suit :

$$w_{k+1} = w_k + f(z_k)x_k, \quad y = f(z) = y = \frac{2}{1 + e^{-\beta z}} - 1 \tag{2.5}$$

D'ou :

$$w_1 = \begin{bmatrix} 1.0 \\ -1.0 \\ 0.0 \\ 0.5 \end{bmatrix} + 0.905 \begin{bmatrix} 1.0 \\ -2.0 \\ 1.5 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 1.905 \\ -2.810 \\ 1.357 \\ 0.500 \end{bmatrix}$$

Pour les deux prochaines itérations :

$$y_1 = -0.077, \ w_2 = w_1 + 0.077x_1 = \begin{bmatrix} 1.828 \\ -2.772 \\ 1.512 \\ 0.616 \end{bmatrix}$$

$$y_2 = -0.932, \ w_3 = w_2 + 0.932x_2 = \begin{bmatrix} 1.828 \\ -3.700 \\ 2.440 \\ -0.783 \end{bmatrix}$$

La règle de Hebb utilisé avec un neurone continue de type Perceptron, revient à rajouter ou soustraire, une fraction (déterminé par β) l'entrée et le vecteur poids.

2.3 Règle d'apprentissage du Perceptron

La regle d'apprentissage du perceptron (Rosenblatt 1958) est caractérisé par :

- La règle est applicable aux neurones de type signe.
- Le signal d'apprentissage est la différence entre l'actuelle et la réponse désirée.
- La nécessité d'un signal désiré implique un apprentissage supervisé.

Soit le perceptron, avec le signal d'apprentissage d :

Algorithme :

$$e = d - y, \quad y = sgn(w^T x) \tag{2.6}$$

$$\Delta w = \mu e x = \mu[d - sgn(w^T x)]x \tag{2.7}$$

Etant donné que d=1 ou -1,

$$\Delta w = \pm 2\mu x \tag{2.8}$$

Exemple :

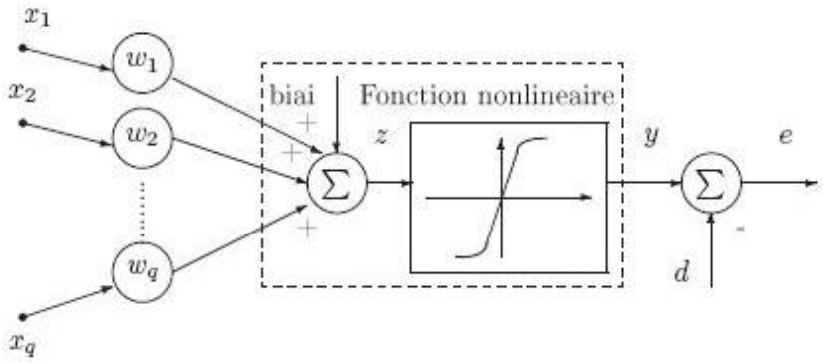


Figure 2.4 – Neurone utilisé avec la régle du Perceptron

Soit : $w_0 = \begin{bmatrix} 1.0 \\ -1.0 \\ 0.0 \\ 0.5 \end{bmatrix}$, $d_0 = -1.0$, $d_1 = -1$, $d_2 = 1$,

et $x_0 = \begin{bmatrix} 1.0 \\ -2.0 \\ 0.0 \\ -1.0 \end{bmatrix}$, $x_1 = \begin{bmatrix} 0.0 \\ 1.5 \\ -0.0 \\ -1.0 \end{bmatrix}$, $x_2 = \begin{bmatrix} -1.0 \\ 1.0 \\ 0.5 \\ -1.0 \end{bmatrix}$

$$z_0 = w_0^T x_0 = [1.0 \ \ -1.0 \ \ 0.0 \ \ 0.5] \begin{bmatrix} 1.0 \\ -2.0 \\ 0.0 \\ -1.0 \end{bmatrix} = 2.5$$

A noter que, $sgn(2.5) \neq d_0$. Si on prend $\mu = 0.1$, on obtient :

$$w_1 = w_0 + 0.1(-1 - 1)x_0$$

$$w_1 = \begin{bmatrix} 1.0 \\ -1.0 \\ 0.0 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 1.0 \\ -2.0 \\ 1.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0.0 \\ 0.7 \end{bmatrix}$$

$$z_1 = w_1^T x_1 = [0.0 \ \ 1.5 \ \ -0.5 \ \ -1.0] \begin{bmatrix} 0.8 \\ -0.6 \\ 0.0 \\ 0.7 \end{bmatrix} = -1.6$$

A noter que, $sgn(-1.6) = d_1$. La mise a jour des poids n'est pas necessaire.

$$z_2 = w_2^T x_2 = [1.0 \ \ -1.0 \ \ 0.5 \ \ -1.0] \begin{bmatrix} -1.0 \\ 1.0 \\ 0.5 \\ -1.0 \end{bmatrix} = -0.75$$

A noter que, $sgn(-0.75) = d_1$. La mise a jour des poids est necessaire.

$$w_3 = w_2 + 0.1(1+1)x_2$$

$$w_3 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0.0 \\ 0.7 \end{bmatrix} + 0.2 \begin{bmatrix} 1.0 \\ -1.0 \\ 0.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

Remarques :

Il n'ya pas de justification évidente pour l'inclusion du vecteur d'entré, *x*, dans l'équation (2.7).
Le choix du coefficient d'apprentissage *1*, n'est pas spécifié.
La règle du perceptron peut être résumé à ce qui suit :

Avec un neurone de Mc Culloch-Pitt
$w_{k+1} = w_k + \mu x$ si $y = 0$ et $d = 1$
$w_{k+1} = w_k - \mu x$ si $y = 1$ et $d = 0$
$w_{k+1} = w_k$ si $y = d$

Avec un neurone de type signe (perceptron avec $\beta \rightarrow 1$)
$w_{k+1} = w_k + 2\mu x$ si $y = 0$ et $d = 1$
$w_{k+1} = w_k - 2\mu x$ si $y = 1$ et $d = 0$
$w_{k+1} = w_k$ si $y = d$

Il peut être prouvé que le vecteur poids, *w*, converge.

2.3.1 Convergence de la règle du perceptron

Théoreme : S'il existe un vecteur de poids, qui classifie correctement l'ensemble d'apprentissage (supposé linéairement séparable), alors la règle d'apprentissage trouvera un, et un seul, vecteur de poids, $w^{¤}$, dans un nombre fini d'itérations.

Assomptions :

- Il existe au moins un vecteur de poids $w^{¤}$.
- Il existe un nombre _ni de vecteurs d'apprentissage.
- La fonction d'activation est de type unipolaire (0 ou 1).

Preuve :

A la *k* ieme itération, on a :

$$w_{k+1} = w_k + \mu e_k x_k, \quad e_k = d_k - y_k \tag{2.9}$$

Si on soustrait w^* des deux cotés de l'équation (2.9), on obtient :

$$w_{k+1} - w^* = w_k - w^* + \mu e_k x_k \tag{2.10}$$

Si y_k est correctement classé, alors il n'a pas de mise a jour. Si on prend la distance euclidiénne de l'equation (2.10), on obtient :

$$\|w_{k+1} - w^*\|^2 = \|w_k - w^*\|^2 + \mu^2\|x_k\|^2 + 2\mu e_k(w_k - w^*)^T x_k \tag{2.11}$$

Il peut etre prouvé que pour our chaque vecteur, x_k, non classé. on a :

$$e_k w^T x_k = -|w^T x_k| \leq 0 \tag{2.12}$$

et aussi :

$$e_k w^{*T} x_k - |w^{*T} x_k| \geq 0 \tag{2.13}$$

D'ou :

$$\|w_{k+1} - w^*\|^2 = \|w_k - w^*\|^2 + \mu^2\|x_k\|^2 - 2\mu\left(|w^{*T} x_k| + |w^T x_k|\right) \tag{2.14}$$

Si on choisi un pas d'apprentissage suffisament petit,

$$\mu^2 \ll \mu \tag{2.15}$$

L'équation (2.14), peut être ramené a :

$$\|w_{k+1} - w^*\|^2 \approx \|w_k - w^*\|^2 - 2\mu\left(|w^{*T} x_k| + |w^T x_k|\right) \tag{2.16}$$

Sachant que $\mu \geq 0$, et que $|w^{*T}x_k|+|w^T x_k| \geq 0$, doit décroître au fil des itérations. Toutefois, $\|.\|$ne peut pas prendre de valeurs négatives, elle doit donc converger dans un nombre fini d'itérations. A noter que cette norme ne dois pas nécessairement converger vers 0 ($w = w^*$).

Le pas d'apprentissage idéal :

Si on réécrit l'équation (2.14) en terme d'erreur, $\varepsilon = w - w^*$ on obtient :

$$\|\varepsilon_{k+1}\|^2 = \|\varepsilon_k\|^2 + \mu^2\|x_k\|^2 - 2\mu\left(|w^{*T} x_k| + |w^T x_k|\right) \tag{2.17}$$

Si on minimise, ε_k par rapport a μ, on obtient :

$$\frac{\partial}{\partial\mu}\|\varepsilon_{k+1}\|^2 = \frac{\partial}{\partial\mu}\left(\|\varepsilon_k\|^2 + \mu^2\|x_k\|^2 - 2\mu\left(|w^{*T} x_k| + |w^T x_k|\right)\right) \tag{2.18}$$

$$0 = 2\mu\|x_k\|^2 - 2\left(|w^{*T} x_k| + |w^T x_k|\right) \tag{2.19}$$

$$\mu_{opt} = \frac{|w^{*T} x_k| + |w^T x_k|}{\|x_k\|^2} = \frac{|(w^* + w_k)^T x_k|}{\|x_k\|^2} \tag{2.20}$$

En pratique, il n'est pas possible d'évaluer μ_{opt}, car cette dernière contient w^{α} dans sa formulation. Et il est souvent suffisant d'interpréter μ_{opt} comme la distance optimale qui puisse ramener w_k au long de la direction de x_k.

2.4 Classification et séparabilité linéaire

2.4.1 Classification :

Le but de la classification (pattern classification) est d'assigner un objet physique, évènement ou phénomène a une classe (ou catégorie) préalablement défini. On peut citer comme exemples :

- Fonction booléennes.
- Pattern de pixels, e.x. afficheur 7 segments.
- Quantification de vecteurs, e.x. Transformation Analog/digital.
- Mémoire associative e.x. reconnaissance de caractères.

2.4.2 Fonctions discriminantes :

Problème :

Supposons qu'il existe un vecteur de forme (patterns) a p dimensions : $x_1, x_2.., x_p$ et que la classification de chaque forme (élément) est connue, il existe n classes. La dimension du vecteur des éléments p est généralement plus importante que le nombre de classes n. Il est aussi raisonnable d'assumer que n est plus important que le nombre de catégories R.
L'appartenance a une catégorie donnée, est déterminé par le classifieur basée sur la comparaison de R fonctions discriminantes $g_1(x)$; $g_2(x)$; ... ; $g_R(x)$ calculé pour le vecteur d'entré x. L'élément x appartient a la catégorie i si et seulement si :

$$g_i(x) > g_j(x) \quad for \quad i,j = 1,2,\cdots,R, \quad i \neq j \tag{2.21}$$

L'équation definissant la frontiére de la décision est :

$$g_i(x) - g_j(x) = 0 \tag{2.22}$$

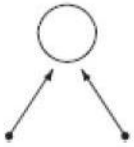
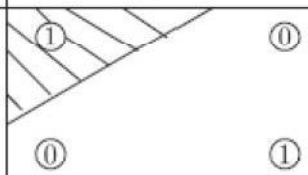
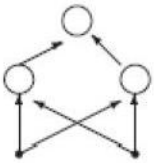
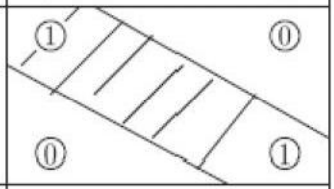
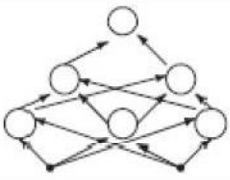
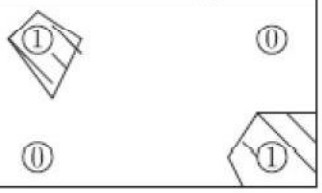
Structure	Type de region de decision	Probleme XOR
	Demi-plan Limite par un Hyperplan	
	Regions convexes Ouverte ou Ferme	
	Complexite limite par le nombre de neurones	

Figure 2.5 – Classification de la fonction XOR par RNA

2.4.3 Séparabilité linéaire

Un neurone de type signe (threshold) implémente la séparation d'un vecteur d'entré en deux classes, ou la frontière entre ces deux classes est défini par :

$$W^T X = -b \qquad (2.23)$$

Ou b est la limite ou le biais. La frontière séparant les deux classes est un hyperplan a n dimensions.

Remarque :

La proportion de fonctions linéaires séparables dans le domaine de n fonctions booléennes décrois exponentiellement avec n.

Un réseau comportant deux couches de neurones de type signe, est capable de calculer toutes les fonctions booleennes par une implémentation de sommes de produits, Figure
2.5.

2.4.4 Exemple : Implémentation de la fonction AND

La fonction AND est implémenté sous Matlab en utilisant les fonction newp pour créer un RNA, et la fonction train pour l'apprentissage du réseau a neurone unitaire de type Perceptron.

Les détails de l'implémentation sont donnés par :

2 entrées + 1 biais, avec une initialisation $w(0) = [0.1\ 0.1]$ et $b(0) = 0.1$.
Apprentissage de 8 époques avec un pas d'apprentissage de 0.1.
La surface de décision est donnée pas :

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

$$P=\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}=\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, T = [0 \quad 0 \quad 0 \quad 1]$$

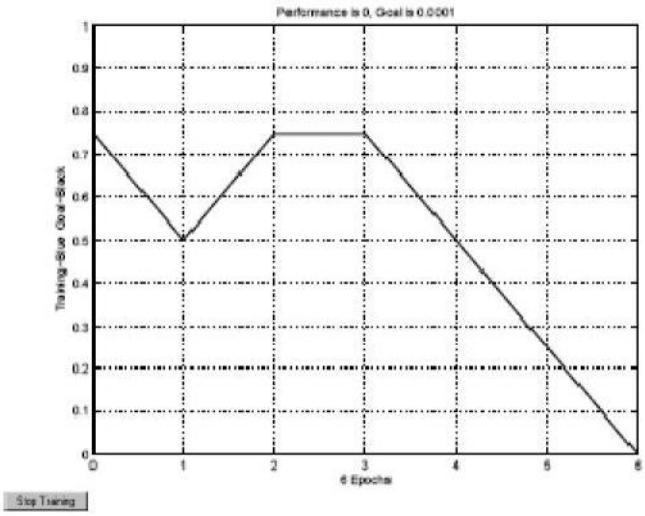
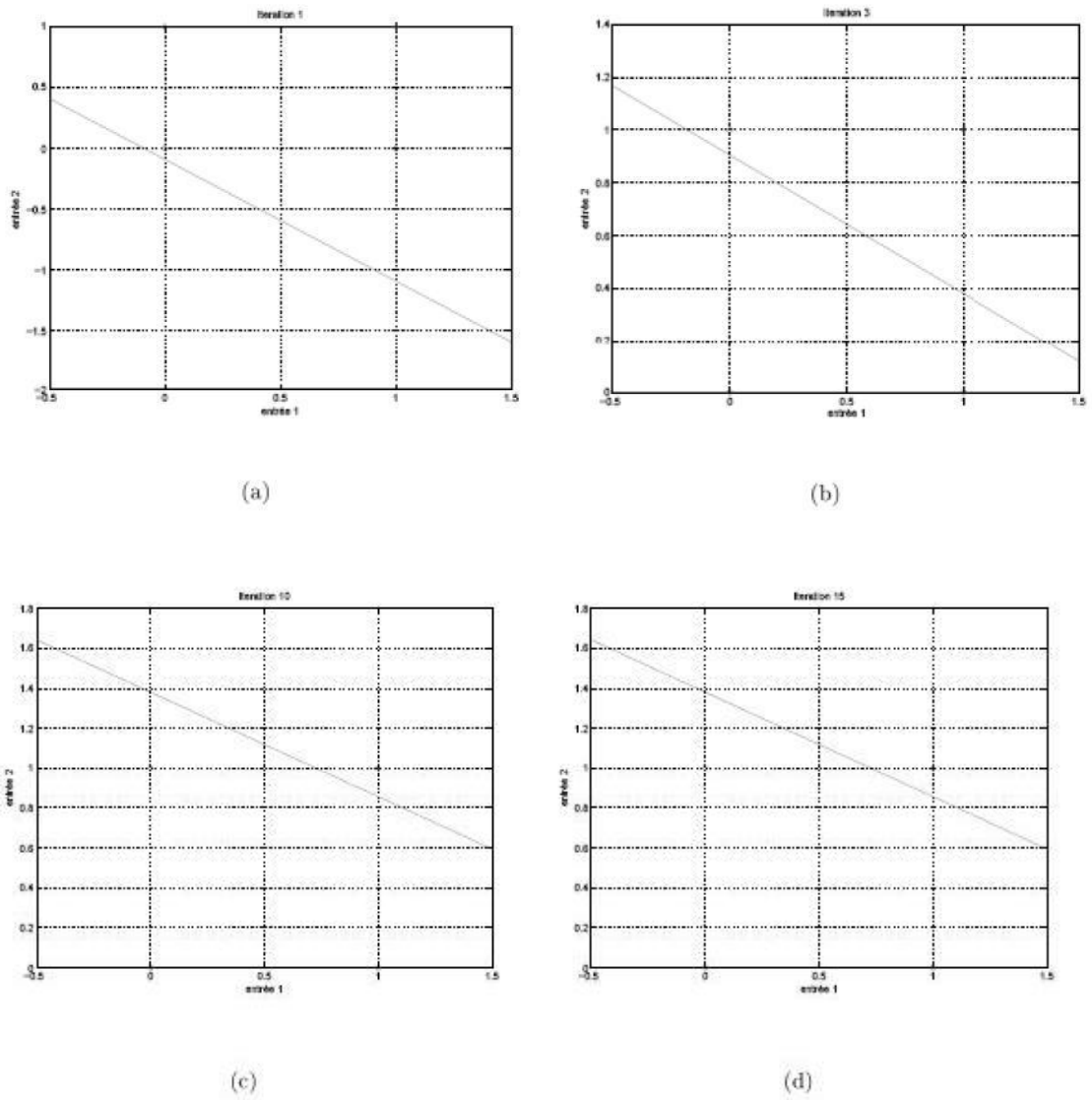


Figure 2.6 – Apprentissage du perceptron pour l'implementation de la fonction AND

On peut voir dans la Figure 2.6, que le but défini au départ (une erreur EMQ=0.0001) est obtenu après un apprentissage qui a duré 6 époques. La Figure 2.7 montre l'évolution de la droite de décision (séparation) durant l'apprentissage, allant de l'époque 1 a l'époque 15 (Figure 2.7(a),(b),(c) et (d)). On peut voir que la ligne séparation, d'ou l'implémentation de la porte AND, est obtenue clairement après l'itération 6. Le problème de séparabilité est résolue avec un neurones. Il est a noter que un neurone, ne peut résoudre le problème donnée par le OU exclusif XOR, car une seule droite ne peut séparer les solutions.



(a) (b)

(c) (d)

Figure 2.7 – Apprentissage et separabilite pour la fonction AND

2.7 L'Algorithme d'apprentissage LMS

En général l'équation de mise un jour pour un système multidimensionnel, équation (2.46), ne peut toujours être implémenté, car le vrai gradient dépend de W_k, Donc une estimation du gradient doit être utilisé.

Solution 1 :

Utiliser une moyenne de la EMQ sur un horizon moyen, et prendre les différences fini de cette dernière.
Le principal désavantage, est qu'on doit attendre qu'un certain nombre d'échantillons d'erreur soit collecté, avant d'être utilisé pour la mise à jour des poids.

Solution 2 :
Au lieu de la l'erreur moyenne quadratique, n'utiliser que l'erreur quadratique EQ :

$$\hat{\xi} = e_k^2 \tag{2.48}$$

Maintenant le gradient estimé est :

$$\hat{\nabla}_k = \begin{bmatrix} \frac{\partial e_k^2}{\partial w_0} \\ \frac{\partial e_k^2}{\partial w_1} \\ \vdots \\ \frac{\partial e_k^2}{\partial w_L} \end{bmatrix} = 2e_k \begin{bmatrix} \frac{\partial e_k}{\partial w_0} \\ \frac{\partial e_k}{\partial w_1} \\ \vdots \\ \frac{\partial e_k}{\partial w_L} \end{bmatrix} = -2e_k X_k \tag{2.49}$$

L'équation de mise a jour devient alors :

$$W_{k+1} = W_k + \mu(-\hat{\nabla}) \tag{2.50}$$

$$W_{k+1} = W_k + 2\mu e_k X_k \tag{2.51}$$

L'équation (2.51) est l'algorithme LMS. L'estimation du gradient, équation (2.46), et la convergence de mise a jour du vecteur poids, équation (2.51) peuvent être démontrée.

2.7.1 L'adaline

L'adaline est un type de neurone inspiré du type signe auquel on rajoute une différence entre la sortie linéaire *z* et une sortie désirée *d*, comme le montre la Figure 2.10.

Caractéristiques :

- La somme linéaire *z*, est utilisée via l'algorithme LMS. Après apprentissage la fonction d'activation du type échelon est rajoutée.
- Les poids sont corrigés par une valeur proportionnelle a la différence entre la sortie actuelle et celle désiré.
- L'adaline et l'algorithme LMS, peuvent donner une solution dans le cas ou le perceptron, utilisant l'apprentissage par perceptron ne converge pas.
-

2.7.2 La règle d'apprentissage delta

La règle d'apprentissage delta, introduite par McClelland and Rumelhart (1986), utilise l'algorithme LMS avec un neurone de type perceptron :

Caractéristiques :

- Utilise seulement un neurone avec une fonction d'activation continue et différentiable
- Une méthode basée sur le gradient
- Peut être généralisé au perceptron multi-couches (PMC)

- Utilise la même approximation de la EMQ que l'algorithme LMS, c.a.d, EQ
- Pas de mesure de stabilité pour μ

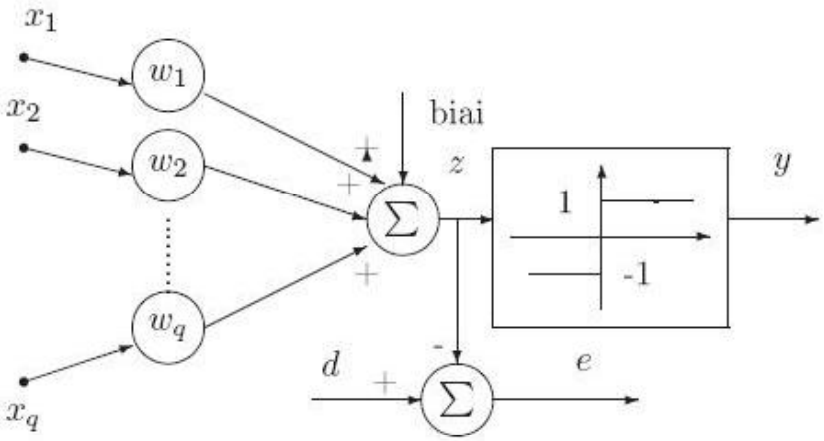


Figure 2.10 – Adaline

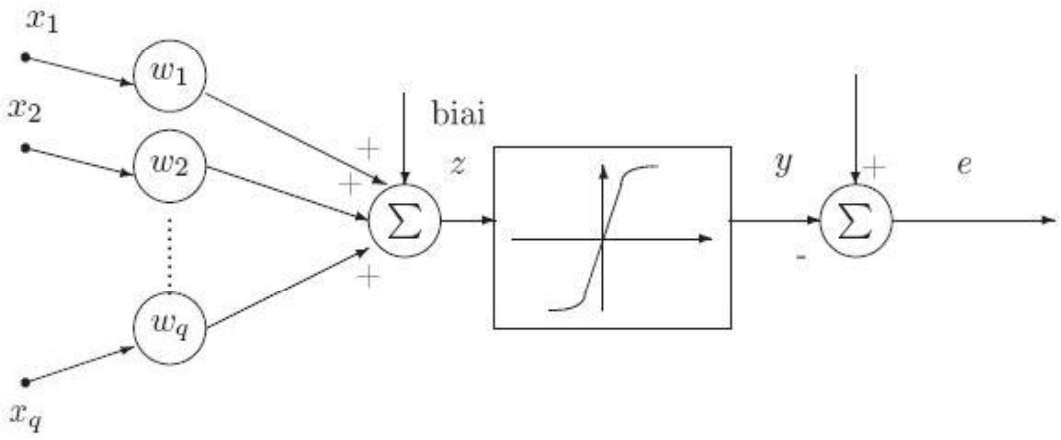


Figure 2.11 – Neurone utilisé avec la régle delta

2.7.3 Dérivation de la règle d'apprentissage delta

Pour le perceptron continu on a :

$$y = f(z), \quad z = W^T X, \quad e = d - y \tag{2.52}$$

ou :

$$f_l(z) = \frac{1}{1 + e^{-\beta z}}, \quad f_t(z) = \frac{2}{1 + e^{-\beta z}} - 1 \tag{2.53}$$

avec, f_l et f_t correspondant respectivement au fonctions log-sigmoidal et tangente-sigmoidal.
Le gradient de l'erreur au carré est évalué comme :

$$\nabla = \frac{\partial e^2}{\partial W} = -2(d - f(W^T X)) f'(W^T X) X \qquad (2.54)$$

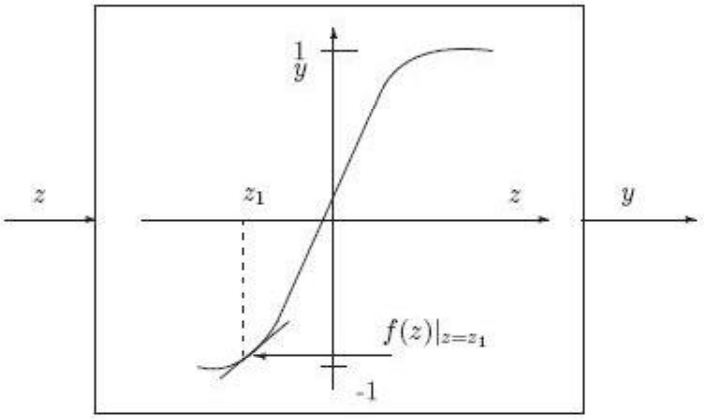
La mise a jour des poids se fait :

$$\triangle W = -\mu \nabla \qquad (2.55)$$

d'ou résulte la régle delta comme :

$$W_{k+1} = W_k + 2\mu e f'(z) X \qquad (2.56)$$

Ou $f'(z)$ est la dérivée de $f(.)$ evalué au point d'opération correspondant a z, comme le montre la Figure 2.7.3.



2.7.4 Evaluation des dérivatives

Étant donné que les poids sont recalables, on peut assumer que $\beta = 1$ sans perdre la notion de généralité de ce qui suit :

Pour la fonction Sigmoid unipolaire (Logsigmoidal) on a :

$$y = f_l(z) = \frac{1}{1 + e^{-z}} \qquad (2.57)$$

$$f_l'(z) = \frac{e^{-z}}{(1 + e^{-z})^2} \qquad (2.58)$$

A partir de l'équation (2.57) on a :

$$1 + e^{-z} = \frac{1}{y} \qquad (2.59)$$

et

$$e^{-z} = \frac{1}{y} - 1 = \frac{1 - y}{y} \qquad (2.60)$$

Ceci donne :

$$f_l'(z) = y(1 - y) \qquad (2.61)$$

Pour la fonction Sigmoid bipolaire (tangente-sigmoidal) on a :

$$y = f_t(z) = \frac{2}{1 + e^{-z}} - 1 \tag{2.62}$$

$$f_t'(z) = \frac{2e^{-z}}{(1 + e^{-z})^2} \tag{2.63}$$

A partir de l'équation (2.62) on a :

$$1 + e^{-z} = \frac{2}{1 + y} \tag{2.64}$$

et

$$e^{-z} = \frac{2}{1 + y} - 1 = \frac{1 - y}{1 + y} \tag{2.65}$$

Ceci donne :

$$f_t'(z) = \frac{1}{2}(1 - y^2) \tag{2.66}$$

En remplacant dans l'équation de mise a jour des poids (2.56) on obtient, por la fonction sigmoid bi-polaire *(tan-sigmoid)* :

$$W_{k+1} = W_k + 2\mu e Y(1 - Y)X \tag{2.67}$$

pour la fonction sigmoid unipolaire *(log-sigmoid)* :

$$W_{k+1} = W_k + \mu e(1 - Y^2)X \tag{2.68}$$