En pratique, il n'est pas possible d'évaluer μ_{opt}, car cette dernière contient $w¤$ 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 frontière de la décision est :

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

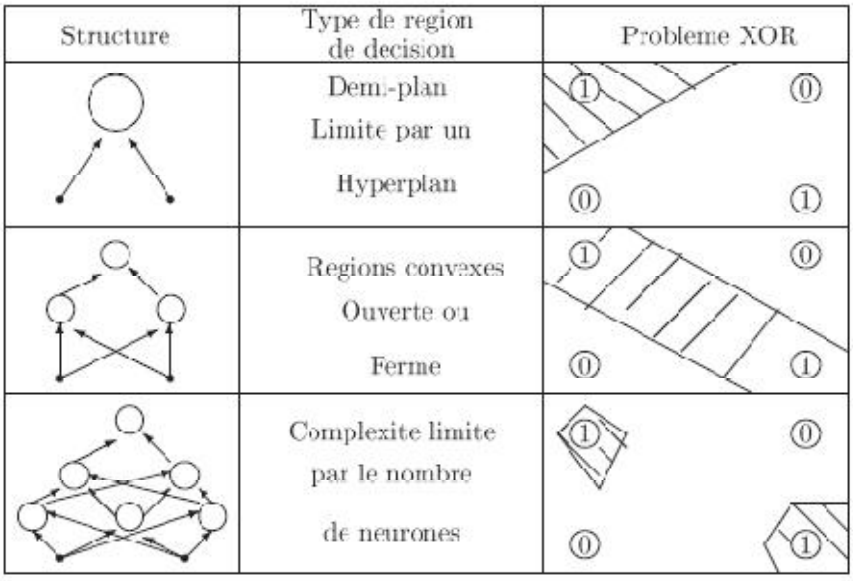


Figure 2.5 – Classification de la fonction XOR par RNA

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 :

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

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

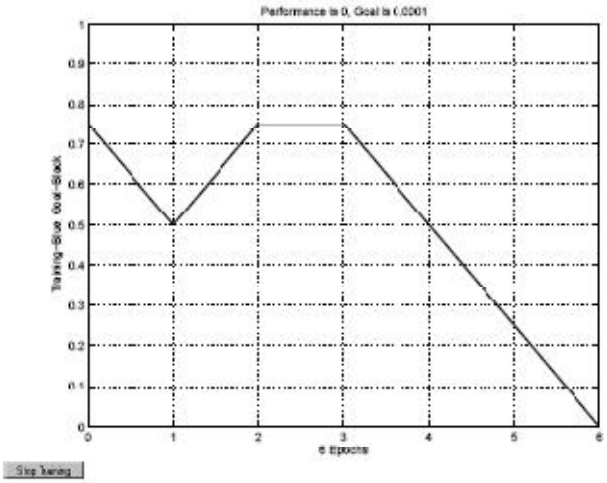
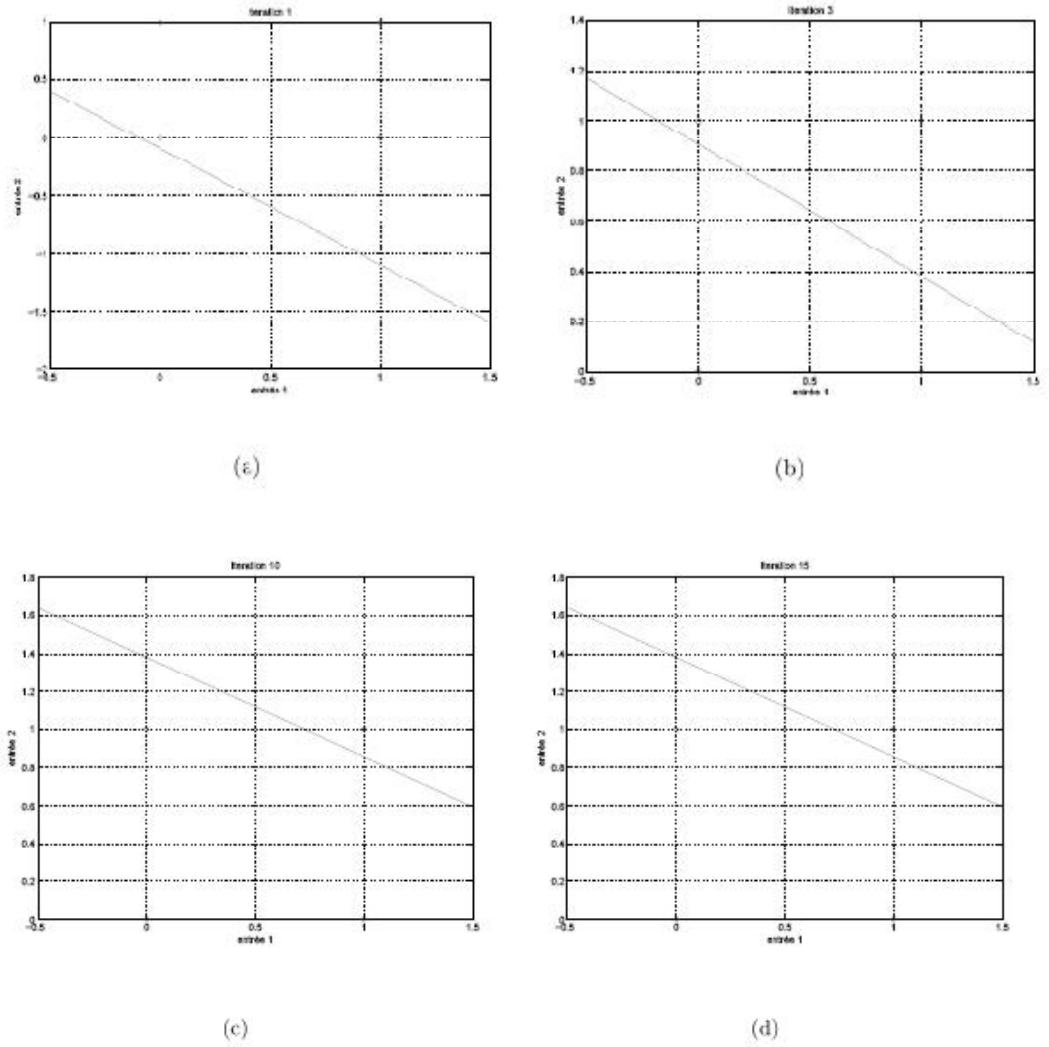


Figure 2.6 – Apprentissage du perceptron pour l'implementation de la fonction AND

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¤, 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 a 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

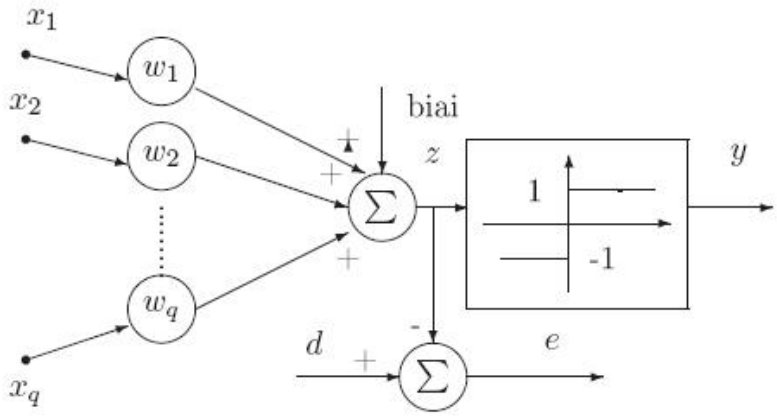


Figure 2.10 – Adaline

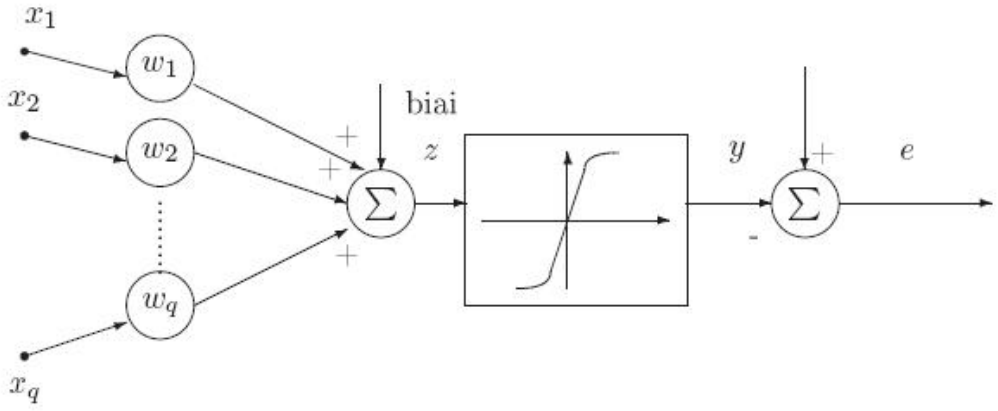


Figure 2.11 – Neurone utilisé avec la régle delta

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 :

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

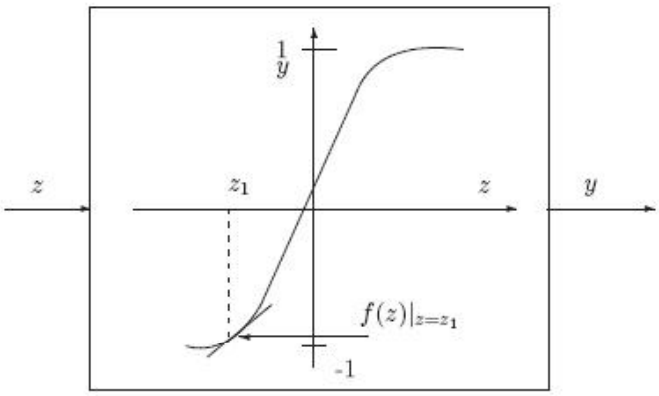
La mise a jour des poids se fait :

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

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)* :

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

3. Neurone simple et problème de classification

3.1 Introduction

Il a été prouve par Cybenko qu'une fonction continue quelconque peut être arbitrairement bien approximée par un réseau feed-forward à couche cachée unique, ou chaque neurone dans la couche cachée a comme fonction d'activation une fonction non linéarité sigmoïdal (le perceptron).
Ce type de neurones utilisés en réseaux donne le célèbre Perceptron Multicouches, ce dernier est probablement le réseau neuronal le plus utilisé pour des problèmes d'approximation de fonctions.

Toutefois son utilisant a des _n de classification reste possible même si, une fonction du type signe, ou d'autre type de RNA comme les réseaux de Hopefield ou les cartes de Kohonen semblent plus appropriés.

En effet, l'incapacité démontrée du Perceptron à résoudre des problèmes intéressants entraîna, en son temps, un ralentissement notable de la recherche sur le connexionnisme. Mais certains travaux démontrèrent, dans le courant des années 70, que l'on pouvait quand même dépasser ces limitations en intercalant, entre la couche d'entrée et la couche de sortie, une ou plusieurs couches intermédiaires (appelées aussi couches cachées), l'information circulant d'une couche à la suivante (réseau feed-forward)(Figure 3.1). Dans ce cas, on est certain de pouvoir approximer à la sortie n'importe quelle fonction de l'espace d'entrée ; mais si la littérature permet d'énoncer ceci, elle ne donne pas