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

d'indication en revanche sur le nombre et la taille de ces couches cachées, ce nombre devant être d'autant plus grand que cette fonction est plus irrégulière.

Ou φ est le vecteur d'entrée (appelé regresseur), et θ est la fonction d'activation des neurones utilisés dans le réseau, fonction sigmoidal pour le PMC.

Largement inspiré de la théorie des systèmes linéaires, (Ljung, 1987) et (Box and Jenkins, 1989), un nombre de regresseurs a été choisi pour être utilisés dans le cas non linéaire, et notamment avec le PMC (Chen et al., 1990 ; Sjoberg, 1995 ; Nørgaard, 2000). Ceci est pratique car comme établi pour le neurone simple, la non linéarité n'apparait que dans la liaison du regresseur à l'espace de sortie. Si la sortie estimée est $\hat{y}$ au temps k, elle est donnée par :

$$\widehat{y}(k|\theta) = g(\varphi(k), \theta) \tag{3.1}$$

A noter que des valeurs précédentes de by sont fonction de $g(\theta(k))$ et donc fonction de θ, c'est donc la raison pour la quelle $\hat{y}$ au temps k est donné par $\hat{y}$ $(k|\theta)$. Il est donc possible de distinguer les regresseurs suivants :

- Structure a reponse impulsionelle finni :

$$\varphi = [u(k-d)\ u(k-1-d)\cdots u(k-m-d)]^T \tag{3.2}$$

- Structure autoregressive avec entrées exogénes (autoregressive with exogenous inputs model structure (NARX)) :

$$\varphi = [y(k-1)\ y(k-2)\cdots y(k-n)\ u(k-1-d)\ u(k-2-d)\cdots u(k-m-d)]^T \tag{3.3}$$

- Structure d'erreur de sortie (output error model structure (NOE)) :

$$\varphi = [y(k-1|\theta)\ y(k-2|\theta)\cdots y(k-n|\theta)\ u(k-1-d)\ u(k-2-d)\cdots u(k-m-d)]^T \tag{3.4}$$

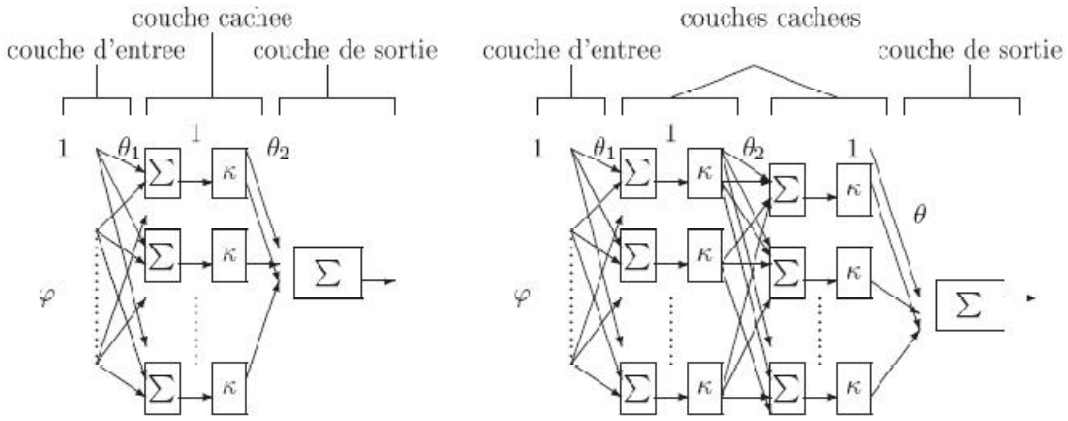
- Structure autoregressive a moyenne ajusté (auto-regressive moving average model (ARMAX)) :

$$\begin{aligned}\varphi = [&y(k-1)\ y(k-2)\cdots y(k-n)\ u(k-1-d)\\ &u(k-2-d)\cdots u(k-m-d)\ \epsilon(k-1)\cdots\ \epsilon(k-r)]^T\end{aligned} \tag{3.5}$$

  Ou $c(k) = y(k) - \widehat{y}(k|\theta)$.
- Structure Box-Jenkins (NBJ) :

$$\begin{aligned}\varphi = [&y(k-1|\theta)\ y(k-2|\theta)\ \cdots\ y(k-n|\theta)\ u(k-1-d)\ u(k-2-d)\cdots\\ &u(k-m-d)\ \epsilon(k-1)\cdots\ \epsilon(k-r)\ \epsilon_u(k-1)\cdots\ \epsilon_u(k-r)]^T\end{aligned} \tag{3.6}$$

  ou $\epsilon_u(k) = y(k) - \widehat{y}_u(k|\theta)$, $\widehat{y}_u(k|\theta)$ sont les sorties simulées n'utilisant que les valeurs passées de u.



(a) Une couche cachée (b) Deux couches cachées

Figure 3.1 – PMC Feedforward

3.2. PMC pour l'approximation de fonctions

Quand utilisés pour l'approximation de fonctions non linéaire (le but), le PMC a dans sa couche de sortie un nombre de neurones équivalent au nombre de sortie de la fonction. En général, le neurone de sortie est de type linéaire.

D'après le théorème de Cybenko (1989) qui stipule :

Soit *f*(.) une fonction continue, bornée et monotone. Soit *Ip* donnant l'hypercube $[0; 1]^p$ de dimension *p*. L'espace de la fonction continue sur *Ip* est donnée par *C*(*Ip*). Alors, si $f \in C(Ip)$ et $\varepsilon > 0$, il existe un entier *M* et un nombre de constantes réelles $\alpha i; \beta i$ et *wij* ou $i = 1; \ldots ; M$ tel que :

$$\widehat{F}(x_1, \cdots, x_p) = \sum_{i=1}^{M} \alpha_i f \left(\sum_{j=1}^{p} w_{ij} x_j - \beta_i \right) \tag{3.7}$$

Comme une approximation de la fonction *F*(.), ou :

$$|F(x_1, \cdots, x_p) - \widehat{F}(x_1, \cdots, x_p)| < \varepsilon \tag{3.8}$$

pour tout $x1; \ldots; xp$.

La figure 3.2 montre un PMC feedforward a une seule sortie *F*, conformément a l'équation (3.7) et construit suivant le théorème de Cybenko.

Remarques :

- Tan-sigmoid et log-sigmoid sont des fonctions continues, bornées et monotones.
- Equation (3.7) décrit un nombre d'entrées *p*, et une seule couches cachée de *M* neurones.

- Le neurone caché *i*, a les poids synaptiques *wi*1;*wi*2; … ;*wip* et les biais βi.

- La sortie du réseau est une combinaison linéaire des sorties correspondants aux neurones Cachés avec les poids α_1; α_2; α_M

- Le théorème de Cybenko est un théorème d'existence, de ce fait il ne précise pas le nombre de neurones de la couche cachée.

- En pratique, et comme le théorème reste vrai pour plusieurs couches cachées de neurones, il pourrait être plus profitable d'utiliser plus d'une couche cachée.

3.2.1. Une ou deux couches cachées ?

Dans le cas de PMC mono couches, les neurones inter-agit de façon global. Par exemple, lors de l'approximation d'une fonction complexe, une amélioration dans une partie de l'espace d'entré, peut entrainer une détérioration dans une autre partie (approximateur global). Par contre dans le cas d'un réseau a deux couches cachées on remarquera les conséquences suivantes :

Des caractéristiques locales sont extraite dans la première couche cachée. En d'autres termes, la première couche partitionne l'espace d'entrée en régions, il s'en suit un apprentissage de caractéristiques locales.

Les caractéristiques globales, sont alors, extraite dans la seconde couche cachée, ou les neurones de cette dernière, combine les sorties des neurones de la première couche, qui opèrent autour d'une région de l'espace d'entré construisant les caractéristiques globale de cette région mais produisant des zéros pour le reste de l'espace d'entrée.

De plus, certains problèmes (utilisant des neurones de types discontinue) ne peuvent être résolue avec une seule couche cachée, mais peuvent l'être par des réseaux a deux couches cachées.

3.3. Apprentissage par retro-propagation

L'apprentissage par rétro propagation de l'erreur, se passe en deux phases alternées. Dans la première, on présente un exemple à l'entrée, et on propage les signaux de couche en couche jusqu'à la sortie, à travers les poids connectant les sorties des neurones d'une couche à ceux de la couche suivante. On peut alors mettre en évidence les erreurs apparaissant sur les neurones de sortie. Dans la seconde phase, on applique l'algorithme de mise à jour expliqué précédemment, dans la dernière couche de poids, puis on propage les erreurs de sortie à travers cette couche de poids sur l'avant-dernière couche, ce qui permet alors de réitérer l'algorithme en mettant à jour l'avant-dernière couche de poids, et ainsi de suite.

Algorithme :

Mathématiquement :

Rappelons-nous de la règle d'apprentissage delta, Section 2.7.3

$$W_{k+1} = W_k + 2\mu e f'(z) X \tag{3.9}$$

$$\Delta w_{ji} = \mu \delta_j y_i \tag{3.31}$$

- Si j est dans une couche de sortie alors : $\delta_j = 2f'_j(z_j)e_j$
- Si j est dans une couche cachée, alors :$\delta_j = 2f'_j(z_j)\sum_{k=1}^{M} \delta_k w_{kj}$

3.4 Exemple du OU exclusif (XOR)

3.4.1 Solution analytique

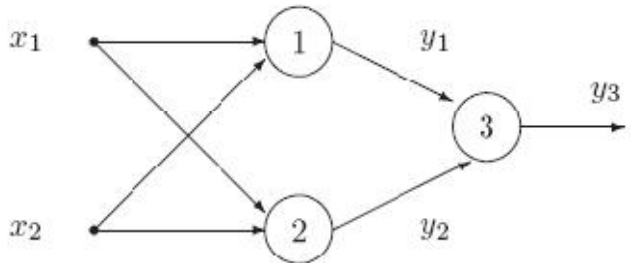


Figure 3.5 – PMC a 3 neurones

Décision au neurone 1 :

$$x_2 > -x_1 + 1.5 \ (y_1 = 1)$$

Décision au neurone 2 :

$$x_2 > x_1 + 0.5 \ (y_2 = 1)$$

Le neurone de sortie implémente la combinaison de y_1 et y_2 comme suit :

$$-2y_1 + y_2 > 0.5 \ \equiv [NOT(y_1)]AND y_2$$

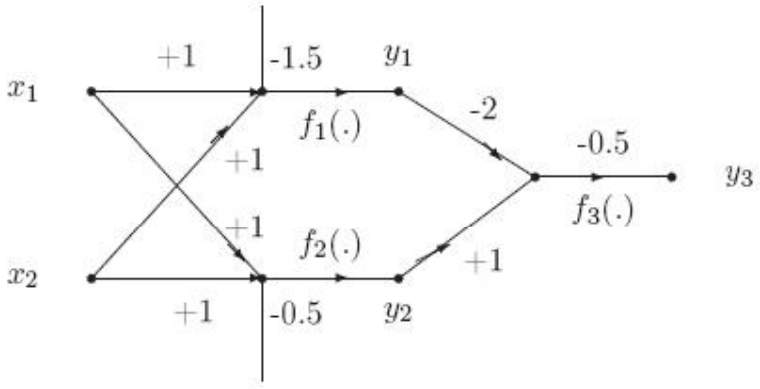


Figure 3.6 – Poids, biais et fonctions du PMC

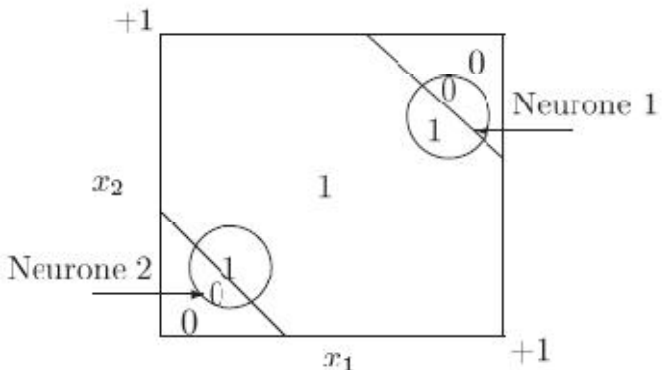


Figure 3.7 – Diagramme de décison du PMC

3.4.2 Solution adaptative

La fonction XOR est implémentée sous Matlab en utilisant les fonctions *newp* pour créer un RNA, et la fonction train pour l'apprentissage du réseau de neurones de type MLP. Les détails de l'implémentation sont donné par :

- 2 entrées + 2 neurones sigmoidal + 1 neurone linéaire.
- Apprentissage de 15 époques avec un pas d'apprentissage de 0.1.
- Initialisation aléatoire des poids et biais.

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

Performance après 5 époques :

$$T_{test} = [0.0000\ 1.0000\ 1.0006\ 0.0002]$$

On peut voir que le MLP a 1 couche cachée, contenant 2 neurones, permet de résoudre le problème du XOR, ou le perceptron seul ne peut y parvenir. L'apprentissage est très rapide et après 3 époques, Figure 3.8, l'erreur de référence est atteinte.

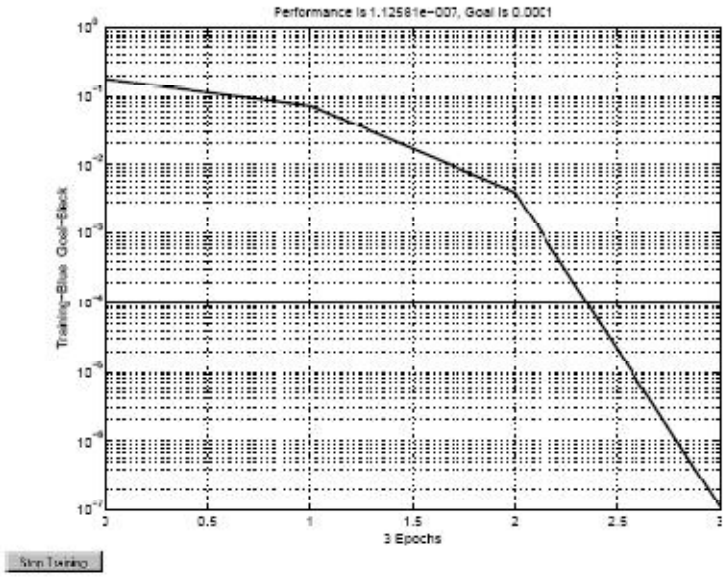


Figure 3.8 – Apprentissage du MLP pour l'implementation de la fonction XOR

3.5. Considération pratiques

Malgré une littérature théorique fournis, l'utilisation des ANN reste tributaire de l'application, sa nature ainsi que son environnement. Cette section donne quelques considération pratique a prendre en compte lors d'applications spécifiques.

3.6.1 Données informative pour les RNA

Le cœur ou la base de toute résolution dite boite noire (classification ou modélisation), est une base de connaissance riche quantitativement et qualitativement. Ceci reste vrai dans le cas de la modélisation par RNA, ou des données décrivant le fonctionnement du procédé sont essentielles pour les phases d'apprentissage et de validation du réseau neuronal.

Théoriquement pour qu'un RNA soit valide dans tout l'espace de fonctionnement, un apprentissage utilisant des données recouvrant tout l'espace d'entrée sortie doit être effectué. En pratique, ceci est irréalisable pour des raisons de faisabilité. En revanche, la validité des modèles est restreinte à une région opérative dans laquelle les données utilisées pour l'apprentissage et la validation ont été collectées.

Les données sont obtenues en appliquant un voir des protocoles d'essai sur le procédé réel en question. Ces protocoles consistent a changer les valeurs de(s) entrée(s), ex., la position d'une valve, un écoulement, un courant électrique, etc., pour récolter les valeur de la ou les sortie(s).

Concernant les modèles utilisés pour la commande et plus particulièrement en régulation, la région opérative se situe autour du point de fonctionnement. Il est souvent prouvé de se trouver confronter a

des obstacles ou problèmes qui empêche le bon déroulement d'une collecte de données informative quand on travaille sur des installations industrielles. Entre autre :
L'installation n'est pas disponible, ou est seulement disponible pendant un laps de temps déterminé pour ne pas interrompre la production.

- L'installation est mal et/ou pas correctement instrumentaliser, et ne permet pas la récolte d'information de données désirées. Il est souvent le cas que la direction s'oppose a tout rajout de capteur et/ou transmetteur qui pourrait aider le protocole d'essai, par peur de possible pannes, perte de temps ou violation d'un certificat de garantie par le fournisseur souvent très scrupuleux.

- Des régions opérative spécifique ne peuvent être investiguer pour des raisons de sécurités ou de productivités.

Ces considérations sont hors des prérogatives de l'ingénieur, et ne peuvent qu'être pris en comptes. L'ingénieur doit donc faire preuve de diplomatie pour la négociation d'un temps, décent, pour pouvoir assurer un protocole d'essai utilisable. De plus, il ne peut que s'accommoder de l'instrumentation existante puisque l'installation de nouveau senseurs/transmetteur n'est pas toujours possible.

Pour un problème de modélisation (approximation de fonctions) le protocole d'essai doit être assez long pour permettre la capture du temps de réponse du système donnée par le procédé.
Il est souvent le cas pour les systèmes lent, que la sortie apparaît comme stable, ou en fait elle est encore en phase convergente. C'est pour cela que l'ingénieur doit être patient lors de la conduction de protocoles d'essais. Une fois que le temps de réponse du système est capturé, alors les variations des entrées peuvent être plus courtes pour capturer le comportement haute fréquence du procédé.

Une méthode dite de validation croisée, cross validation, peut être utilisée dans le but de maximiser les informations extraites d'un protocole d'essai restreint.

3.6.2 Méthodes d'apprentissages

Apres la détermination d'une topologie appropriée le PMC doit subir une phase d'apprentissage dans le but de définir ces paramètres c.a.d., les poids et les biais de chaque neurone, ex : l'apprentissage par rétro propagation.
Apres la phase d'apprentissage, les capacités de prédiction du PMC doivent être testées sur un nouvel ensemble de donnée, ceci est appelé « split sample validation ».

Dans ce qui va suivre les problèmes engendrer par un sur apprentissage ainsi que l'utilisation d'un ensemble de donnée restreint lors de la modélisation sont adresse et des solutions sont proposées.

Sur apprentissage et arrêt prématuré

L'estimation des paramètres du PMC, est obtenue par la minimisation d'un critère quadratique. L'algorithme d'apprentissage peut, donc, être exécuter jusqu'a ce qu'il n y'ai plus d'amélioration perçue, c.a.d., jusqu'à ce qu'un minimum global (ou local) soit atteint. Toutefois il a été noté dans la littérature que si le modèle est évalué sur un ensemble dit de validation, en utilisant l'erreur de validation (l'erreur obtenue par cet ensemble), celle-ci va au début décroitre, puis va commencer à ce détérioré. Celle ci revient donc à croitre au fur et a mesure de l'apprentissage malgré que l'erreur de l'ensemble d'apprentissage va continuer a diminuer. Ce phénomène est connue sous le terme de sur apprentissage, « over training ».

En d'autre termes, au fur et a mesure de l'apprentissage, le PMC va avoir tendance à apprendre l'ensemble de donnée utilisé pour l'apprentissage, il va en résulter une mauvaise généralisation et une mauvaise prédiction pour un ensemble de données différent. Dans le but de palier à ce problème, une méthode d'apprentissage supervisé basée sur l'erreur de validation est implémentée. Cette méthode vérifie l'erreur de validation à chaque itération de l'apprentissage et la compare à sa valeur précédente pour déterminer le moment de commencement de la détérioration.

A ce moment, la valeur des poids et biais est sauvegardée. Pour être sure que ceci est un minimum global (ou du moins un minimum local décent) l'apprentissage n'est pas interrompu et devrait être effectuée pendant un nombre d'itérations suffisant pour être sure que l'erreur de validation ne décroitra pas.

Validation croisée

La validation croisée est une amélioration par rapport a la validation split sample, dans le sens ou tous les ensembles de données récoltés sont utilisés tour a tour pour la validation. Ceci s'avère plus efficace quand la taille de l'espace des données est restreint et/ou limité. La méthode est aussi appelée sélection de modèles croisés (Cross Model Sélection).

En pratique, la méthode divise l'ensemble des données D en n sous ensembles. Il en résulte n modèles a identifier en utilisant n sous ensembles de données. A chaque fois un nouveau sous ensemble différent est sélectionnée pour l'opération de validation, le reste des sous ensembles, concaténés, sont utilisés pour l'apprentissage. Ayant plusieurs estimations de validation recouvrant l'espace de données entier, confère à l'estimation un plus grand degré d'exactitude.

A la fin de la validation croisée on obtient n modèles. On peut alors choisir le meilleur ou le plus appropriée à notre région opérative, ou alors combiner tous les modèles obtenues. Une combinaison linéaire peut être utilisée à cet effet.