ECOLE NATIONALE POLYTECHNIQUE

Département de Génie Electrique

Projet de Fin d'Etudes

en vue de l'obtention du diplôme d'Ingénieur d'Etat
en Génie Electrique: option Automatique

Thème:

IDENTIFICATION ET COMMANDE DES SYSTEMES DYNAMIQUES PAR RESEAUX DE NEURONES

Proposé par:
F.BOUDJEMA
D.BOUKHETALA

Dirigé par:
F.BOUDJEMA
D.BOUKHETALA
M. C. SOUAMI

Etudié par:
HAMZI Boumediene
LABIOD Salim

Juillet 1995

ECOLE NATIONALE POLYTECHNIQUE

Département de Génie Electrique

Projet de Fin d'Etudes

en vue de l'obtention du diplôme d'Ingénieur d'Etat
en Génie Electrique: option Automatique

Thème:

IDENTIFICATION ET COMMANDE DES SYSTEMES DYNAMIQUES PAR RESEAUX DE NEURONES

Proposé par:
F.BOUDJEMA
D.BOUKHETALA

Dirigé par:
F.BOUDJEMA
D.BOUKHETALA
M. C. SOUAMI

Etudié par:
HAMZI Boumediene
LABIOD Salim

Juillet 1995

Je dédie ce travail à Mes Parents.

Boumediene

Je dédie ce travail

à Mes Parents

à mes frères

à mes soeurs

à ma famille

à mes amis.

Salim.

ملخص:

هـذا البحث يستعرض بعض طرق و هياكل تعليم ا لشبكا ت ا لعصبية ا لسا كنة و ا لد ينا ميكية ا لمستعملة في مجا ل التحكم الا لي.

في أ و ل الأ مر تطرقنا ا لى مشكل تمثيـل الأ نظمـة الـد ينـا ميكيـة غـير الخطيـة (تمثيـل مباشر، تمثيل عكسي)، قمنا بعد ذ لك بـإ ستعرا ض طـرق التحكـم ا لعصبي (نقـل مسـيطر موجود، تحكم تلاؤمي، تحكم عكسي).

في كل الحالات قدمنا نتائج محاكات عد دية.

كلمات مفتاحية: شبكا ت عصبية – تحكم غير خطي– تمثيل غير خطي – أ نظمة غير خطية.

Abstract:

This work exposes some structures, and learning methods for feedforward and recurrent neural networks as applied to Automatic Control. Nonlinear system identification (including forward plant modeling and inverse modeling) is discussed first, followed by consideration of neural controllers (including supervised, inverse, MRAC, disturbance rejection , STR, Computed Torque and criticism function based, controllers).Simulation results are reported in all cases.

Key words: Neural Networks, Nonlinear Control, Nonlinear Identification, Nonlinear Systems.

Résumé:

Ce travail expose quelques structures et méthodes d'apprentissage pour les réseaux de neurones statiques et dynamiques. En premier lieu, l'identification des systèmes non-linéaires (Identification directe, Identification Inverse) est exposée. Ensuite, une présentation de différentes techniques de commandes neuronales (Supervisée, Inverse, Commande adaptative et commande basée sur les fonctions critiques) est faite. Dans tous les cas des simulations sont effectuées.

Mots clés: Réseaux de neurones, Commande Non-linéaire, Identification Non-linéaire, Systèmes non-linéaires.

AVANT - PROPOS

Ce travail a été effectué au Laboratoire d'Automatique et au Laboratoire des Système Intelligents et Autonomes (LARSIA) de l'Ecole Nationale Polytechnique.

Nous exprimons notre profonde reconnaissance à Mrs. F. Boudjema, D. Boukhetala et M.C.Souami, qui en dirigeant ce travail nous ont fait profiter de leurs connaissances, de leurs conseils, de leurs aides et leur soutien et de l'intérêt bienveillant qu'ils nous ont témoigné. Qu'ils soient remerciés pour les nombreuses discussions, que nous avons eu , et par l'intérêt qu'ils montrent à notre travail.

Nous voudrons terminer en saluant la promotion de l'Automatique de Juin 1995.

Enfin , nous voulons exprimer nos remerciements à tous ceux qui ont contribué de prés ou de loin à l'aboutissement de ce travail.

SOMMAIRE

Chapitre II.

Identification Par Réseaux de Neurones

Chapitre III.

Commande Par Réseaux de Neurones

INTRODUCTION GENERALE

" Le génie est fait de un pour cent d'inspiration et de quatre vingt dix neuf pour cent de transpiration."

Thomas Alva EDISON (1847-1931)

INTRODUCTION GENERALE

La théorie des systèmes fournit des outils d'analyse et de synthèse parfaitement adaptés aux systèmes linéaires. Cependant, en pratique, les méthodes linéaires ne s'avèrent pas toujours applicables, parce qu'il n'est pas toujours possible de linéariser le système à commander. Même, lorsque la construction d'un modèle est réalisable, celui-ci ne permet que d'obtenir une estimation qualitative du système dans des limites définies, souvent très petites, en conservant des rapports qualitatifs avec une précision parfois insuffisante.

De nouvelles méthodes de calcul qui doivent prendre en compte les caractéristiques particulières des systèmes non-linéaires s'avèrent nécessaires.

Dans ce travail, nous sommes intéressés par l'identification et la commande des systèmes non-linéaires en utilisant des approximateurs de fonctions, basés sur l'interconnexion de plusieurs éléments de base qui sont non-linéaires, ces approximateurs de fonctions sont appelés "réseaux de neurones".

La première partie de ce travail présente un introduction à la théorie des réseaux de neurones, ou on donnera d'abord quelques définitions sur les réseaux de neurones (éléments de base, principe de fonctionnement...) , ensuite on subdivisera les réseaux de neurones en deux classes, les réseaux statiques et les réseaux dynamiques, et puis on présentera les méthodes de d'apprentissage des réseaux dans chaque classe; de même que de la capacité des réseaux de neurones a approximer n'importe quelle fonction, et enfin, on parlera des propriétés des réseaux de neurones qui les rendent souhaitables en identification et en commande des systèmes dynamiques.

Le second chapitre, présente les méthodes qui permettent l'identification des caractéristiques directes et inverses des systèmes dynamiques; L'identification directe est intéressée par l'émulation de la dynamique du système, elle peut être réalisée en utilisant les réseaux de neurones généralisés qui sont des combinaisons entre les réseaux de neurones et d'opérateurs linéaires, dans ce cadre quatre modèles de systèmes non-linéaires vont être proposés. Contrairement à l'identification directe, l'identification inverse est intéressée par l'émulation de la dynamique inverse du système non-linéaire; Ce chapitre sera terminé par des simulations.

La commande des systèmes non-linéaires par réseaux de neurones est présentée dans le troisième chapitre. On y présentera différents types de commande, commande supervisée, commande inverse, commande adaptative avec modèle de référence, rejet de perturbations, régulateur auto-ajustable, commande par bouclage linéarisant et enfin commande basée sur les fonctions critiques.

Dans la dernière partie, on présentera une conclusion générale et quelques perspectives.

Chapitre I

INTRODUCTION AUX RESEAUX DE NEURONES

" Seule la théorie décide de ce que l'on peut observer "

Albert EINSTEIN (1879-1955)

CHAPITRE I
INTRODUCTION AUX RESEAUX DE NEURONES

I. Introduction :

Tout a commencé il y a environs quatre millions d'années, lorsque le cerveau humain fit son apparition sur terre, depuis lors l'homme n'a cessé de se poser des questions comme celles concernant le sens de la vie, où celles concernant le sens de son existence sur terre, où encore celles concernant sa destinée; mais aussi celles concernant les phénomènes qui l'entourent, par exemple pourquoi une pierre une fois lancée vers le haut finit-elle par retomber ? ... si ces questions ont pu être posées c'est grâce à la merveilleuse machine à penser que constitue le cerveau; jusqu'au jour ou il se posa une question, qui est la base de notre travail; cette question concerne l'analyse et la compréhension du principe de fonctionnement de ce qui lui permettait d'analyser et de comprendre, c'était le début d'une longue et rude ruée vers le cerveau qui ne c'est encore pas achevée; va t'elle se terminer ? ça c'est la partie philosophique du problème, car les philosophes se sont penchés aussi sur ce problème, car analyser ce qui nous permet d'analyser n'est pas si facile qu'on puisse l'imaginer; et qu'on laissera le soin aux philosophes d'analyser, ce qui n'est pas notre problème à présent. En ce qui nous concerne nous sommes intéressés par le cerveau lui même; Comme l'a dit James Weston co-découvreur de l'A.D.N. (considérée comme l'une des plus complexes pièces biologiques sur terre) "The brain is the last and greatest biological frontier"; en effet, cette merveilleuse machine est composée de $36^{3.6^4}$ entités élémentaires appelées neurones, ce nombre est à peu prés égal au nombres d'étoiles dans la voie lactée ainsi qu'au nombre de galaxies dans l'univers observable; chaque neurone est relié à d'autres neurones par des synapses (dont le nombre est estimé à 36000 par neurone); Ainsi le cerveau contient à peu prés 10^{26} synapses. Le cerveau peut être considéré comme étant un système dynamique bouclé, non-linéaire, asynchrone, possédant une architecture parallèle, dont les dimensions sont des dimensions cosmologiques !. Un seul neurone n'est capable de rien faire ou presque, c'est l'interconnexion des neurones qui permet au cerveau de réaliser des prouesses (réflexion, vision, audition...); on appelle l'interconnexion de ces neurones "un réseau de neurones" - nous y voila ! - ; et c'est en essayant de comprendre le fonctionnement de ces réseaux de neurones que l'homme c'est trouvé en train de modéliser la merveilleuse machine que constitue le cerveau, c'est ce qu'a prétendu faire l'intelligence Artificielle (I.A) dans les années cinquante; Depuis lors l'I.A n'a cessé d'attirer des chercheurs, grâce à elle les systèmes experts sont apparus, les jeux électroniques, ainsi que le traitement de la parole mais elle n'est pas parvenue a résoudre les problèmes que l'homme résout avec brio (Traitement d'images, Reconnaissance des formes, Reconnaissance de la parole, Marche ...) car elle est basée sur le calcul symbolique et la logique d'ordre un , tandis que les problèmes courants ont plutôt un aspect numérique et probabiliste , ce qui n'est pas le cas de l'intelligence artificielle ; c'est cet aspect qui a permis aux *Réseaux de neurones artificiels* de voir le jour.

Dans le présent chapitre on va d'abord donner quelques définitions sur les réseaux de neurones (éléments de base , principe de fonctionnement ...) , et puis on va parler de deux types de réseaux [1] , les réseaux statiques et les réseaux dynamiques, on parlera ensuite de l'apprentissage de chaque réseau cité; de même que les propriétés des réseaux de neurones et enfin on parlera de la capacité des réseaux de neurones a approximer n'importe quelle fonction .

II. Les Réseaux De Neurones :

C'est la capacité du cerveau de faire des grandes choses à partir des éléments de base que constituent les neurones; qui a fasciné les chercheurs et qui a relancé la recherche dans cette direction; les chercheurs ont d'abord commencé par essayer de modéliser le neurone; le premier modèle fut réalisé par McCulloch et Pitts en 1943; dans ce modèle "Un Neurone formel fait la somme pondérée des potentiels d'actions qui lui parviennent (chacun de ces potentiels est une valeur numérique qui représente l'état du neurone qui l'a émit) , puis s'active suivant la valeur de cette sommation pondérée ; si cette somme dépasse un certain seuil , le neurone est activé et transmet une réponse (sous forme de potentiel d'action) dont la valeur est celle de son activation , si le neurone n'est pas activé, il ne transmet rien "; cette définition ne considère qu'une certaine classe de réseaux de neurones, En 1987 Hecht-Nielsen introduit le terme de **"Mapping Neural Network"(MNN),** Un MNN est défini comme suit :
"Un "mapping neural network" est un réseau de fonctions effectuant une relation, $\phi : I^n \rightarrow R^m$ *, en se basant sur l'interconnexion de neurones, considérés comme éléments de base effectuant une relation non-linéaire d'une façon parallèle et distribuée, ou* I^n *est un hypercube de n dimensions "*, cette définition rassemble une large classe de réseaux de neurones . D'après cette définition un réseau de neurone est caractérisé par le neurone comme élément de traitement de base, et de la façon dont ces derniers sont connectés . Ce sont ces deux aspects qu'on va maintenant développer .

1) Le neurone :

Un neurone est une fonctionnelle caractérisée par les éléments suivants (Fig. 1) :

(1) Un sommateur pondéré .
(2) Un système dynamique linéaire SISO .
(3) Une fonction non-linéaire statique (fonction d'activation) .

Chacun de ces éléments sera décrit dans les sections suivantes :

a) Le Sommateur Pondéré :

Le sommateur pondéré est décrit par l'équation suivante :

$$v_i(t) = \sum_{j=1}^{N} a_{ij}\, y_j(t) + \sum_{k=1}^{M} b_{ik}\, u_k(t) + w_i \ , \qquad \text{(I-1)}$$

[1] On utilisera sans ambiguïté les termes réseaux ou réseaux de neurones pour spécifier les réseaux de neurones artificiels .

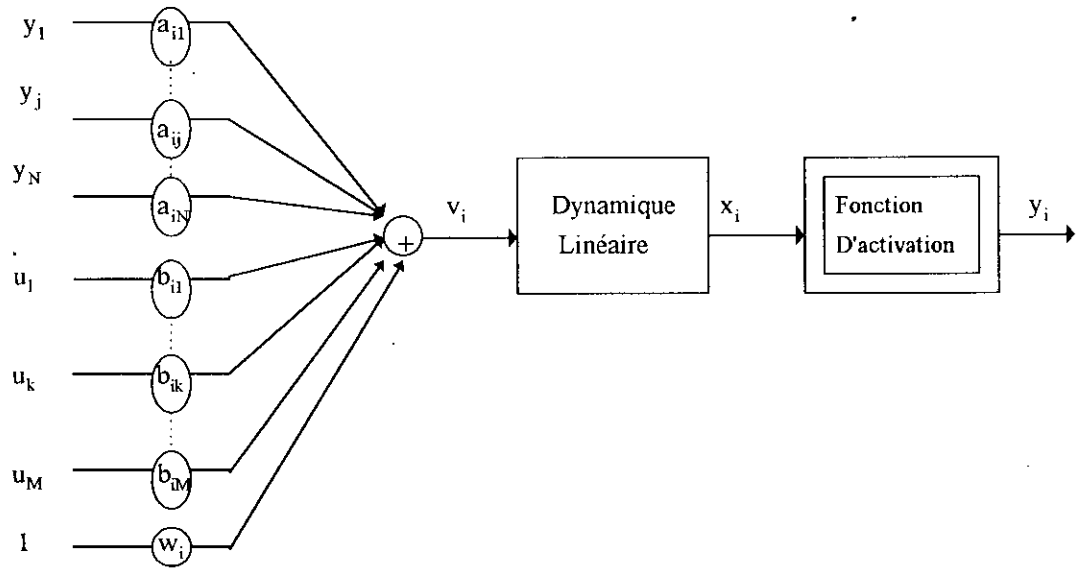


Figure 1 : Modèle d'un neurone

On remarque que v_i est fonction de la somme pondérée des sorties de tout les éléments y_j des entrées externes u_k et des poids correspondants a_{ij} et b_{ik} ; ainsi que des constantes w_i[2].

En comprimant l'équation (1) sous forme matricielle on trouvera :

$$v(t) = Ay(t) + Bu(t) + w \qquad \text{(I-2)}$$

où :

v(t) : vecteur colonne de dimension Nx1

y(t): vecteur colonne (Nx1) représentant les N sorties y_j

u(t): vecteur colonne (Nx1) représentant les M entrées u_k

w : vecteur colonne (Nx1) représentant les N constantes w_i .

A : Matrice N x N dont l'élément ij est a_{ij} .

B : Matrice N x M dont l'élément jk est b_{ik} .(les éléments w_i peuvent être incorporés avec les entrées u (il correspondent a u=1.))

b) Le Système Dynamique :

Le système linéaire mono-entrée mono-sortie, possède v_i comme entrée et x_i comme sortie . Sous la forme de fonction de transfert , on aura :

[2] Lorsque l'entrée du neurone est la somme pondérée de ses entrées extérieures, on dit que le neurone est un "neurone d'ordre un", un neurone d'ordre quelconque est un neurone dont la relation entre son entrée I et ses entrées externes x_i , i=1,n, ainsi que le matrice des poids v; est la suivante :

$$I = v_0 + \sum_{j_1=1}^{n_i} v_{j_1} x_{j_1} + \sum_{j_1=1}^{n_i} \sum_{j_2}^{n_i} v_{j_1 j_2} x_{j_1} x_{j_2} + \ldots$$

Ce type de neurones génère un autre type de réseaux plus généraux appelés "réseaux de neurones d'ordres supérieurs" ("High-Order neural networks"[Giles.C.L. and Maxwell.T., 1987].

$$\bar{x}_i(s) = H(s)\,\bar{v}_i(s) \tag{I-3}$$

ou la barre dénote la transformée de Laplace. Dans le domaine temporel l'équation (I-3) représente le produit de convolution donc :

$$x_i(t) = \int_{-\infty}^{t} h(t-t')\,v_i(t')\,dt' , \tag{I-4}$$

Avec H(s) représentant la fonction de transfert de h(t) .

Cinq choix de H(s) existent :

$$\begin{aligned} &H(s) = 1, \\ &H(s) = \frac{1}{s}, \\ &H(s) = \frac{1}{1+sT}, \\ &H(s) = \frac{1}{\alpha_0 + \alpha_1 s}, \\ &H(s) = e^{-sT}, \end{aligned} \tag{I-5}$$

Ceci correspond à :

$$\begin{aligned} &h(t) = \delta(t), \\ &h(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases}, \\ &h(t) = \frac{1}{T} e^{-\frac{t}{T}}, \\ &h(t) = \frac{1}{\alpha_0} e^{-\left(\alpha_1/\alpha_0\right)t}, \\ &h(t) = \delta(t-T), \end{aligned} \tag{I-6}$$

ou δ est l'impulsion de Dirac .Dans le domaine temporel les relations entrées sorties sont :

$$\begin{aligned} x_i(t) &= v_i(t), \\ \dot{x}_i(t) &= v_i(t), \\ T\dot{x}_i(t) + x_i(t) &= v_i(t), \\ \alpha_0\dot{x}_i(t) + \alpha_1 x_i(t) &= v_i(t), \\ x_i(t) &= v_i(t-T). \end{aligned} \tag{I-7}$$

On remarque que les trois premières équations ne sont que des cas particuliers de la quatrième.

Une version discrète des équations dynamiques est possible, par exemple la quatrième équation devient :

$$\alpha_0 x_i(t+1) + \alpha_1 x_i(t) = v_i(t) \qquad \text{(I-8)}$$

dans ce cas t indique l'instant d'échantillonnage .

c) Fonction d'activation :

La fonction d'activation g(.) transforme le signal x_i non borné a l'instant *t* en un signal borné y_i [Kosko ,1992a] :

$$y_i = g(x_i) \qquad \text{(I-9)}$$

cette fonction est en général monotone non-décroissante; ainsi l'augmentation de l'entrée ne peut que faire augmenter la sortie ou la garder constante; elles ne peuvent jamais faire décroître le signal [3] .

La monotonicité de la fonction d'activation implique que les neurones sont non-linéaires mais pas trop , c'est la semi-linéarité ; le choix d'une fonction d'activation linéaire facilite les calculs , mais ce type de neurones n'est pas robuste; tandis que les neurones dont la fonction d'activation est non linéaire augmente la capacité du réseau a approximer des fonctions complexes, facilitent la suppression des bruits mais les calculs et l'analyse risquent d'être complexes, en plus ceci favorise l'instabilité du réseau de neurones considéré comme étant un système dynamique [Kosko , 1992a].

La fonction d'activation g(.) peut avoir l'un des choix doubles :

(1) Dérivable / non-dérivable .
(2) Ressemble a un pic / ressemble a un échelon .
(3) Positive / moyenne nulle .

La première catégorie permet de distinguer entre les fonctions lisses et les fonctions dures. Les fonctions lisses sont fondamentales dans les algorithmes d'apprentissage tels que la "backpropagation" -voir la section III-2-, tandis que les fonctions d'activations non-dérivables sont utilisés lorsqu'on désire avoir des réponses possédant deux valeurs (la fonction signum par exemple) .

La seconde catégorie permet de distinguer entre les fonctions d'activations qui possèdent des valeurs importantes autour de zéro et celles qui possèdent des valeurs importantes loin de l'origine .

La troisième catégorie distingue entre les fonctions qui varient entre 0 et 1 et celles qui varient entre -1 et 1 .

[3] En fait il existe une fonction d'activation qui viole ces conditions qui est la Gaussienne , c'est la fonction d'activation des neurones situés dans le champ de vision , elle est la base de réseaux de neurones assez spéciaux appelés Gaussian Potential Function Networks (GPFN) [S.Lee and R.M.Kil , 1988] ; apparemment ce type de réseaux possède la capacité d'approximer des fonctions multidimensionnelles plus que les fonctions monotones ceci est basé sur la théorie des "Radial Basis Functions" (RBF) [T.Poggio and F.Girosi , 1990].

On va maintenant citer quelques fonctions d'activations usuelles; dont la **sigmoïde** qui est la plus utilisée [Fig. 2.] ; une fonction sigmoïde $\phi(.)$ est une fonction bornée , monotone croissante ayant la propriété suivante [Funahashi , 1989]:

"Si nous définissons $\phi_\varepsilon(x)= \phi(x/\varepsilon)$ $(\varepsilon>0)$; Alors les dérivées $\phi'_\varepsilon(x) = (1/\varepsilon)\phi'(x/\varepsilon)$ convergent dans le sens de la fonction généralisée [Kolmogorov A. and S. Fomine, 1973], vers la fonction δ quand $\varepsilon \to 0$. Ainsi, si $\phi(\infty)-\phi(-\infty)= 1$, alors, pour toute fonction $g(x)$ infiniment dérivable , on aura : $\lim_{\varepsilon\to 0^+} \int_{-\infty}^{\infty} \phi'_\varepsilon(x)\cdot g(x)\,dx = g(0)$."

On peut citer comme exemples les fonctions suivantes :

Exemple 1 : Pour $\phi(x)= 1/(1+\exp(-x))$, $\phi'_\varepsilon(x) = 1/\varepsilon \exp(-x/\varepsilon)/(1+\exp(-x/\varepsilon))^2$; ainsi $\phi(x)$ est une fonction sigmoïde .

Exemple 2 : Pour $\phi(x) = 1/\sqrt{2\pi}\int_{-\infty}^{x} \exp(-t^2/\varepsilon)\,dt$, $\phi'_\varepsilon(x) = \varepsilon/\sqrt{2\pi}\exp(-x^2/2\varepsilon)$, donc $\phi(x)$ est une fonction sigmoïde .

Exemple 3 : Pour $\phi(x)$ définie par $\phi(x)=0\ (x<0)$, $\phi(x)=x\ (0<x<1)$, et $\phi(x)=1\ (x\geq 1)$, $\phi'_\varepsilon(x)=0\ (x<0 \text{ et } x\geq\varepsilon)$, $\phi'_\varepsilon(x)= 1/\varepsilon\ (0\leq x<\varepsilon)$, et $\phi(x)$ est une fonction sigmoïde .

Comme on l'a cité précédemment dans le modèle de McCulloch et Pitts, une fonction seuil $\phi(x)=1\ (x\geq 0)$, $=0\ (x<0)$ est utilisée comme fonction d'activation du neurone de sortie .

Les fonctions sigmoïde $\phi(x)$ possédant la propriété $\phi(-\infty)=0$ et $\phi(\infty)=1$ sont appropriées comme fonctions d'activations du neurone de sortie , car $\phi_\varepsilon(x)$ tend vers la fonction seuil utilisée dans le modèle de Mc-Culloch et Pitts quand $\varepsilon \to 0^+$.

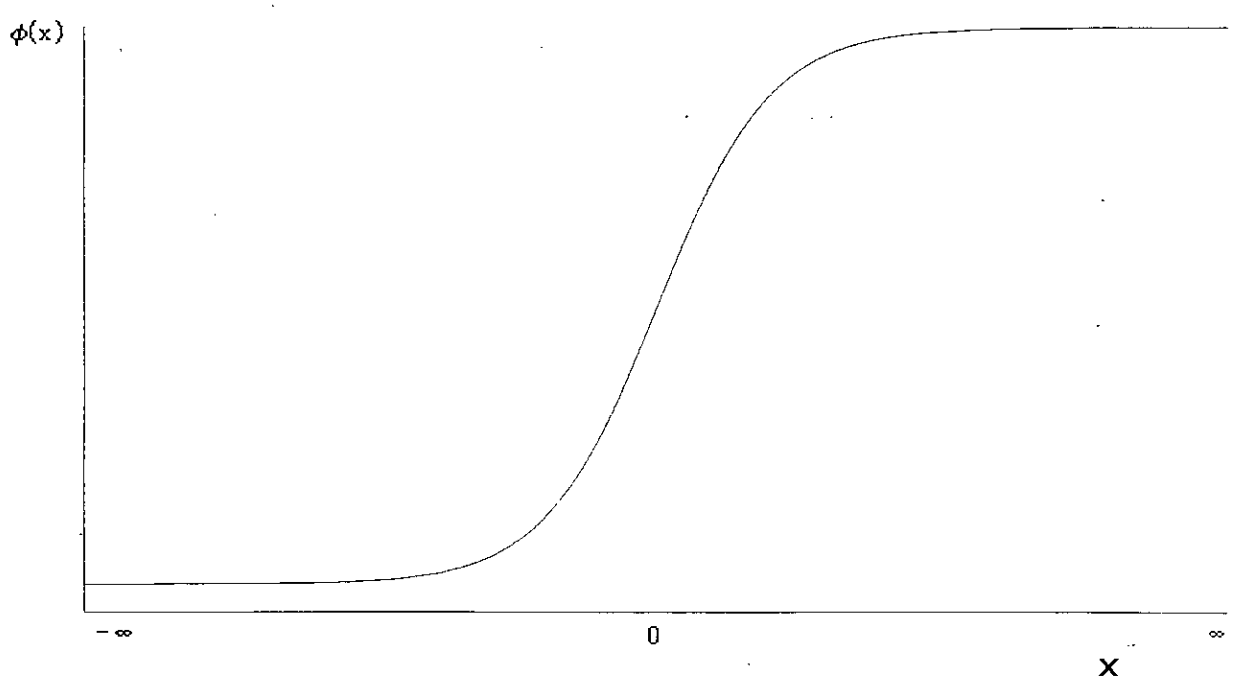


Figure 2 : Fonction Sigmoïde .

Mac-Culloch et Pitts ont démontré qu'on peut concevoir n'importe quelle fonction logique en utilisant le modèle cité ci-dessus[4] ; Intuitivement on peut dire qu'on peut approximer n'importe quelle fonction continue en utilisant des fonctions d'activation sigmoïdes, ceci a été démontré dans [Funahashi, 1989], [Cybenko, 1989] .

On peut citer d'une manière non-exhaustive les fonctions d'activations suivantes :

La première est définie par :

$$g(x^{k+1}) = \begin{cases} 1 & \text{si } x^{k+1} > T \\ g(x^k) & \text{si } x^{k+1} = T \\ 0 & \text{si } x^{k+1} < T \end{cases} \qquad \text{(I-10)}$$

pour n'importe quelle $T \in \Re$. L'indice k indique l'instant d'échantillonnage; si l'entrée a l'instant k est égale au seuil T le neurone garde son ancienne sortie, sinon elle est soit 0 ou 1 .

La seconde ressemble a la fonction sigmoïde c'est la **tangente hyperbolique** :

$$g(x) = \tanh(cx) . \qquad \text{(I-11a)}$$

Comme la sigmoïde,la tangente hyperbolique est égale a un rapport entre la somme de deux exponentielles :

$$\tanh(cx) = \frac{e^{cx} - e^{-cx}}{e^{cx} + e^{-cx}} \qquad \text{(I-11b)}$$

Cette propriété permet une expression simple de la dérivée , ainsi que la positivité de cette dernière .

Une autre fonction d'activation est la fonction **linéaire seuillée** définie par :

$$g(x) = \begin{cases} 1 & \text{si } cx \geq 1 \\ 0 & \text{si } cx < 0 \\ cx & \text{ailleurs} \end{cases} \qquad \text{(I-12)}$$

qui peut être mise sous la forme $g(x) = \min(1, \max(0, cx))$; entre ses valeurs maximales et minimales la fonction est monotone croissante , car $g' = c > 0$.

La fonction linéaire seuillée n'est linéaire que si l'entrée est bornée; la fonction d'activation $g(x) = cx$ n'est pas bornée, elle n'est pas souhaitable comme fonction d'activation . On peut remarquer que les fonctions sigmoïde et tangente hyperbolique se comportent comme des fonctions linéaires quand l'entrée est relativement petite, ainsi elles peuvent approximer la fonction linéaire seuillée.

---

[4] Ceci peut être facilement compris en se rappelant que toute fonction logique peut être écrite sous forme de somme de produits, ainsi cette fonction peut être réalisée par un réseau qui possède deux couches, la première couche implante les portes "AND", tandis que la seconde implante les portes "OR".

L'**exponentielle seuillée** est une autre fonction qu'on peut utiliser elle est définie par :

$$g(x) = \min(1, e^{cx}) \tag{I-13}$$

Quand $e^{cx} < 1$, $g' = c e^{cx} > 0$. La seconde dérivée est elle aussi positive $g'' = c^2 e^{cx} > 0$, ainsi que les dérivées supérieures $g^{(n)} = c^n e^{cx} > 0$; ainsi elle amplifie toute les variations ; ceci accélère la convergence.

La **distribution exponentielle** est un autre exemple des fonctions exponentielles seuillées :

$$g(x) = \max(0, 1 - e^{-cx}) \tag{I-14}$$

La fonction est monotone croissante car $g' = c e^{-cx} > 0$. En plus , on peut remarquer que cette fonction d'activation est strictement convexe car $g'' = -c^2 e^{-cx} < 0$; ceci produit un effet de diminution si le signal s'approche de la saturation .

La dernière fonction d'activation appartient a la **famille des fonctions rationnelles :**

$$g(x) = \max\left(0, \frac{x^n}{c + x^n}\right) \quad , \; n > 1 \; . \tag{I-15}$$

Pour des entrées positives , la fonction est monotone croissante car :

$$g' = \frac{c n x^{n-1}}{(c + x^n)^2} > 0.$$

2) Les connexions :

Dans le domaine des réseaux de neurones on parle surtout de **champs de neurones** pour spécifier un regroupement topologique de plusieurs neurones [Kosko, 1992]. En général un réseau de neurones contient plusieurs champs de neurones . Le regroupement topologique des neurones dans un champ de neurones est souvent défini par la proximité des neurones par rapport à d'autres neurones qui seront considérés comme origine du champ .

Le cas le plus simple, qui est le cas qu'on va considérer , les neurones ne seront pas topologiquement ordonnés . Ils ne seront reliés que par des connexions. Kohonen [Kosko , 1992a] appelle ce manque de structure topologique dans un champ de neurones, **topologie d'ordre zéro** .

On dénotera par F_X le champ de neurones par défaut . Un second champ de neurones sera noté par F_y , un troisième par F_z ....

L'interconnexion de plusieurs neurones sera décrite par un système d'équations différentielles du premier ordre, ou par un système d'équations aux différences qui régissent l'évolution au cours du temps des sorties de chaque neurone en fonction de ses entrées, si nous considérons les champs F_X et F_y , les équations qui régissent le réseau de neurones seront décrites par :

$$\dot{x}_1 = g_1(F_X, F_Y, \ldots.) \tag{I-16}$$

$$\dot{x}_n = g_n(F_X, F_Y, \ldots.) \tag{I-17}$$

$$\dot{y}_1 = h_1(F_X, F_Y, \ldots.) \tag{I-18}$$

$$\dot{y}_p = h_p(F_X, F_Y, \ldots.) \tag{I-19}$$

Sous forme vectorielle les équations précédentes deviennent :

$$\dot{\mathbf{x}} = \mathbf{g}(F_X, F_Y, \ldots) \tag{I-20}$$

$$\dot{\mathbf{y}} = \mathbf{h}(F_X, F_Y, \ldots) \tag{I-21}$$

les variables x_i et y_j sont respectivement les entrées du i -ème élément du champ F_X et du j-ème neurone du champ F_y. Les arguments des fonctions g_i et h_j incluent toutes les informations concernant les connexions et les entrées.

On peut noter l'absence de la variable temporelle des équations précédentes; ainsi le modèle du réseau est un système dynamique **autonome**.

A titre d'exemple considérons le cas d'un réseau statique ($H(s)=1$) ainsi les équations précédentes deviennent un système d'équations algébriques :

$$\begin{aligned} x(t) &= Ay(t) + Bu(t) + w \\ y(t) &= g(x(t)) \end{aligned} \tag{I-22}$$

ou x est un vecteur de N éléments x_i(voir Figure 1) et $g(x)$ est un vecteur dont les éléments sont $g(x_i)$. Si par exemple , le neurone se comportait comme un filtre passe bas alors $H(s) = \frac{1}{1+Ts}$ alors dans ce cas le champ de neurones sera régit par les équations différentielles suivantes :

$$\begin{aligned} T\dot{x}(t) + x(t) &= Ay(t) + Bu(t) + w \\ y(t) &= g(x(t)) \end{aligned} \tag{I-23}$$

On peut facilement remarquer que les solutions de (I-22) constituent le régime permanent de (I-23) .

Une version discrète de (I-23) est :

$$\begin{aligned} Tx(t+1) + (1-T)x(t) &= Ay(t) + Bu(t) + w \\ y(t) &= g(x(t)) \end{aligned} \tag{I-24}$$

Le comportement de ces champs de neurones dépend clairement de la matrice d'interconnexion *A* et de la forme de *H(s)* . Quelques formes particulières de la matrice *A* seront discutées dans les parties suivantes.

3) Les Réseaux De Neurones Statiques :

Comme indiqué dans la partie précédente, l'interconnexion de plusieurs neurones définit un champ de neurones, ce champ de neurones peut avoir plusieurs architectures; ceci dépend du choix de la dynamique du neurone utilisé comme élément de base.

Un réseau de neurones statique est un réseau tel que H(s)=1 dans l'équation (I-3); c'est un réseau dont la topologie de la connexion ne contient pas de boucles synaptiques fermées. La matrice A est telle que le réseau de neurones puisse être décomposé en plusieurs champs de neurones élémentaires appelés "couches"; les couches différentes des couches d'entrée ou de sortie sont appelées "couches cachées", un neurone dans une couche ne reçoit ses entrées que des neurones situés plus en amont, dans le sens entrée-sortie, mais pas forcément seulement des neurones situés sur la couche immédiatement précédente (Fig. 3).

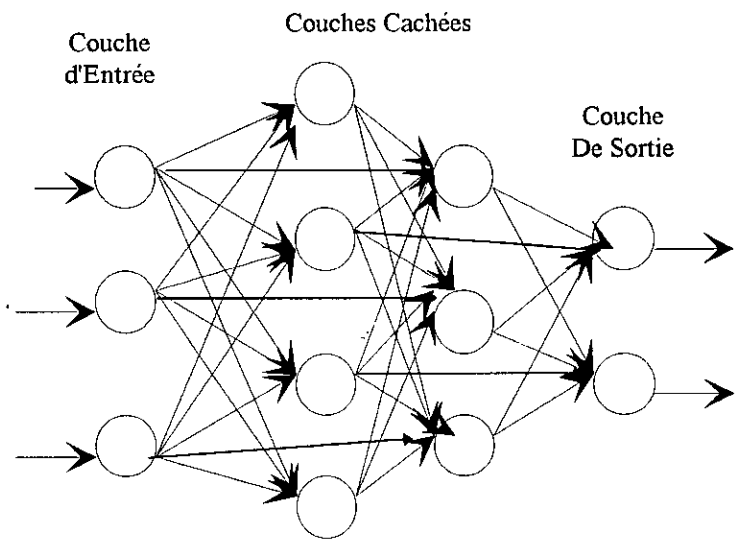


Figure 3 : Architecture d'un Réseau Statique

Par exemple, dans un réseau de neurones qui possède 4 couches, chaque couche contenant *N* neurones, on peut écrire les vecteurs *x, y, u* et *w* de l'équation (I-22) comme suit:

$$\begin{bmatrix} x^1(t) \\ x^2(t) \\ x^3(t) \\ x^4(t) \end{bmatrix} = A \begin{bmatrix} y^1(t) \\ y^2(t) \\ y^3(t) \\ y^4(t) \end{bmatrix} + B \begin{bmatrix} u^1(t) \\ u^2(t) \\ u^3(t) \\ u^4(t) \end{bmatrix} + \begin{bmatrix} w^1 \\ w^2 \\ w^3 \\ w^4 \end{bmatrix}, \qquad \text{(I-25)}$$

Les indices supérieurs indiquent le numéro de la couche. La structure des matrices A et B est la suivante:

$$A=\begin{bmatrix} 0_{NN} & 0_{NN} & 0_{NN} & 0_{NN} \\ A^2 & 0_{NN} & 0_{NN} & 0_{NN} \\ 0_{NN} & A^3 & 0_{NN} & 0_{NN} \\ 0_{NN} & 0_{NN} & A^4 & 0_{NN} \end{bmatrix};\quad B=\begin{bmatrix} B^1 & 0_{NM} & 0_{NM} & 0_{NM} \\ 0_{NM} & 0_{NM} & 0_{NM} & 0_{NM} \\ 0_{NM} & 0_{NM} & 0_{NM} & 0_{NM} \\ 0_{NM} & 0_{NM} & 0_{NM} & 0_{NM} \end{bmatrix} \tag{I-26}$$

Avec: 0_{NN} est la matrice nulle de dimensions $N\times N$, 0_{NM} est la matrice nulle de dimensions $N\times M$. A^2, A^3, A^4 sont les matrices des poids dont les dimensions sont $N\times N$, tandis que B^1 est une matrice des poids de dimensions $N\times M$.
Pour la première couche on a:

$$\begin{aligned} x^1(t) &= B^1u^1(t)+w^1, \\ y^1(t) &= g(x^1(t)), \end{aligned} \tag{I-27}$$

Pour les couches 2, 3 et 4 on aura :

$$\begin{aligned} x^{\ell}(t) &= A^{\ell}y^{\ell-1}(t)+w^{\ell}, \\ y^{\ell}(t) &= g(x^{\ell}(t)), \end{aligned} \tag{I-28}$$

avec ℓ=2,3,4.

g(.) est la fonction définie dans la section (1-c).

4) Les Réseaux De Neurones Dynamiques:

Les réseaux de neurones dynamiques ou récurrents sont différents des réseaux de neurones statiques car leur architecture contient un bouclage (la topologie des connexions est bouclée). En général, la sortie de chaque neurone est réinjectée en entrée de chaque neurones, grâce a des poids variables. Un réseau de neurones composé de cès éléments, dont tout les poids sont en général non-nuls, est dit *entièrement connecté*. Le réseau étant fondamentalement dynamique, il ne contient qu'une seule couche (Fig.4).

L'introduction de ce bouclage produit un réseau de neurones dynamique avec différents points d'équilibre; Ils est régit par l'équation suivante:

$$\begin{aligned} \dot{x}(t) &= F(x(t),u(t),\theta), \\ y(t) &= G(x(t),\theta). \end{aligned} \tag{I-29}$$

Dans ce cas, x représente l'état, u les entrées externes, θ est un vecteur de paramètres du réseau. F est une fonction qui représente la structure du réseau, G est une fonction qui représente la relation entre les états et les sorties; F doit satisfaire les conditions de Lipschitz sur x.

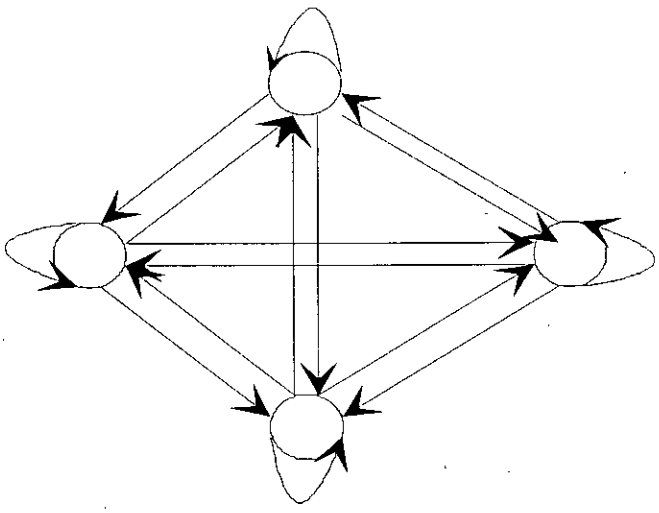


Fig 4: Architecture d'un réseau récurrent

Contrairement a un réseau de neurones statique qui donne la même sortie chaque fois qu'on lui présente la même entrée, un réseau de neurones dynamique, peut donner une sortie différente en lui présentant la même entrée a des instants différents, ceci dépend des entrées qu'on lui a présentées précédemment. Ainsi, l'ordre de la présentation des entrées-sorties est tout aussi important que les exemples eux mêmes.

4-1) Les Réseaux De Hopfield:

L'un des réseaux dynamiques les plus utilisés est appelé *réseau de Hopfield* [Kosko, 1992]; ces réseaux constituent un cas particulier de l'équation (I-29), ils sont basés des neurones dynamiques qui sont régit par l'équation différentielle suivante :

$$\begin{aligned} T_i \dot{x}_i &= -x_i + \sum_{j=1}^{N} a_{ij} y_j + u_i, \\ y_i &= g_i(x_i), \quad i = 1, \ldots, N, \end{aligned} \tag{I-30}$$

avec $y_i(t_0) = y_i^0$; y_j sont des sorties de tous les neurones dont le nombre est N.

On peut distinguer entre deux types de réseaux dynamiques [Hunt *et al*, 1992]. Le premier suppose une séparation entre les états et le sorties:

$$\begin{aligned} T &= \mathrm{diag}[T_1, \ldots, T_N], \\ T\dot{x} &= -x + Ay + u, \\ y &= g(x) \end{aligned} \tag{I-31}$$

tandis que le second assume que la sortie est une combinaison linéaire des états, i.e. en notation matricielle:

$$\begin{aligned} s &= Ay, \\ T\dot{x} &= -x + g(s) + u, \\ y &= x \end{aligned} \qquad \text{(I-32-a)}$$

Une version discrète de l'équation (I-32-a) est :

$$\begin{aligned} s(t) &= Ay(t), \\ Tx(t+1) &= -x(t) + g(s(t)) + u(t), \\ y(t) &= x(t). \end{aligned} \qquad \text{(I-32-b)}$$

g(.) est une fonction qui satisfait les conditions suivantes :

$$\begin{aligned} &\text{(i)}\ x_i g(x_i) > 0 \quad \forall x_i \in \Re \\ &\text{(ii)}\ \lim_{|x_i|\to\infty} g_i(x_i) = \mathrm{sgn}(x_i) \\ &\text{(iii)}\ g_i(x_i)/x_i \geq g_i(y_i)/y_i \quad \forall |x_i| \leq |y_i| \\ &\text{(iv)}\ g_i'(x_i) = dg_i(x_i)/dx_i > 0 \quad \forall x_i \in \Re \end{aligned} \qquad \text{(I-33)}$$

A partir de ces conditions on peut facilement remarquer que les fonctions $g_i(x_i) = \tan^{-1}(\pi\lambda x_i / 2)2/\pi$ et $g_i(x_i) = \tanh(\lambda x_i)$ satisfont les conditions citées [Michel. *et al,* 1989],[Sudharsanan *et al,* 1991].

Le réseau de Hopfield appartient a une très importante classe de réseaux de neurones qui sont *globalement stables*. Ils convergent exponentiellement vers les points d'équilibre, pour n'importe quelles entrées, comme on le démontrera dans la section suivante.

4-2) Stabilité des Réseaux De Neurones Dynamiques :

Le réseau de Hopfield appartient à une large classe de réseaux de neurones dynamiques proposée et analysée par Cohen et Grossberg [Cohen and Grossberg, 1983], cette classe est définie par le système d'équations différentielles suivant :

$$\frac{dx_i}{dt} = a_i(x_i)\left[b_i(x_i) - \sum_{j=1}^{n} c_{ij} d_j(x_j)\right] \qquad \text{(I-34-a)}$$

avec: **C**=[c_{ij}] est une matrice symétrique, i.e. $c_{ij} = c_{ji}$, $a_i(.)$sont des fonctions définies positives, i.e. $a_i(x_i) \geq 0$, $d_j(.)$ sont des fonctions monotones croissantes.

La stabilité d'un réseau de neurones dynamique est très importante (sinon on ne pourrait rien stocker...). Pour pouvoir étudier la stabilité de ce réseau, on appliquera la méthode de Lyapounov, i.e. un système dynamique est stable s'il existe une fonction de Lyapounov pour ce système. Cohen et Grossberg ont démontré qu'une fonction de

Lyapounov existait pour le système dynamique décrit par l'équation (I-34-a), elle est égale à :

$$L = -\sum_{i=1}^{n}\int_{0}^{x_i} b_i(\xi_i)\, d'(\xi_i)\, d\xi_i + \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{jk}\, d_j(x_j)\, d_k(x_k) \qquad \text{(I-34-b)}$$

En effet :

$$\frac{dL}{dt} = \sum_{i=1}^{n}\frac{\partial L}{\partial x_i}\frac{dx_i}{dt} \qquad \text{(I-34-c)}$$

Puisque la matrice **C** est symétrique :

$$\frac{\partial L}{\partial x_i} = -d_i'(x_i)\left[b_i(x_i) - \sum_{j=1}^{n} c_{ij}\, d_j(x_j)\right] \qquad \text{(I-34-d)}$$

En injectant (I-34-a) et (I-34-d) dans (I-34-c), on trouve :

$$\frac{dL}{dt} = -\sum_{i=1}^{n} a_i(x_i)\, d_i'(x_i)\left[b_i(x_i) - \sum_{j=1}^{n} c_{ij}\, d_j(x_j)\right]^2 \qquad \text{(I-34-e)}$$

Puisque, $a_i(x_i) \geq 0$ et $d_i'(x_i) \geq 0$; donc $dL/dt \leq 0$. Quand $dL/dt = 0$, le système atteint le point d'équilibre. Le problème avec ce théorème est le fait que les poids doivent être positifs, et que la matrice des poids doit être symétrique, ces deux conditions sont le prix a payer pour permettre la généralité; Généralement, ces deux conditions sont facilement violées; En plus ce théorème est une *condition suffisante*, donc la non-vérification de ces conditions n'implique pas l'instabilité.

En 1991, dans [Sudharsanan and Sundareshan, 1991], les auteurs ont pu relaxer ces deux conditions, dans le cas d'un réseau de Hopfield. Dans cet article, les auteurs ont démontré quelques théorèmes, ou des conditions sur la stabilité exponentielle[5] et l'instabilité des points d'équilibre ont été établies, ces conditions assurent l'unicité du point d'équilibre dans une zone spécifiée. Le degré de l'exponentielle stabilité des points d'équilibre a été estimé, ainsi que les estimées des régions d'attraction des points d'équilibre stables.

Le lecteur intéressé par l'étude de la stabilité des réseaux de neurones dynamiques est renvoyé vers les articles [Li, Michel and Porod, 1988], [Michel, Farrell and Porod, 1989] et [Sudharsanan and Sundareshan, 1991].

---

[5] On dit qu'un point d'équilibre isolé $x^*=0$, du système $\dot{x} = f(x(t))$, est *exponentiellement stable* dans $B = \{x: \|x\| \leq r\}$, $(r>0)$; avec un degré η, si chaque trajectoire commenceant à n'importe quelle condition initiale réalisable $x(t_0) = x_0 \in B$, satisfait la condition : $\|x(t)\| \leq \pi\|x_0\| \exp(-\eta(t-t_0))$, $\forall t \geq t_0, \forall x \in B$; avec, π et η étant des constantes positives indépendantes des conditions initiales (t_0, x_0).

Une seconde définition stipule qu'un point d'équilibre $x^*=0$ du système précédent est exponentiellement stable avec un degré η s'il existe une fonction de Lyapounov $L: \Re^n \rightarrow \Re$, qui satisfait les conditions suivantes : a) L(x) possède des dérivées partielles continues par rapport à chaque élément $x \in \Re^n$; b) L(x) est définie positive dans B; c) La dérivée par rapport au temps, le long des trajectoires du système précédent satisfait: $dL(x)/dt \leq -2\eta L(x)$; $\forall x \in B, \forall t \geq t_0$.

III. Apprentissage Des Réseaux De Neurones:

On appelle apprentissage l'opération par laquelle le réseau de neurones acquiert la capacité de faire certaines taches en modifiant ses paramètres internes (connexions) en utilisant un algorithme d'adaptation paramétrique appelé *algorithme d'apprentissage*, ainsi apprentissage rime avec *changement, c'est n'importe quel changement sur n'importe quelle connexion* . On dit que le réseau a appris la paire entrée-sortie (x_i , y_i) s'il répond avec y_i si x_i est présentée comme entrée; la paire (x_i , y_i) représente un échantillon d'une fonction $f:R^n \rightarrow R^p$, la fonction f associe *p* vecteurs **y** a *n* vecteurs **x** . Le réseau a appris la fonction f s'il répond par **y** , avec **y** = f(**x**), si **x** qui varie dans le domaine de définition de f. On dit que le réseau a partiellement appris s'il répond par y′, qui est proche de y = f(x), lorsqu'on lui présente x′ qui est proche de x .

On peut subdiviser le processus d'apprentissage en trois classes :

- Apprentissage Supervisé ("Supervised Learning")
- Apprentissage Non-Supervisé ("Unsupervised Learning")
- "Reinforcement Learning".

1) Apprentissage Non-Supervisé:

L'apprentissage non-supervisé représente la capacité des réseaux de neurones a apprendre sans connaître la sortie désirée. Ce type d'apprentissage possède souvent une moindre complexité dans les calculs en comparaison avec l'apprentissage supervisé (qui a besoin de la sortie désirée.); Il apprend rapidement, parfois un seul passage sur des données bruitées suffit. Il est particulièrement souhaitable dans les processus rapides, ou on n'a pas suffisamment de temps, ou d'informations [Kosko, 1992].

2) Apprentissage Supervisé :

Dans l'apprentissage supervisé la sortie désirée est connue, et c'est en minimisant l'erreur entre la sortie désirée et la sortie du réseau qu'on modifie les paramètres internes du réseau de neurones. Dans ce cas on devrait avoir un ensemble de paires entrées-sorties désirées qu'on appelle exemples. Le processus d'apprentissage supervisé est parfois appelé "Learning by Examples" [Abu-Mostafa, 1989]. Le processus d'apprentissage à partir d'exemples est illustré dans la figure 5.

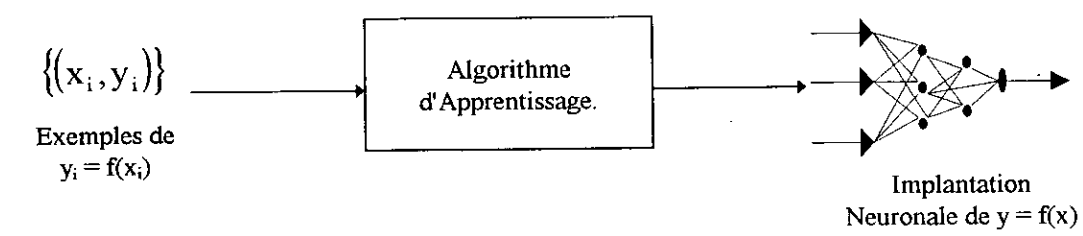


Fig.5: Apprentissage à partir d'exemples.

Le but est donc de produire un réseau de neurones (ou de fonctions) qui implante la fonction inconnue, f, quand un nombre suffisant d'exemples est à notre disposition. Le processus est automatisé grâce à l'algorithme d'apprentissage. Cette implantation est souvent une approximation de f.

L'un des algorithmes les plus répandus est celui de la "Backpropagation" (rétro-propagation), cet algorithme change les poids d'un réseau dont l'architecture est fixée par le superviseur, à chaque fois qu'un exemple $y_i = f(x_i)$ est présenté. Ce changement est fait de telle sorte à minimiser l'erreur entre la sortie désirée y_i et la réponse du réseau a une entrée x_i . Ceci est réalisé grâce à la descente du gradient, à chaque itération le signal d'entrée se propage dans le réseau dans le sens entrée-sortie, une sortie est ainsi obtenue, l'erreur entre cette sortie et la sortie désirée est calculée puis rétropropagée dans le sens sortie-entrée. Malheureusement, cet algorithme souffre des problèmes de la méthode de la descente du gradient, i.e., les paramètres du réseau se trouvent bloqués dans un minimum local. La figure 6 illustre l'algorithme.

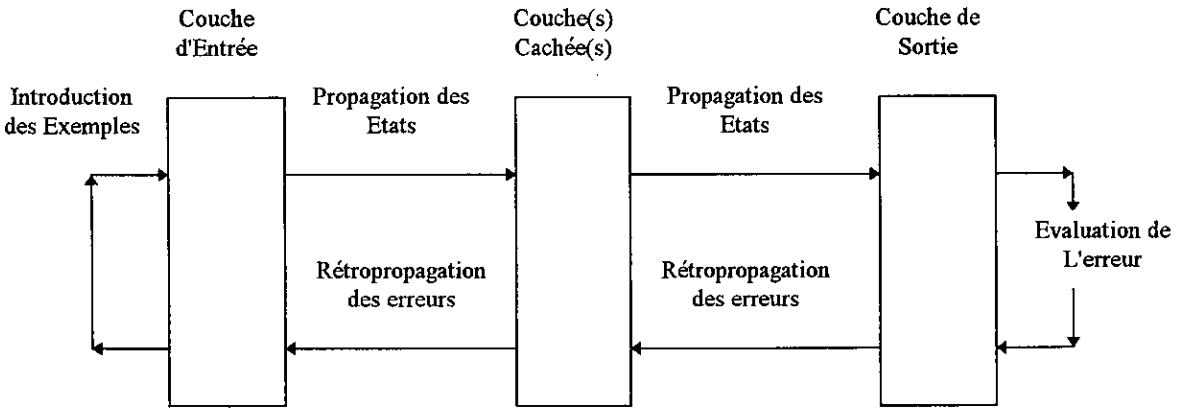


Figure 6 : Apprentissage des Réseaux de Neurones en utilisant l'algorithme "Backpropagation".

L'algorithme "Backpropagation" peut être appliqué soit aux réseaux de neurones statiques, soit aux réseaux de neurones dynamiques; C'est ce point qui sera développé dans les parties suivantes.

2 - 1) Apprentissage des Réseaux De Neurones Statiques :

Un réseau de neurones statique associe un vecteur de sorties à chaque vecteur d'entrées. Pour un vecteur d'entrées $\mathbf{x}$, le réseau lui associe, un vecteur de sorties $\hat{\mathbf{y}}$ suivant la relation:

$$\hat{\mathbf{y}} = \eta(\mathbf{x}) \qquad \text{(I-35)}$$

ou ηest la fonction qui représente le réseau de neurones (n'oublions pas qu'un réseau de neurones n'est rien d'autre qu'un approximateur de fonctions, donc il est lui même une fonction.).

Le but de l'apprentissage supervisé est de minimiser l'erreur quadratique $e^2(t) = (y - \hat{y})^2$, la méthode utilisée pour cela est la "backpropagation" dont le principe a été expliqué dans la partie précédente. Plusieurs variantes ont été proposées, on peut citer les algorithmes suivants:

- FFN-pattern.
- FFN-batch.
- "Backpropagation" avec momentum.
- Pseudo Impedance Control.
- Robust Backpropagation.

Dans les parties suivantes, on va détailler chacun de ces algorithmes.

2-1-1) FFN-pattern :

FFN-pattern est le plus utilisé des algorithmes d'apprentissage des réseaux de neurones statiques, il a été découvert en 1974 par Werbos, puis redécouvert en 1986 par Rumelhart, Hinton et Williams . Dans cette partie on va présenter FFN-pattern .

Soit $E_p = \frac{1}{2}\sum_j (y_{pj} - \hat{y}_{pj})^2$ la mesure de l'erreur sur le p-ième exemple, et $E = \sum E_p$ la mesure de l'erreur globale sur tout les exemples.

L'algorithme FFN-pattern minimise E_p, sa dérivation est simple et peut être trouvée dans n'importe quel document qui traite les réseaux de neurones d'une façon détaillée; On peut citer [Kosko, 1992], [Freeman and Skapura, 1992], [Karayiannis and Venetsanopoulos, 1993]. On va seulement présenter les différentes étapes de l'algorithme :

Etape 1: *Initialiser les poids* (en général on devrait les prendre entre -.5 et .5).
Etape 2: *Présenter l'entrée et la sortie désirée.*
Etape 3: *Calculer la sortie du réseau:* utiliser les équations (I-27) et (I-28).
Etape 4: *Adapter les poids:* Utiliser un algorithme d'adaptation récursif basé sur la descente du gradient, en commenceant des neurones de sorties et remonter jusqu'au neurones d'entrées. Ajuster les poids selon:

$$a_{ji}(k+1) = a_{ji}(k) + c_k \Delta a_{ji}(k) \qquad \text{(I-36)}$$

ou: $\Delta_p a_{ji} = \delta_{pj} y_{pi}$; Dans l'équation précédente $a_{ji}(t)$ est le poids entre les neurones i et j qui appartiennent a deux champs de neurones qui sont proches , c_k est le pas d'adaptation, δ_{pj} est un terme d'erreur pour le j-ième neurone.
Si le neurone j est un neurone de sortie, δ_{pj} peut être calculé en utilisant l'équation suivante:

$$\delta_{pj} = (y_{pj} - \hat{y}_{pj}) g'_j(x_{pj}) \qquad \text{(I-37)}$$

x_{pj} représente la somme pondérée de la couche précédente.
Si le neurone j est un neurone qui appartient aux couches cachées, dans ce cas δ_{pj} est calculé comme suit:

$$\delta_{pj} = g'_j\left(s_{pj}\right)\sum_k \delta_{pk} w_{kj} \tag{I-38}$$

L'algorithme FFN-pattern n'implante pas exactement la descente du gradient, mais elle est efficace, cependant l'explication de ceci n'existe pas; Rumelhart,Hinton et Williams ont démontré que cet algorithme implante la descente du gradient si les fonctions d'activations étaient linéaires; ou si le pas était faible. C'est l'algorithme FFN-Batch qui implante exactement la descente du gradient et qu'on va présenter dans la partie suivante.

2-1-2) FFN-Batch:

Dans cette version, on minimise l'erreur $E = \sum E_p$ qui est une erreur globale (sur *tous* les exemples). Dans cette partie, on présentera la dérivation de Werbos en [Werbos, 1990a], celle ci est plus rigoureuse .

D'abord définissons la notion de *' Ordered Derivatives'*, ce type de dérivées est plus général que les dérivées partielles; en effet lors du calcul de dérivées partielles on suppose que les autres variables sont constantes, ainsi elles n'ont aucun effet sur la variable qu'on dérive, ce qui n'est pas très rigoureux; C'est pour cela que Werbos a introduit ce nouveau type de dérivées.

Soit le système ordonné d'équations [Werbos, 1990a] défini par:

$$x_i = f_i\left(x_{i-1},\ldots\ldots,x_r\right), \quad i = 1,\ldots\ldots,N+1. \tag{I-39}$$

La dérivée ordonnée d'un variable x_{N+1} en tenant compte des autres variables est définie par:

$$\frac{\partial^+ x_{N+1}}{\partial x_i} = \sum_{j>i}^{N+1} \frac{\partial^+ x_{N+1}}{\partial x_j}\frac{\partial f_j}{\partial x_i} \tag{I-40}$$

L'équation (I-40) une équation récursive qui permet de calculer la dérivée $\frac{\partial^+ x_{N+1}}{\partial x_i}$, pour le système d'équations (I-39).

L'erreur $E = \frac{1}{2}\sum_{t=1}^{P}\left(\hat{y}(t) - y(t)\right)^T\left(\hat{y}(t) - y(t)\right)$, ou y(t) représente la sortie désirée du k-ième exemple. L'algorithme du gradient [Ljung, 1987], appliqué à E, pour ajuster les poids du réseau donne :

$$a_{ij}(k+1) = a_{ij}(k) - c_k \frac{\partial E}{\partial a_{ij}}, \tag{I-41}$$

dans ce cas *k* représente le k-ième passage sur *tous* les exemples.

$$\frac{\partial E}{\partial a_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial a_{ij}} = \frac{\partial E}{\partial y_i} g'(s_i) y_j \qquad \text{(I-42)}$$

s_i est l'entrée totale du *i-iéme* neurone., i.e. la somme pondérée de toutes les sorties des neurones situés dans la couche précédente.

Pour la couche de sortie, $\partial E / \partial y_i = \hat{y}_i - y_i$, le problème est de trouver l'expression de cette dérivée pour les couches cachée; c'est l'apport de Rumelhart *et al* en 1986.

Rumelhart *et al* ont résolu ce problème, en utilisant la règle de la chaîne pour le calcul de la dérivée i.e.:

$$\frac{\partial E}{\partial y_i} = \sum_{t=1}^{P}\sum_{j}\frac{\partial E}{\partial y_j}\frac{dy_j}{ds_j}\frac{\partial s_j}{\partial x_i} = \sum_{t=1}^{P}\sum_{j}\frac{\partial E}{\partial y_j} g'(s_j)\, a_{ji}, \qquad \text{(I-43)}$$

avec y_i appartenant à la couche *l*, y_j appartient à la couche *l+1*. En commenceant par la couche de sortie, on remonte jusqu'à la couche d'entrée.

Cette formulation est imprécise, vu la définition de la dérivée partielle.

Puisque, le réseau est un système ordonné. L'équation (I-44) peut être écrite sous la forme :

$$\frac{\partial^+ E}{\partial y_i} = \frac{\partial E}{\partial y_i} + \sum_{j>i}^{N}\frac{\partial^+ E}{\partial y_j}\frac{\partial y_j}{\partial y_i} \qquad \text{(I-45)}$$

x_{N+1} est remplacé par E dans (I-40). Et l'algorithme FFN-Batch en découle :

$$z_i = \sum_{j>i} a_{ji} g'(s_j) z_j + \varepsilon_i,$$

$$\frac{\partial E}{\partial a_{ij}} = \sum_{t=1}^{P} g'(s_i) z_i(t) y_j(t), \qquad \text{(I-46)}$$

$$a_{ij}(k+1) = a_{ij}(k) - c_k \frac{\partial E}{\partial a_{ij}},$$

avec: $z_i = \partial^+ E / \partial y_i$ et $\varepsilon_i = \partial E / \partial y_i$.

Dans ce cas, les neurones sont numérotés suivant des nombres consécutifs indépendamment de la structure des couches, i.e. $y_1, y_2, \ldots, y_i, \ldots, y_N$. Les différentes étapes de l'algorithme sont les mêmes que celles de FFN-pattern.

2-1-3) "Backpropagation" avec momentum :

Une version améliorée, proposée par Rumelhart [Kosko, 1992] est une équation aux différences linéaire d'ordre deux, dans ce cas l'équation (I-36) devient:

$$a_{ji}(k+1) = a_{ji}(k) + c_k \Delta a_{ji}(k) + b_k \Delta a_{ji}(k-1) \qquad \text{(I-47)}$$

Rumelhart appelle le terme additif "*momentum*". Ce terme accélère le changement des poids, il accélère ainsi la convergence, mais son vrai rôle reste inconnu [Dote, 1990].

2-1-4) "Robust Backpropagation":

Une autre version proposée par .White en 1989, est appelée **"Robust Backpropagation"**, elle est basée sur les statistiques robustes; En effet White a démontré que la "Backpropagation" est un simple de cas d'approximation stochastique, ainsi elle appartient au domaine des statistiques.

La robustesse dont on parle est celle de l'insensibilité aux échantillons qui sont "bizarres", **"Robust Backpropagation"** remplace le terme $\left(y_{pj} - \hat{y}_{pj}\right)$, par une **fonction atténuatrice,** $s\left(\left(y_{pj} - \hat{y}_{pj}\right)\right)$. La fonction atténuatrice, s(x)=x n'est pas statistiquement robuste. Tandis que les fonctions atténuatrices suivantes le sont :

$$s(x) = \tanh\left(\frac{x}{2}\right),\ s(x) = \frac{2x}{1+x^2},\ s(x) = \max\left(-c, \min\left(c, x\right)\right). \qquad \text{(I-48)}$$

En général, on peut utiliser les fonctions d'activations citées plus haut. En appliquant cette technique, on obtiendra un estimateur statistique robuste, qui ignore les exemples étranges durant l'observation [Kosko, 1992].

2-1-5) "Pseudo Impedance Control":

Une autre version, est celle proposée par Nagata *et al* [Nagata *et al*, 1990] elle est appelée **"Pseudo Impedance Control"**, elle a été introduite pour décroître la possibilité de tomber dans un minimum local; Elle a été déduite par analogie avec les systèmes mécaniques, dans ce cas les poids sont régit par l'équation différentielle du troisième ordre suivante :

$$J\frac{d^3w}{dt^3} + M\frac{d^2w}{dt^2} + D\frac{dw}{dt} = -\frac{\partial E}{\partial w(t)} \qquad \text{(I-49)}$$

On retrouve la "backpropagation" classique en annulant J et M.
Le lecteur intéressé par cette méthode est renvoyé vers l'article de Nagata *et al*, pour savoir comment choisir les coefficients J, M, D qui doivent être positifs pour assurer la stabilité du système dynamique.

Dans tous les algorithmes présentés on peut remarquer que le pas d'apprentissage c_k pouvait être variable, dans ce cas il doit être décroissant en fonction de la période d'échantillonnage k, Il doit décroître lentement, mais pas trop lentement, rapidement mais pas trop. Dans ce cas la séquence c_k , est contrainte à décroître lentement comme la somme divergente:

$$\sum_{k=1}^{\infty} c_k = \infty \qquad \text{(I-50)}$$

La décroissance lente de c_k , génère une plus grande rapidité dans l'apprentissage dans l'équation (I-36).
Mais aussi elle doit décroître rapidement comme la somme convergente définie par:

$$\sum_{k=1}^{\infty} c_k^2 < \infty \qquad \text{(I-51)}$$

La décroissance rapide de c_k, génère un moindre oubli de la part de **w(k)**. On remarque que la série harmonique $c_k = 1/k$, vérifie les conditions (I-50) et (I-51).
Ces deux contraintes résolvent le dilemme de Grossberg souvent appelé **"stabiliy-plasticity dilemma"** [Kosko, 1992]; Dans ce dilemme Grossberg pose la question " Comment un réseau peut-il être suffisamment stable de telle façon à mémoriser les échantillons précédents, et être suffisamment rigide pour pouvoir apprendre de nouveaux exemples ?". La solution proposée ci-dessus est une solution basée sur les pas d'adaptation et non sur l'architecture du réseau. L'équation (50) permet la rigidité tandis que l'équation (51) la contraint. L'équation (51) permet la stabilité tandis que l'équation (50) la contraint.

Les méthodes d'apprentissage présentées ci-dessus ne sont applicables qu'aux réseaux de neurones statiques; En ce qui concerne les réseaux de neurones dynamiques on doit développer d'autres méthodes d'apprentissage à cause de leurs architecture différente de celle des réseaux statiques. La section suivante présentera les différentes méthodes d'apprentissage des réseaux de neurones dynamiques.

2 - 2) Apprentissage des Réseaux De Neurones Dynamiques :

L'architecture d'un réseau de neurones dynamique ne suffit pas pour prédire sont comportement dynamique; Il faudrait aussi spécifier l'algorithme d'apprentissage [Hunt *et al*, 1992]. Des réseaux de neurones dynamiques qui possèdent les mêmes architectures, mais qui sont entraînés avec différents algorithmes, évoluent d'une manière différente; Ainsi *un réseau de neurones dynamique est la combinaison de deux systèmes dynamiques: transmission et ajustement.*

L'apprentissage des réseaux de neurones dynamiques, peut se faire de deux différentes manières. La première technique est "Fixed point Learning" qui impose au réseau dynamique un point d'équilibre; la seule contrainte sur le régime transitoire est qu'il disparaisse. La seconde technique est "Trajectory learning" qui impose au réseau une certaine dynamique, c'est une généralisation de la première technique car quand $t \to \infty$, on atteint le point d'équilibre.

2-2-1) "Fixed point Learning":

Dans ce cas deux approches existent, soit la minimisation de l'erreur instantanée (équivalent de FFN-pattern), soit la minimisation de l'erreur globale (équivalent de FFN-batch).

La minimisation de l'erreur instantanée requiert une combinaison entre une certaine architecture du réseau dynamique et un choix de la fonction d'activation qui doit posséder les caractéristiques citées précédemment pour avoir une descente du gradient [Karkasoglu *et al*, 1993].

La minimisation de l'erreur globale se fait en utilisant l'algorithme "*recurrent backpropagation*".

2-2-1-a) Minimisation de l'erreur instantanée:

Cet algorithme est applicable aux réseaux de neurones qui possèdent trois couches, dont la couche cachée est entièrement connectée. L'architecture du réseau est montrée dans la figure 7. Le réseau est régit par :

$$\dot{\mathbf{x}} = -\mathbf{x} + \mathbf{W}\mathbf{g}(\mathbf{x}) + \mathbf{B}\,\mathbf{z}(\mathbf{k}) \qquad \text{(I-52)}$$

$$y(k) = h^T x^* \qquad \text{(I-53)}$$

avec : $x \in \Re^q$, $g(.):\Re^q \rightarrow \Re^q$ est un vecteur dont les éléments sont les fonctions g(.) qui possèdent les propriétés (I-33). $W \in \Re^{q\times q}, B \in \Re^{q\times p}, h \in \Re^q$, sont les poids d'interconnexion.

Dans (I-53) $\mathbf{x}^*$, spécifie un point d'équilibre stable de (I-52) pour l'échantillon d'entrée z(k) à l'instant k. En supposant que le point d'équilibre est atteint, (I-52) et (I-53) deviennent :

$$x^* = W\,g(x^*) + B\,z(k);\; y(k) = h^T x^* \qquad \text{(I-54)}$$

Ce qui est équivalent à :

$$x_i^* = \sum_{j=1}^{q} w_{ij} g_j(x_j^*) + \sum_{l=1}^{p} b_{il} z_l(k); \quad y(k) = \sum_{i=1}^{q} h_i x_i^* \qquad \text{(I-55)}$$

avec: $x_i^*, g_j, w_{ij}, b_{il}, h_i$ et z_l sont respectivement des éléments des matrices: $x^* \in \Re^q$, $g \in \Re^q, W \in \Re^{q\times q}, B \in \Re^{q\times p}, h \in \Re^q, z \in \Re^p$.

L'apprentissage d'un tel réseau se fait à l'aide de la "Backpropagation" en minimisant l'erreur de sortie :

$$E(k) = \left[y_d(k) - y(k)\right]^2 = \left[\, y_d(k) - \mathbf{h}^T\mathbf{x}^* \,\right]^2 \qquad \text{(I-56)}$$

$y_d(k)$ étant la sortie désirée.

L'algorithme d'apprentissage peut être facilement déduit, en appliquant l'algorithme de descente du gradient à ce réseau.
En effet :

$$h_i(k+1) = h_i(k) - \mu_1 \frac{\partial E(k)}{\partial h_i}$$
$$w_{ij}(k+1) = w_{ij}(k) - \mu_2 \frac{\partial E(k)}{\partial w_{ij}} \qquad \text{(I-57)}$$
$$b_{ij}(k+1) = b_{ij}(k) - \mu_3 \frac{\partial E(k)}{\partial b_{ij}}$$

En utilisant les équations (I-54) et (I-56) on obtient :

$$h_i(k+1) = h_i(k) - \mu_1 \left[y_d(k) - y(k)\right] x_i^*, \quad i = 1,\ldots,q$$
$$w_{ij}(k+1) = w_{ij}(k) - \mu_2 h_i(k)\left[y_d(k) - y(k)\right] g_j\left(x_i^*\right), \; i,j = 1,\ldots,q; i \neq j \qquad \text{(I-58)}$$
$$b_{ij}(k+1) = b_{ij}(k) - \mu_3 h_i(k)\left[y_d(k) - y(k)\right] z_j(k), \; i = 1,\ldots,q; j = 1,2,\ldots p$$

μ_1, μ_2, μ_3 sont des pas d'adaptation, choisis d'une façon appropriée.

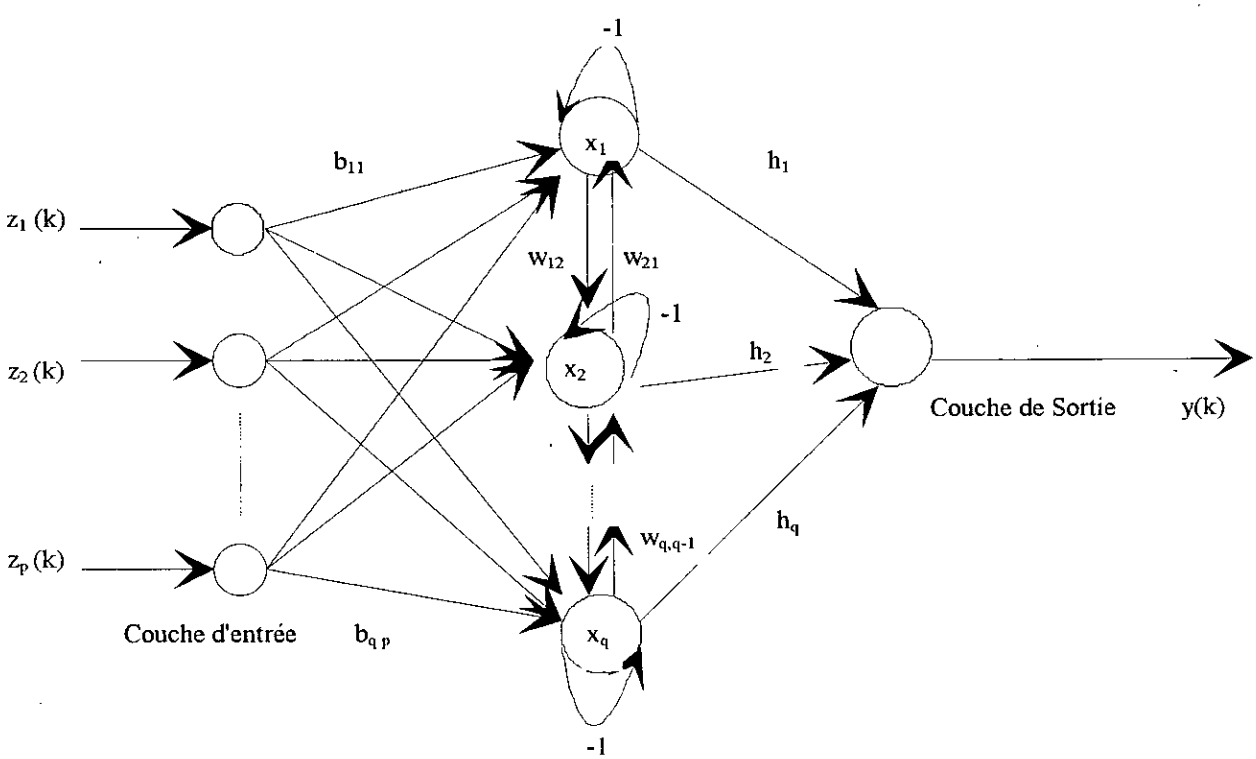


Figure 7: Architecture du Réseau Dynamique

Pour que le réseau soit stable, plusieurs conditions doivent être vérifiés; Ces conditions concernent les poids initiaux, les pas d'apprentissage; Si nous notons par $M \in \Re^{q\times q}$ la partie symétrique de la matrice $I - W\Gamma$, avec $\Gamma = \mathrm{diag}\left[\gamma_1\ \gamma_2 \ldots . \gamma_q\right]$, $\gamma_i = \left.\dfrac{d\,g_i(x_i)}{d\,x_i}\right|_{x_i = x_i^*}$. La réelle positivité de M suffit pour assurer que l'algorithme (I-58) soit un descente du gradient, qui minimise E(k). Si on choisit une fonction d'activation $g_i(x_i)$, qui se sature rapidement, alors $\Gamma = 0$, et donc M sera définie positive. On devrait noter que la sélection d'une fonction d'activation qui se sature rapidement permet une stabilité exponentielle des points d'équilibre, ce qui assure une convergence rapide du réseau dynamique.

En ce qui concerne le pas d'apprentissage, on pourrait choisir des pas variables décroissants en fonction du temps (par exemple $\mu(k) = 1/k$); Mais, parfois des pas fixes suffisent; Les pas qu'on choisira lors des simulations satisfont les conditions suivantes : $\mu_1 \le 2/\|x^*\|$, $\mu_2 \le 2/q$, $\mu_3 \le 2/\xi$, avec $\xi = \max_i \left\{ z_i^2(k) + \sum_{j\neq i}^{p} \left| z_i(k)\, z_j(k) \right| \right\}$ [Karakasoglu *et al*, 1993].

C'est ce type de réseaux dynamiques qu'on va utiliser, puisqu'il possède la capacité d'un apprentissage rapide; Il est donc souhaitable en identification et en commande des systèmes dynamiques.

2-2-1-b) "Recurrent Backpropagation ":

Cet algorithme minimise l'erreur :

$$E = \frac{1}{2}\sum_{i=1}^{p}\left(y_\infty^k - y\right)^T\left(y_\infty^k - y\right), \qquad \text{(I-59a)}$$

y_∞^k sont des vecteurs des points d'équilibres désirés, qui sont solutions de l'équation (I-32) en annulant la partie gauche.

Soit e_i, l'erreur définie par:

$$e_i = \begin{cases} y_{\infty_i} - y_i, & \text{si le neurone i appartient au champ de sortie} \\ 0, & \text{sinon} \end{cases} \qquad \text{(1-59b)}$$

Durant l'apprentissage le réseau ne reçoit aucune entrée extérieure, il est excité par des conditions initiales puis évolue en tenant compte de y_∞^k, *qui est la référence fixée.*

La dérivation de l'algorithme [Karayiannis and Venetsanopoulos, 1993] révèle que l'adaptation des poids dépend d'un réseau auxiliaire dont la dynamique est :

$$\dot{z}_i = -z_i + \sum_j a_{ji}\, g'(s_j)\, z_j + e_i \qquad \text{(I-60)}$$

Cette équation contient une somme de non-linéarités, contrairement à l'équation (I-31). L'entrée extérieure de ce réseau auxiliaire est e_i , qui est l'erreur entre la sortie des neurones qui appartiennent au champ de sortie, et les sorties désirées.

L'algorithme de "Recurrent Backpropagation" peut être résumé sous la forme suivante :

Etape 1: Calculer les points d'équilibre x_i^*, à partir de l'équation (I-32).
Etape 2: Calculer les points d'équilibre z_i^*, à partir de l'équation (I-60).
Etape 3: Adapter les poids en utilisant l'équation :

$$\dot{a}_{ij} = \alpha\, g(s_i(\infty))\, x_i^*\, z_i^* \qquad \text{(I-61)}$$

Puisqu'on n'est intéressés que par les points d'équilibres, les conditions initiales pourraient être quelconques.

2-2-2) "Trajectory Learning":

Dans ce cas c'est l'erreur E définie par:

$$E = \frac{1}{2}\int_{t_0}^{t_1}\left[\left(d(\tau) - y(\tau)\right)^T \left(d(\tau) - y(\tau)\right) d\tau\right] \qquad \text{(I-62)}$$

où: $d(\tau)$ est le vecteur des trajectoires désirées, t_1 peut être constant (cas des techniques "off-line") où variable (cas des techniques "on-line"). La version discrète de l'équation (I-62) est :

$$E = \frac{1}{2}\sum_{t_0}^{t_1}\left[d(\tau) - y(\tau)\right]^T \left[d(\tau) - y(\tau)\right], \qquad \text{(I-63)}$$

Deux méthodes peuvent être utilisées minimiser cette erreur . La première méthode est la "Backpropagation Through Time"(BPTT), la seconde est la "Real Time Recurrent Learning"(RTRL).

2-2-2-a) "Backpropagation Through Time":

BPTT est une méthode basée sur le *remplacement d'un réseau récurrent à une seule couche, par un réseau statique équivalent qui possède t_1 couches*; Ceci peut être facilement compris dans le cas discret. La "Backpropagation" classique est appliquée à ce réseau, et puisque chaque couche contient les mêmes poids "vus" à différents instants; leurs valeur finale est la somme des poids aux instants précédents [Werbos, 1990a].

L'algorithme de la BPTT sera :

$$E = \frac{1}{2}\sum_{\tau=t_0}^{t_1}\sum_i \left(d_i(\tau) - y_i(\tau)\right)^2, \quad t_1 = \text{const},$$

$$e_i(\tau) = \begin{cases} d_i(\tau) - y_i(\tau), & \text{si le neurone i appartient au champ de sortie} \\ 0, & \text{sinon} \end{cases}$$

$$z_i(\tau) = \sum_j a_{ji}\, g'\left(s_j(\tau+1)\right) z_j(\tau+1) - e_i(\tau), \quad z_i\left(\tau_1\right) = 0, \tag{I-64}$$

$$\frac{\partial E}{\partial a_{ij}} = \sum_{\tau=t_0}^{t_1} g'\left(s_i(\tau)\right) z_i(\tau) y_i(\tau),$$

$$a_{ij}^{t_1} = a_{ij}^{t_0} - \alpha \frac{\partial E}{\partial a_{ij}},$$

Selon Williams et Zipser, cet algorithme peut être utilisé "on-line", dans ce cas les calculs doivent être effectués à chaque instant, ce qui requiert un espace mémoire illimité, ainsi qu'une capacité de calcul illimitée.

La version continue a été introduite par Pearlmutter [Pearlmutter, 1989], lorsqu'il a critiqué la dérivation de la "Recurrent Backpropagation" qui est basée sur la supposition qu'un seul point d'équilibre existait; Il a ainsi proposé une procédure qui permet de calculer $\partial E/\partial a_{ij}$; On va maintenant présenter la version continue de la BPTT :
Le problème qui se pose est la minimisation de l'erreur (I-62) sous la contrainte (I-32-b); c'est un simple problème de calcul des variations; Le Lagrangien sera donc égal à:

$$L = \int_{t_0}^{t_1} \left\{ \frac{1}{2}\sum_i \left(d_i - y_i\right)^2 - \sum_i z_i \left[T_i \dot{y}_i + y_i - g\left(s_i\right) - u_i \right] \right\} d\tau, \tag{I-65}$$

z_i, sont les multiplicateurs de Lagrange; L'application de l'équation d'Euler-Lagrange [Elsgolc, 1961], sur (I-65) donne :

$$T_i \dot{z}_i = z_i - \sum_j a_{ji}\, g'\left(s_j\right) z_j - e_i \tag{I-66}$$

Puisque $y_i(t_0)$ ne dépend pas des poids; et en imposant les conditions aux limites suivantes:

$$z_i\left(t_1\right) = 0 \tag{I-67}$$

On obtient alors :

$$\frac{\partial E}{\partial a_{ij}} = \int_{t_0}^{t_1} g'(s_i)\, z_i\, y_j\, d\tau, \tag{I-68}$$

$$\dot{a}_{ij} = -\alpha \frac{\partial E}{\partial a_{ij}} \tag{I-69}$$

Ces deux équations avec l'équation (I-66) définissent complètement l'algorithme. Notons que l'équation (I-66) est définie sur $[t_0,t_1]$ avec $z_i(t_1)=0$, ainsi cette équation doit être intégrée en remontant dans le temps .

Cette méthode n'est pas récurrente car elle utilise un réseau statique équivalent lors de l'apprentissage; Mais, lors de l'implantation le réseau fonctionne comme un réseau récurrent.

2-2-2-b) "Real-Time Recurrent Learning":

La BPTT est une méthode qui est essentiellement "Off-Line", et qui lors de l'apprentissage en temps réel requiert un gigantesque capacité de stockage; La méthode présentée ci-dessous, i.e. "Real-Time Recurrent Learning" (RTRL) résout ce problème. Elle a été introduite en 1989 par Williams et Zipser [Williams and Zipser, 1989a].

Considérons un réseau qui possède *n* neurones, *m* entrées extérieures. Soit y(t) le vecteur $n\times 1$ à l'instant *t* qui définit le vecteur de sorties, x(t) le vecteur $m\times$ qui représente les entrées extérieures à l'instant *t* . On notera par z(t) le vecteur $(m+n)\times$, qui rassemble les vecteurs x(t) et y(t), soit *U* l'ensemble des indices k pour lequel z_k, est une sortie du réseau, *I* l'ensemble des indices pour lequel z_k est une entrée externe. Ainsi :

$$z_k(t) = \begin{cases} x_k(t) & \text{si } k \in I \\ y_k(t) & \text{si } k \in U \end{cases} \tag{I-70}$$

Soit **W** la matrice des poids du réseau, selon la notation précédente, **W** est une matrice $n\times(n+m)$.

Notons par :

$$s_k(t) = \sum_{l\in U\cup I} w_{kl} z_l(t) \tag{I-71}$$

l'entrée du *k-ième* neurone à l'instant *t*, pour $k \in U$, sa sortie à l'instant *k+1* sera notée par:

$$y_k(t+1) = g_k\left(s_k(t)\right) \tag{I-72}$$

g_k est la fonction d'activation du *k-ième* neurone.
On va maintenant dériver l'algorithme que Williams et Zipser appellent "temporal supervised learning". Soit T(t) l'ensemble des indices $k \in I$, pour lesquels une trajectoire désirée existe à l'instant *t*.

Définissons le vecteur **e** dont les dimensions sont $n \times 1$:

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{si} \quad k \in T(t) \\ 0 & \text{ailleurs} \end{cases} \tag{I-73}$$

Cette formulation permet différentes trajectoires pour différents neurones à des instants différents. Dénotons par:

$$J(t) = \frac{1}{2}\sum_{k \in U}\left[e_k(t)\right]^2 \tag{I-74}$$

L'erreur totale à l'instant *t*. Supposons en plus, que le réseau est lancé de l'instant t_0 à l'instant t_1; L'objectif est de minimiser l'erreur globale :

$$J_{totale} = \sum_{t_0+1}^{t_1} J(t) \tag{I-75}$$

Ceci est possible grâce à la descente du gradient.
Puisque l'erreur globale (I-75) est la somme des erreurs à différents instants; Le changement global des poids est :

$$\Delta w_{ij} = \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t) \tag{I-76}$$

$$\Delta w_{ij}(t) = -\alpha \frac{\partial J(t)}{\partial w_{ij}} \tag{I-77}$$

Calculons le gradient :

$$-\frac{\partial J(t)}{\partial w_{ij}} = \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \tag{I-78}$$

Les équations (I-71) et (I-72) permettent de calculer $\partial y_k(t)/\partial w_{ij}$; en effet :

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = g_k'\left(s_k(t)\right)\left[\sum_{l \in U} w_{kl} \frac{\partial y_l(t)}{\partial w_{ij}} + \delta_{ik} z_j(t)\right] \tag{I-79}$$

δ_{ik} est l'opérateur δ de Kronecker. En supposant que les états initiaux du réseau ne dépendent pas des poids on aura :

$$\frac{\partial y_k(t_0)}{\partial w_{ij}} = 0 \tag{I-80}$$

Ces équations sont valables pour tout $k \in U, i \in U$, et $j \in U \cup I$.

En notant par:

$$p_{ij}^{k}(t) = \frac{\partial y_k(t)}{\partial w_{ij}} \tag{I-81}$$

On aura, ainsi crée un réseau auxiliaire défini par:

$$p_{ij}^{k}(t+1) = g_k'\left(s_k(t)\right)\left[\sum_{l \in U} w_{kl} p_{ij}^{l}(t) + \delta_{ik} z_j(t)\right] \tag{I-82}$$

dont les conditions initiales sont :

$$p_{ij}^{k}(t_0) = 0 \tag{I-83}$$

L'algorithme sera donc :

Etape 1: Calculer à chaque instant t de t_0 à t_1, les quantités $p_{ij}^{k}(t)$, en utilisant les équations (I-82) et (I-83) .

Etape 2: Calculer le changement des poids

$$\Delta w_{ij}(t) = \alpha \sum_{k} e_k(t)\, p_{ij}^{k}(t) \tag{I-84}$$

Etape 3: Calculer le changement global, qui est défini par (I-76).

L'algorithme ci-dessus supposait que les poids étaient fixes entre t_0 et t_1 (puisque le changement n'était possible qu'a l'instant t_1). L'algorithme RTRL relaxe cette supposition, en supposant que la borne supérieure t_1 est variable (elle est égale à t), ce qui permet un ajustement en temps réel, sans accumuler tous les changements.

Le problème avec cet algorithme, i.e. RTRL, est qu'après des instants assez longs, il n'implante plus la déscente du gradient. Mais, comme noté par Williams et Zipser, ceci est équivalent à l'algorithme FFN-pattern, qui n'implante pas la déscente du gradient.

3) "Reinforcement Learning":

Dans ce type d'apprentissage, une seule valeur qui mesure la qualité de la réponse du réseau est réinjectée dans le réseau et est utilisée lors de l'apprentissage. Lors de l'apprentissage on n'a pas besoin de la sortie désirée; le réseau a seulement besoin de savoir si sa sortie est bonne ou mauvaise. Cette méthode est basée sur l'adaptation d'un réseau auxiliaire appelé "critic network", c'est la sortie de ce réseau qui mesure la qualité de la sortie du réseau. La figure 8 montre le fonctionnement de cette méthode [Werbos, 1990b].

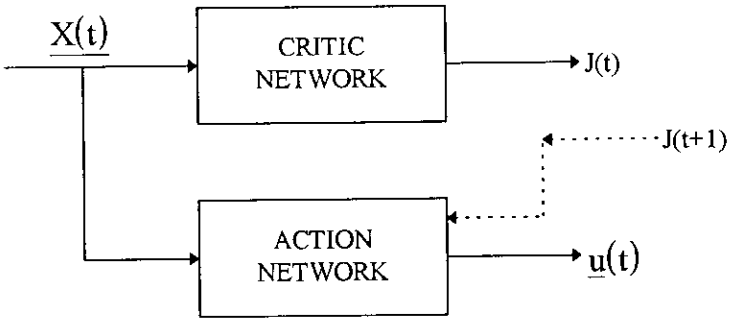


Figure 8 : Schéma du "Reinforcement Learning"

Le réseau noté par "action network", possède comme sortie $\underline{u}(t)$, c'est la sortie qu'on voudrait être égale à la sortie désirée, Le second réseau ("critic network") possède comme entrée X(t), et comme sortie un seul nombre, J(t), qui mesure la qualité de la sortie $\underline{u}(t)$ par rapport à la sortie désirée. Deux problème se posent :

1) Comment ajuster les poids de l'"Action Network", en fonction de J(t+1) ?.
2) Comment ajuster les poids du "Critic Network", en réponse à la sortie.

Ceci est possible grâce à la programmation dynamique stochastique ("Heuristic Dynamic Programming", HDP.) [Werbos, 1990b] .

IV. Approximation Des Fonctions en Utilisant Les Réseaux De Neurones:

C'est en se basant sur les théorèmes de Kolmogorov et Sprecher qu'en 1987, Hecht-Nielsen a pu démontrer que n'importe quelle fonction pouvait être approximée par un réseau de neurones qui possède quatre couches; Dans cette partie on va démontrer cette propriété, et puis parler des limites de ces théorèmes.

En 1900, dans son célèbre treizième problème, Hilbert affirma qu'il existe des fonctions analytiques à trois variables qui ne peuvent pas être représentées par une superposition finie de fonctions continues à deux variables.

Et c'est en 1957 que Kolmogorov prouva que cette affirmation était fausse; la formulation originale du théorème de Kolmogorov est la suivante:

" Toute fonction multivariables $f(x_1, x_2, \ldots\ldots, x_n)$ définie sur $I^n (n \geq 2)$ peut être représentée sous la forme :

$$f(\mathbf{x}) = \sum_{j=1}^{2n+1} \chi_j \left(\sum_{i=1}^{n} \psi_{ij}(x_i) \right), \qquad \text{(I-85)}$$

ou: χ_j, ψ_{ij} sont des fonctions continues monovariable et ψ_{ij} sont des fonctions monotones indépendantes de f."

En 1965, Sprecher a raffiné le théorème de Kolmogorov et obtint le théorème suivant:

" Pour chaque nombre entier $n \geq 2$, il existe une fonction $\psi(x)$ réelle, monotone croissante, $\psi([0,1]) = [0,1]$, dépendante de n qui possède la propriété suivante : Pour chaque nombre fixé $\delta > 0$ il existe un nombre rationnel ε, $0 < \varepsilon < \delta$, tel que n'importe quelle fonction multivariables, $f(\mathbf{x})$, définie sur I^n, peut être représentée sous la forme :

$$f(\mathbf{x}) = \sum_{j=1}^{2n+1} \chi \left[\sum_{i=1}^{n} \lambda^i \psi(x_i + \varepsilon(j-1) + j - 1) \right] \qquad \text{(I-86)}$$

ou la fonction χ est une fonction réelle et continue; λ est une constante indépendante de f."

Ces deux théorèmes stipulent que toute fonction continue multivariables peut être représentée sous la forme de superposition d'un petit nombre de fonctions monovariable. En réseaux de neurones, ceci implique que toute fonction continue qui possède n variables peut être approximée par un réseau de neurones qui possède n(2n+1) neurones dans la première couche cachée, et (2n+1) dans la seconde couche cachée.

Mais, comme l'a prédi Lorentz en 1962, en affirmant "Will it [Kolmogorov's theorem] have useful applications? ... One wonders whether Kolmogorov's theorem can be used to obtain positive results of greater [than trivial] depth".; ce 'merveilleux' théorème est inutilisable pour concevoir un réseau de neurones car :

(1) Le théorème de Kolmogorov requiert différentes fonctions d'activation pour chaque neurone.

(2) En 1954, Vitushkin, formula le théorème suivant :
" Il existe des fonctions n ($n \geq 2$) variables, r ($r = 1,2,...$) fois dérivables, non représentables par la superposition de fonctions continues r fois dérivables m variables ($m < n$); Il existe des fonctions continues à deux variables r fois dérivables, qui ne sont pas représentables sous la forme de somme de fonctions monovariable.".
A partir de ce théorème, Vitushkin démontra qu'il existe un certain entier r au del du quel la r-ième dérivée des fonctions ψ_{ij} n'existaient plus. Ainsi, on ne pourrait appliquer les méthodes d'apprentissage connues; En plus ceci poserait un problème de généralisation et robustesse au bruit.

(3) Les fonctions χ_j dépendent de la fonction qu'on veut approximer.

Il est clair, à partir de la discussion précédente, que le fait d'approximer des fonctions arbitraires, n'est pas suffisant pour avoir une bonne approximation.
En 1990, dans [Poggio and Girosi, 1990] les auteurs ont noté que la question clé n'est pas l'approximation des fonctions arbitraires, mais celle de la *meilleure approximation.* Un approximateur de fonctions possède la propriété précédente, si parmi toute les fonctions approximatrices il en existe une qui possède une distance minimale de la fonction donnée. Rigoureusement on peut dire:

" Si f(X) est une fonction continue définie sur un ensemble **X,** et F(W,X) est une fonction approximatrice qui dépend de $W \in P$ et de X, le problème de l'approximation est de déterminer les paramètres W^*, tels que:

$$\rho\left(F(W^*,X),f(X)\right) \leq \rho\left(F(W,X),f(X)\right) \tag{I-87}$$

pour tous W dans l'ensemble P.

Si une solution à ce problème existe, alors la solution possède la propriété de meilleure approximation. L'existence de la meilleure approximation dépend évidement de la classe de fonctions F(W,X)."

A partir de cette définition, il est clair que F(W,X) peuvent correspondre à des réseaux de neurones statiques; En effet :

- Le problème classique d'approximation linéaire :

$$F(W,X) = W \cdot X,\ W \in R^n, X \in R^n \tag{I-88}$$

Correspond une réseau sans couches cachées.

- Le problème classique d'approximation :

$$F(W,X) = \sum_{i=1}^{m} W_i \Phi_i(X) \tag{I-89}$$

Correspond un réseau statique qui possède une seule couche cachée.

- Le problème d'approximation :

$$F(W,X) = \phi\left(\sum_n w_n \phi \sum_i v_i \phi\left(\cdots \phi \sum_j u_j X_j\right)\right), \tag{I-90}$$

avec ϕ est une fonction sigmoïde.
Correspond à un réseau multicouches, dont les éléments de base éffectuent une somme de leurs entrées, avec des poids $\mathbf{W}=\{w_n, v_i, u_j, \cdots\}$, puis éffectuent une transformation sigmoïdale.
Ceci est un nouveau problème d'approximation, dont la motivation est la démonstration de McCulloch et Pitts, selon laquelle toute fonction logique peut être réalisée par un réseau de neurones à une seule couche cachée, dont la fonction d'activation est la fonction seuil.

En 1988, Irie-Miyake a prouvé qu'un réseau statique à trois couches, avec un nombre infini de neurones dans la couche cachée peut approximer n'importe quelle fonction , pourvu que les fonctions d'activation soient bornées et absolument intégrables. En 1989, Funahashi a étendu la démonstration de Irie-Miyake en y incluant les fonctions sigmoïdales, de telle façon à ce que toute fonction continue pouvait être réalisée par un réseau de neurones à trois couches, dont la couche cachée possède des fonctions d'activations bornées monotones croissantes. Un résultat similaire a été obtenu par Hecht-Nielsen en 1989, en démontrant qu'un réseau multicouches pouvait implanter une fonction sinusoïdale, permettant ainsi aux réseaux multicouches la possibilité d'approximer n'importe quelle fonction en utilisant le développement en séries de Fourier.
Cybenko [Cybenko, 1989] a trouvé un résultat similaire en démontrant que toute somme finie de fonctions sigmoïdales était dense dans l'éspace des fonctions continues définies sur un hypèrcube unitaire de dimension n.
Tous, ces théorèmes possèdent la forme suivante :

" Soit $\phi(x)$, une fonction non-constante, bornée et monotone croissante. Soit K une région bornée de $\mathbf{R^n}$, et $f(x_1,\cdots,x_n)$ une fonction réelle continue, définie sur K. Alors, pour tout $\varepsilon>0$, il existe un entier N et des constantes réelles c_i, $\theta_i\,(i=1,\ldots..,N)$, $w_{ij}\,(i=1,\ldots,N;\ j=1,\ldots.,n)$ telles que :

$$\bar{f}(x_1,\ldots,x_n)=\sum_{i=1}^{N} c_i\phi\left(\sum_{j=1}^{n} w_{ij}x_j-\theta_i\right) \qquad \text{(I-91)}$$

satisfait $\max_{x\in K}\left|f(x_1,\cdots,x_n)-\bar{f}(x_1,\ldots,x_n)\right|<\varepsilon$.". Ce théorème implique, que pour tout $\varepsilon>0$, il existe un réseau de neurones à trois couches, dont les fonctions d'activations des neurones situés dans la couche cachée sont $\phi(x)$, et celles des neurones situés dans la couche de sortie sont linéaires, et dont la relation entrée sortie est $\bar{f}(x_1,\ldots,x_n)$, tel que

$$\max_{x\in K}\left|f(x_1,\cdots,x_n)-\bar{f}(x_1,\ldots,x_n)\right|<\varepsilon. \qquad \text{(I-92)}$$

Malheureusement tous ces merveilleux résultats, n'ont aucune implication pratique; Car ils ne donnent ni la méthode par laquelle les poids peuvent être obtenus, ni le nombre de neurones nécessaire. Cybenko - lui même - a conclu son article de 1989 comme suit :

" While the approximating properties we have described are quite powerful, we have focused only on existence. The important questions that remain to be answered deal with feasibility, namely how many terms in the summation (or equivalently, how many neural nodes) are required to yield an approximation of a given quality?, What properties of the function being approximated play a role in determining the number of terms? At this point, we can only say that we suspect quite strongly that *the overwhelming majority of approximation problems will require astronomical number of terms...*". On ne pouvait mieux conclure !..

V. Les Réseaux de Neurones et La Commande :

Les réseaux de neurones possèdent plusieurs propriétés qui leurs permettent d'être des candidats naturels, lors de l'identification et la commande des systèmes dynamiques. Vus, sous cet angle les réseaux de neurones sont vus comme étant de modélisateurs de processus, tous ce que nous savons sur la dynamique du système (et même ce que nous ne savons pas) sera implicitement stocké dans le réseau, et ceci grâce aux propriétés suivantes:

- *Systèmes Non linéaires*: Les réseaux de neurones peuvent approximer n'importe quelle fonction.
- *Le Parallélisme*: Selon la définition de Hecht-Nielsen un réseau de neurones, est l'interconnexion de plusieurs neurones, considérés comme éléments de base effectuant une relation non-linéaire et distribuée. Ceci augmente leurs robustesse.
- *Implantation "Hardware":* Les récents développements dans la technologie des circuits VLSI (Very Large Scale Integration), ont permis *l'implantation de différents algorithmes d'apprentissage* des réseaux de neurones. Les réseaux de neurones VLSI traitent les signaux avec des processeurs parallèles fabriqués avec des simples amplificateurs ainsi que de résistances, à la place des calculs effectués séquentiellement sur une machine de von Neuman. L'apprentissage se fait en changeant les valeurs de résistances entre les amplificateurs.
- *Apprentissage et généralisation*: Les réseaux de neurones peuvent être entraînés grâce à un certain nombre d'exemples. Un réseau de neurones, bien entraîné, peut généraliser, quand des données non-apprises lui sont présentées. En plus, les réseaux de neurones, peuvent être entraînés en temps réel.
- *Systèmes multivariables*: Les réseaux de neurones sont naturellement multivariables; ils sont directement applicables aux systèmes MIMO.

Ces propriétés rendent les réseaux de neurones souhaitables en identification et en commande des systèmes dynamiques.

Conclusion.

Nous avons présenté, dans la première partie de ce chapitre, des définitions permettant de se familiariser avec les réseaux de neurones, rappelons qu'un réseau de neurones n'est rien d'autre qu'un approximateur de fonction basé sur l'interconnexion de plusieurs entités élémentaires appelées neurones, les réseaux de neurones ont été subdivisés, selon leur architecture, en deux classes (statiques et dynamiques). Dans le seconde partie, on a parlé de l'apprentissage de réseaux de neurones, ce dernier est défini comme étant n'importe quelle changement sur n'importe quelle connexion, ce processus a été subdivisé en trois classes (supervisé, non-supervisé, "reinforcement learning"). Dans la troisième partie, on a pu voir, comment les réseaux de neurones pouvaient approximer n'importe quelle fonction, cependant les théorèmes sur lesquels cette propriété a été démontrée ne donnent aucune "recette". Dans la dernière partie, on a cité quelques caractéristiques qui rendent les réseaux de neurones souhaitables en identification et en commande des systèmes non-linéaires, c'est cette caractéristique qui sera mise en évidence dans les chapitres suivants.

Chapitre II

IDENTIFICATION PAR RESEAUX DE NEURONES

" J'admire la beauté de cette simplicité logique en laquelle je crois, faite d'ordre et d'harmonie que nous ne pouvons qu'appréhender qu'avec humilité et de façon seulement imparfaite."

Albert EINSTEIN (1879-1955)

CHAPITRE II
IDENTIFICATION PAR RESEAUX DE NEURONES

I. Introduction:

Le but ultime de la science est de comprendre le fonctionnement des processus rencontrés dans la vie courante. Une des méthodes qui permet de réaliser ce but est "la modélisation" . L'identification des systèmes est intéréssée par la construction de modèles mathématiques à partir des observations entrées-sorties du système.

Contrairement à l'identification des systèmes linéaires ou des méthodes existent [Ljung, 1987], l'identification des systèmes non-linéaires est une tache difficile, deux méthodes existent : 1) méthodes paramétriques , 2) méthodes non-paramétriques .

Les méthodes paramétriques supposent que le modèle du système possède une forme mathématique particulière exprimée en fonction d'un ensemble de paramètres inconnus, le problème consiste donc en l'identification de ces paramètres en s'aidant des données entrée-sortie.Ces modèles paramétriques sont rares en systèmes non-linéaires et souvent inutilisables, car ils requièrent un grand nombre de paramètres, on peut citer par exemple le modèle paramétrique basé sur les séries de Volterra [Isidori, 1989].

Les méthodes non-paramétriques n'assument aucune connaissance de la structure du système et ils sont basés sur la manière de déterminer la classe de modèles appropriés pour une application particulière.

Les réseaux de neurones avec leurs capacité d'approximer n'importe quelle fonctions peuvent etre utilisés comme modèles de base des systèmes non-linéaires, [Miller *et al*, 1990][Hunt *et al*, 1992].

Une importante question dans le contexte de l'identification des systèmes est l'identifiabilite du système, autrement dit, étant donnée une structure particulière au modèle , est-ce que le systèmes étudié peut être représenté convenablement par cette structure ?. Dans l'absence des résultats théoriques concrets pour l'identification par les réseaux neuronaux, nous supposerons que tous les systèmes étudiés dans ce cadre appartiennent a la classe des systèmes représentables par les réseaux choisis[Hunt *et al*, 1992] [Narendra and Parthasarathy, 1990, 1991].

Dans le présent chapitre, nous allons d'abord parler de l'identification directe, en utilisant les *réseaux de neurones généralisés* [Narendra and Parthasarathy, 1990], qui sont une combinaison entre des opérateurs linéaires et des réseaux de neurones; On parlera ensuite de l'identification inverse, puis de l'important lien qui existe entre la convergence

et l'excitation persistante, aprés on présentera des simulations pour valider la théorie, et enfin on présentera une conclusion.

II. Identification Directe :

La procédure d'entraînement des réseaux de neurones pour représenter les systèmes dynamiques va être référée par l'identification directe . Un structure permettant cette tache est schématisée dans la Fig.1; On remarque que le réseau de neurones est placé en parallèle avec le système, et c'est l'erreur entre la sortie du système et du réseau (l'erreur de prédiction) qui est utilisée pour l'apprentissage du réseau.

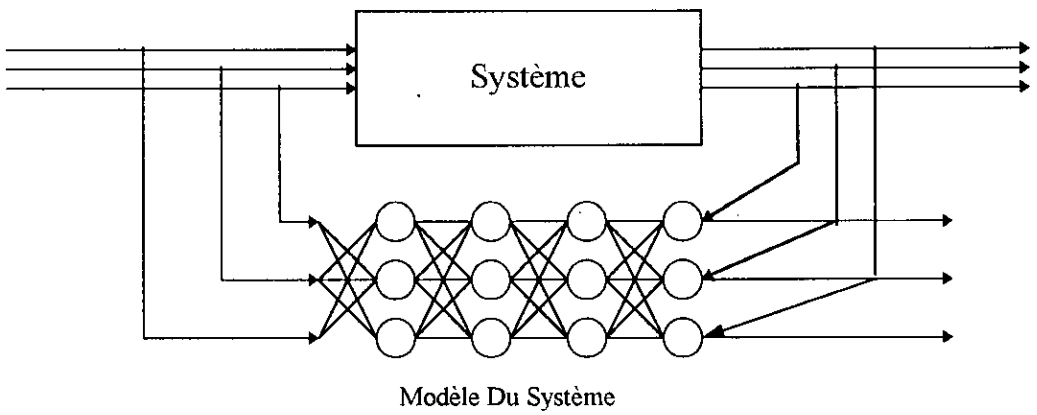


Fig. 1. Identification d'un système dynamique par Réseaux de Neurones

Le comportement dynamique des systèmes physiques étudies pose un problème pour l'intégration de cette dynamique dans les réseaux de neurones. Cette dynamique n'est pas un caractéristique intrinsèque des réseaux statiques. Ainsi, ces deniers peuvent échouer lorsqu'on veut capturer les principales caractéristiques du système. On peut résoudre ce problème soit en utilisant les réseaux récurrents, soit en introduisant un comportement dynamique dans les neurones[1] . Soit en augmentant le nombre d'entrées du réseau, en considérant les sorties et les entrées précédentes[Narendra and Parthasarathy, 1990].

[1] L'introduction d'une dynamique dans les neurones, est basée sur une analogie avec les systèmes biologiques. Il a été suggéré qu'un filtre passe bas d'ordre 1, pouvait représenter les caractéristiques dynamiques. L'introduction de ce filtre est relativement simple, la sortie du neurone sera régie par l'équation:

$$y^f(t) = \lambda y^f(t-1) + (1-\lambda)y(t), \quad 0 < \lambda < 1$$

cette méthode, permet de réduire considérablement les dimensions du réseaux. L'inconvénient de cette méthode est que le paramètre λ, ne peut être déterminé à priori, et il faudrait le déterminer en plus des poids du réseau.

Dans ce cadre, quatre modèles sont introduits pour la représentation des systèmes non-linéaires SISO avec la possibilité de généraliser aux systèmes MIMO sans aucun problème[Narendra and Parthasarathy, 1990].

Ces modèles ne sont en fait qu'une extension des modèles utilisés dans les systèmes linéaires aux systèmes non-linéaires.

En tenant compte de ceci, des modèles d'identification vont être proposes en utilisant des réseaux de neurones qui peuvent être entraînés par l'algorithme de la "backpropagation".

II-1) Caractérisation:

Les modèles des quatre classes des systèmes considérés ici peuvent être décrits par les équations aux différences non-linéaires suivantes :

Modèle I:

$$y_p(k+1) = \sum_{i=0}^{n-1} \alpha_i y_p(k-i) + g\left[u(k), u(k-1), \ldots, u(k-m+1)\right]. \qquad \text{(II-1)}$$

Modèle II:

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \ldots, y_p(k-n+1)\right] + \sum_{i=0}^{m-1} \beta_i u(k-i). \qquad \text{(II-2)}$$

Modèle III:

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \ldots, y_p(k-n+1)\right] + g\left[u(k), u(k-1), \ldots, u(k-m+1)\right] \qquad \text{(II-3)}$$

Modèle IV:

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \ldots, y_p(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)\right] \qquad \text{(II-4)}$$

Avec $\left[u(k), y_p(k)\right]$ représentant la paire entrée-sortie du système SISO à l'instant k, $m \leq n$. Les fonctions $f: \Re^n \rightarrow \Re$, dans les modèles II et III, ainsi que $f: \Re^{n+m} \rightarrow \Re$ dans le modèle IV, et $g: \Re^m \rightarrow \Re$ dans (II-3)ont supposées dérivables par rapport à leurs arguments.

On constate, que pour tous les modèles, la sortie du système a l'instant k+1 dépend a la fois de ses n valeurs passées y(k-i)(i=0,n-1), et des m valeurs passées de l'entrée u(k-j)(j=0,m-1). Dans la partie suivante, chaque modèle sera succinctement décrit.

Modèle I: Dans ce modèle, la sorties du système non-lineaire inconnu est assumée dépendante linéairement de ses valeurs précédentes et non linéairement des valeurs passées de l'entrée (fig.2). L'avantage de ce modèle est qu'il permet de discuter la stabilité en régime libre des systèmes non linéaires.

Modèle II: La sortie du système non-lineaire inconnu dans ce cas est assumée dépendante linéairement de l'entrée u(k) et de ses valeurs passées, et non-lineairement de ses propre anciennes valeurs (fig. 3). L'avantage de ce modèle est qu'il est directement

applicable dans la commande par réseaux de neurones comme on va le montrer dans le chapitre suivant.

Modèle III: Dans ce cas, la sortie du système non-lineaire inconnu dépend non-lineairement a la fois de ses valeurs passées et des valeurs passées de l'entrée. Cependant les effets de l'entrée et de la sortie sont additionnes. La représentation de ce modèle est schématisée par la figure 4 .

Modèle IV: Ce modèle est le plus général, puisqu'il englobe les précédents modèles. La sortie à n'importe quel instant dans ce cas est une fonction non-lineaire des valeurs passées de l'entrée et de la sortie a la fois. La représentation de ce modèle en utilisant des lignes de retards est schématisée par la figure 5. Ce modèle est analytiquement, moins maniable que les autres modèles.

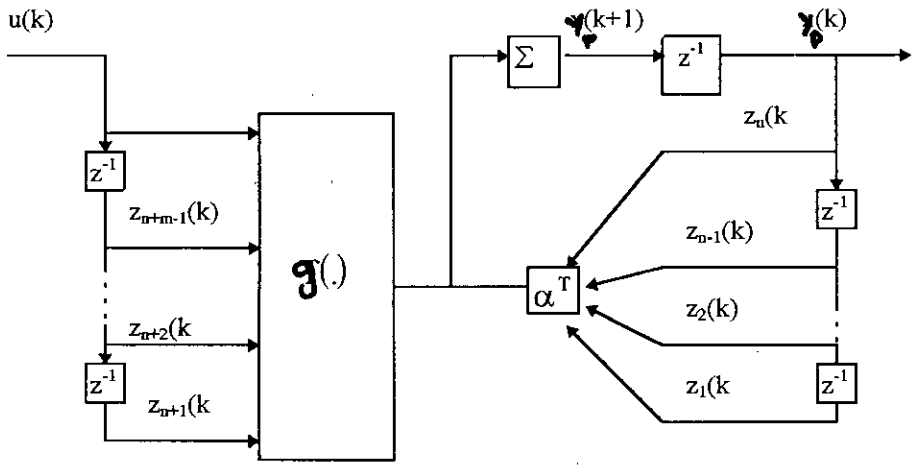


Figure 2: Structure du Modèle I.

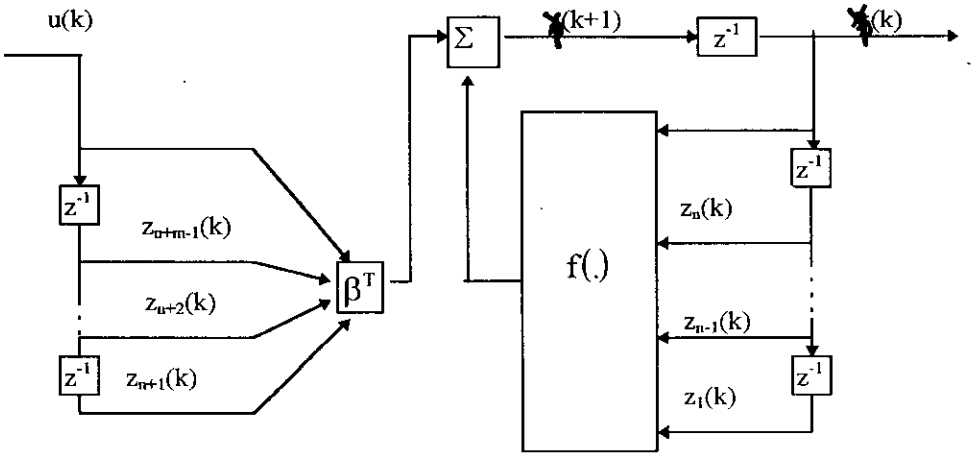


Figure 3: Structure du Modèle II.

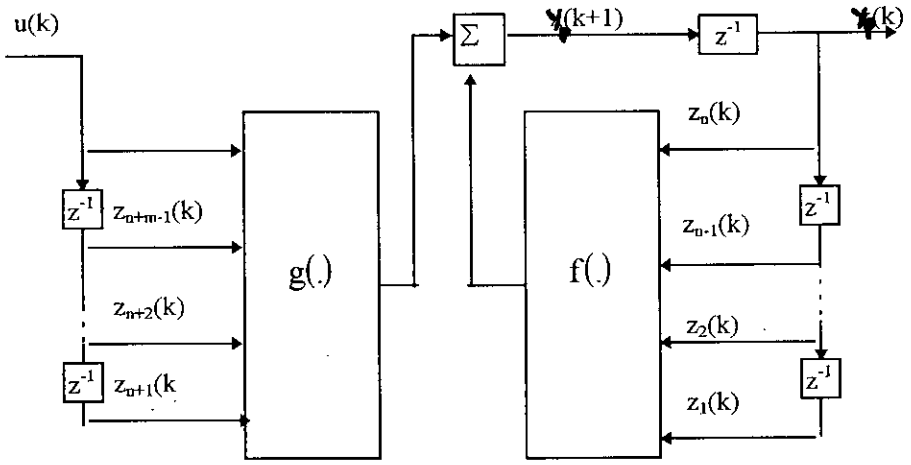


Figure 4: Structure du modèle III.

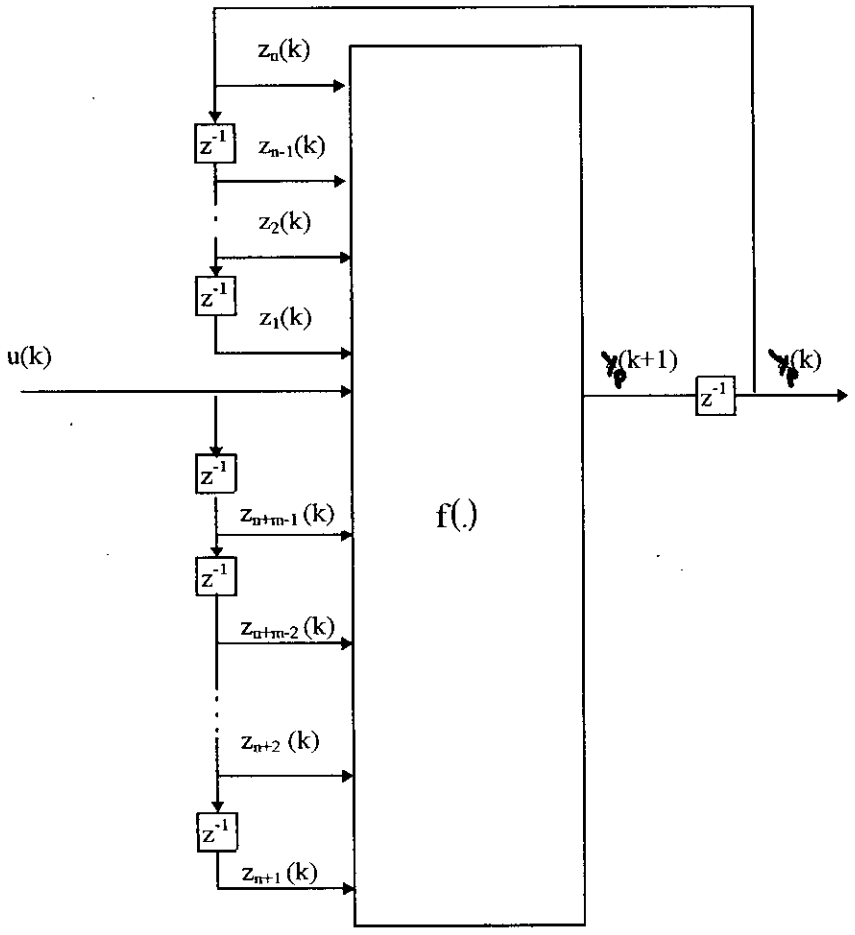


Figure 5: Structure du modèle IV.

II-2) Procédure d'Identification :

Le modèle d'identification du système est compose de réseaux de neurones et des lignes de retards.

Dans chaque cas, le réseau de neurones est supposé contenir suffisament de paramètres (poids), afin d'être apte à représenter exactement la caractéristique entrée-sortie du système non-lineaire correspondant. Dans ce cas, la fonction non-lineaire dans l'équation aux différences décrivant le système peut être remplacée par un réseau de neurones avec des poids fixes mais inconnus. En plus on doit supposer qu'une solution théorique pour le problème de l'identification est supposée existante.

Pour identifier le système, un modèle d'identification est choisit basé sur des informations à priori concernant le système. Si y(k+1) et $\hat{y}(k+1)$ sont respectivement la sortie à l'instant k+1 du système et du modèle . L'erreur $e(k+1) \stackrel{\Delta}{=} \hat{y}(k+1) - y(k+1)$, est utilisée pour ajuster les poids du réseau de neurones par l'algorithme de la "backpropagation", et ceci dépend de la structure choisie.

a. Modèle Parallèle :

La méthode d'identification par le modèle parallèle est basée sur l'utilisation des sortie du modèle lui même pour l'entraînement du réseaux de neurones (fig. 6.).

Pour identifier un système non-lineaire de la quatrième classe avec cette approche, le modèle d'identification sera le suivant :

$$\hat{y}(k+1) = \eta\left[\hat{y}(k), \hat{y}(k-1), \cdots, \hat{y}(k-n+1); u(k), u(k-1), \cdots, u(k-m+1)\right]. \qquad (II\text{-}5)$$

L'ajustement des poids se fait à l'aide de l'algorithme RecN-pattern (cet algorithme est le même que FFN-pattern, la seule différence est l'utilisation des sorties du réseau à la place de la sortie du système) [Qin *el al*, 1992].

b. Modèle Série-Parallèle :

Contrairement au modèle parallèle, le modèle série-parallèle choisit une structure entrée-sortie pour le modèle d'identification qui est la même que celle du système.

Cependant, il n'y a pas de retour (feed-back) autour du réseau (fig. 6). C'est cette propriété qui nous permettra d'ajuster les poids à l'aide de la "backpropagation' classique (l'algorithme FFN-pattern peut être utilisé dans ce cas).

Dans le cas d'un système non-linéaire appartenant à la quatrième classe, le modèle d'identification sera le suivant :

$$\hat{y}(k+1) = \eta\left[y(k), y(k-1), \cdots, y(k-n+1); u(k), u(k-1), \cdots, u(k-m+1)\right]. \qquad (II\text{-}6)$$

En assumant que l'erreur de sortie tende vers une valeur négligeable, le modèle série parallèle peut être remplace par le modèle parallèle sans problèmes majeurs. Ainsi, l'apprentissage sera effectué en se basant sur le modèle série-parallèle, et le test sera effectué en utilisant le modèle parallèle.

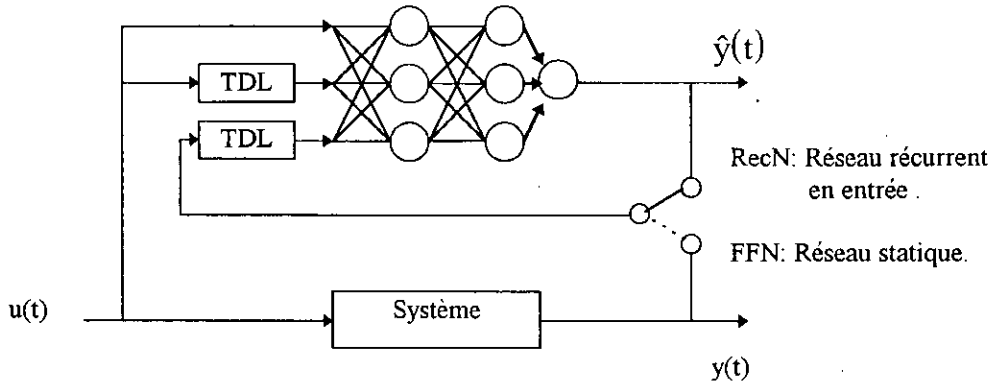


Fig.6 : Schéma de l'identification en utilisant les modèles parallèle ou série parallèle (TDL: Tapped Delay Line.) .

III. Identification Inverse :

L'utilisation des réseaux de neurones pour l'identification de la dynamique inverse des systèmes non-linéaires présente une grande promesse pour la recherche, car les modèles inverses des systèmes dynamiques ont une utilité immédiate dans la commande, ceci sera plus détaillé dans le troisième chapitre .

Le problème majeur avec l'identification inverse arrive quand plusieurs entrées produisent la même sortie, autrement dit, quand l'inverse du système n'est pas bien défini; dans ce cas le réseau de neurones tend à faire correspondre les mêmes entrées à différentes sorties.

Dans ce qui suit nous allons considérer deux méthodes qui permettent de déterminer un réseau de neurones qui approxime la fonction inverse du système.

Dans le cas d'un système décrit par le modèle IV, la fonction f^{-1} conduisant à la génération de l'entrée u(k) exige la connaissance de la valeur future de y(k+1). Pour surmonter ce problème on remplace cette valeur future par la valeur de $y_d(k+1)$ qui est assumée disponible à l'instant k, car $y_d(i)$, $i \in N$, est liée au signal de référence ou

d'entraînement. Ainsi, La représentation entrée-sortie du réseau modélisant l'inverse du système est décrite par:

$$u(k) = \eta[y(k),\cdots,y(k-n+1),r(k+1);u(k-1),\cdots,u(k-m+1)], \qquad \text{(II-7)}$$

dans ce cas le réseau η approxime f^{-1}, ainsi, le modèle inverse reçoit en entrée les sorties actuelles et passées du système, le signal de référence et les valeurs passées de l'entrée du système.

Plusieurs approches existent pour identifier l'inverse des systèmes dynamiques non-linéaires par les réseaux de neurones, et seront discutées dans le prochain chapitre. La figure 7, montre le schéma global de l'identification inverse .

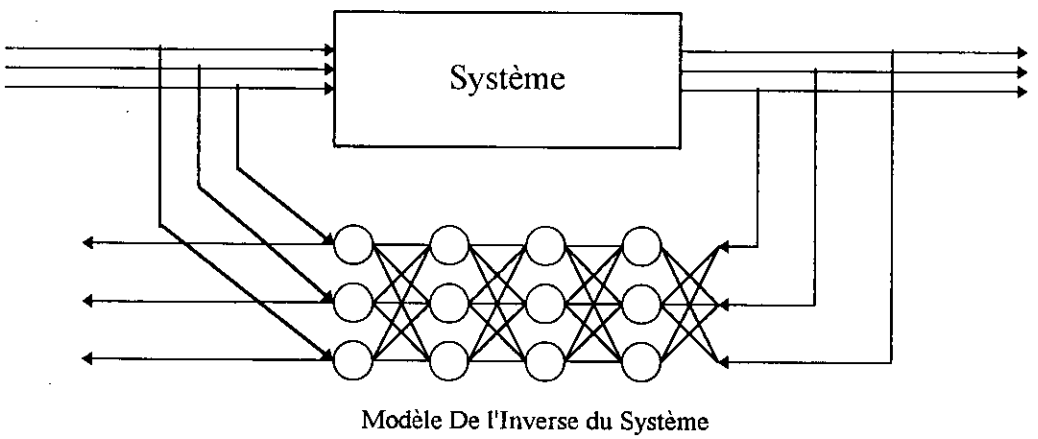


Fig.7: Identification Inverse d'un Système Dynamique par Réseaux de Neurones

IV. Convergence et Excitation Persistante :

Comme en identification classique ou le signal d'entraînement des systèmes adaptatifs joue un rôle important; Le signal d'entraînement des réseaux de neurones est très important pour que le modèle d'identification soit proche du système réel.

L'ensemble des données d'entraînement doit représenter toutes les classes des entrées auxquelles le système peut être soumis. Ceci permettra au système de répondre d'une façon satisfaisante lorsqu'il sera excité par un signal non-appris. Ainsi le signal d'excitation doit être tel qu'il puisse *faire sortir toute la dynamique du système* .La recherche du signal possédant cette propriété est le problème de l'*excitation persistante* [Narendra and Annaswamy , 1989].

Une méthode en boucle ouverte est basée sur l'utilisation d'un signal aléatoire uniformément distribué à travers tout l'espace de travail *W*. Dans le cas de systèmes bouclés ceci signifie l'introduction d'un signal auxiliaire , ceci entraîne une certaine dégradation des performances du système. Cette approche suppose que le système étudié est totalement inconnu; en réalité on possède une certaine connaissance du système.

V. SIMULATIONS :

Dans cette partie, on va présenter les résultats de l'identification des quatre modèles précédents, en *comparant entre deux différentes architectures*, la première consiste en un réseau dont l'architecture est la même que celle du modèle, la seconde architecture est celle d'un réseau multicouches dont l'architecture est la même pour tous les quatre modèles, le réseau appartient au quatrième modèle. En plus, on va étudier l'*influence du nombre d'entrées*, car généralement on n'a pas une estimation du décalage qui existe. Ensuite on identifiera un système dynamique par un réseau dynamique; On proposera, ensuite une architecture qui permettrait d'étudier la *stabilité en régime libre des système non-linéaires*. Et enfin, on fera une comparaison entre les réseaux statiques, entraînés par les méthodes FFN-pattern et FFN-Batch, et les réseaux récurrents en entrée entraînés par l'algorithme RecN-pattern.

Pour une discussion plus simple, on notera une classe de fonctions générée par un réseau de neurones contenant N couches par le symbole $\eta^{N}_{i_1,i_2\ldots i_{N+1}}$.Tel que le réseau contienne i_1 entrées, i_{N+1}sorties, et (N-1) champs de neurones intermédiaires, chaque champ de neurones contenant respectivement $i_2, i_3, \ldots, i_N$ neurones.

Exemple 1:

Le système qu'on veut identifier possède l'équation aux différences suivante:

$$y_p(k+1) = 0.3y_p(k) + 0.6y_p(k-1) + f[u(k)]. \qquad \text{(II-8)}$$

La fonction inconnue possède la forme $f(u) = u^3 + 0.3u^2 - 0.4u$.

A partir de l'équation (II-8), il est clair que le système libre, est asymptotiquement stable ainsi a une entrée bornée correspond une sortie bornée. Pour identifier ce système on utilisera deux modèles série-parallèle.
Le premier est régit par l'équation aux différences :

$$\hat{y}_p(k+1) = 0.3y_p(k) + 0.6y_p(k-1) + N[u(k)] \qquad \text{(II-9)}$$

Le second par:

$$\hat{y}_p(k+1) = N\left[u(k), y_p(k), y_p(k-1)\right]. \qquad \text{(II-10)}$$

Les poids du réseau ont été ajustés a chaque instant, en utilisant la backpropagation statique. Les deux réseaux appartiennent a la classe $\eta^4_{1,20,10,1,1}$,et c'est la méthode FFN-pattern qui a été utilisée pour l'apprentissage, avec un pas de 0.3 pour la première couche, 0.25 pour la seconde, 0.2 pour la troisième, et 0.15 pour la dernière un terme momentum égal a 0.2 pour le premier réseau. Pour le second on a utilisé un pas de 0.2 pour la première couche, 0.15 pour la seconde, 0.1 pour la troisième, 0.08 pour la dernière, et un terme momentum égal a 0.2. L'entrée du système était une entrée aléatoire dont l'amplitude était uniformément distribuée dans l'intervalle [-1,1],pour le test on a utilisé une entrée sinusoïdale $u(k) = \sin\left(2\pi k/250\right)$ pour $k \le 250$ et $u(k) = \sin\left(2\pi k/250\right) + \sin\left(2\pi k/25\right)$ pour $k \ge 250$.Les résultats sont sur la figure 8-a ,qui représente le premier réseau, et la figure 8-b, qui représente le second réseau. On remarque que l'erreur est si petite qu'on ne peut distinguer entre le modèle et le système. L'adaptation s'est faite durant 22000 itérations pour le premier réseau, et de 35000 pour le second. Ainsi, le second réseau approxime de la même façon que le premier. Ces résultats ne font que confirmer la capacité des réseaux de neurones à approximer n'importe quelle fonction.

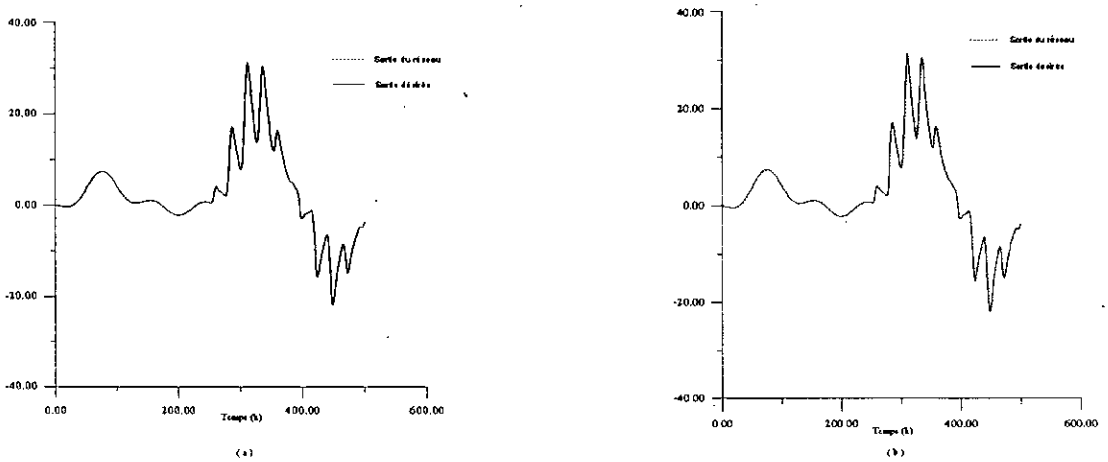


Figure 8 : Sorties du système et du modèle.(a):premier réseau.(b):second réseau.

Exemple 2:

Le système qu'on veut identifier est gouverné par l'équation aux différences suivante:

$$y_p(k+1) = f\left[y_p(k), y_p(k-1)\right] + u(k) \qquad \text{(II-11)}$$

avec:

$$f\left[y_p(k), y_p(k-1)\right] = \frac{y_p(k)y_p(k-1)\left[y_p(k)+2.5\right]}{1+{y_p}^2(k)+{y_p}^2(k-1)} \qquad \text{(II-12)}$$

Ceci correspond au second modèle. Deux modèles série-parallèle sont utilisés, le premier correspond à l'équation:

$$\hat{y}_p(k+1) = N\left[y_p(k), y_p(k-1)\right] + u(k), \qquad \text{(II-13)}$$

Le second correspond à :

$$\hat{y}_p(k+1) = N\left[u(k), y_p(k), y_p(k-1)\right], \qquad \text{(II-14)}$$

ou N est un réseau de neurones qui appartient a la classe $\eta^4_{1,20,10,1,1}$.L'entrée u(k) correspond a une entrée aléatoire uniformément répartie dans l'intervalle [-2,2],et des pas de 0.3,0.25,0.2,0.15 pour la première, seconde, troisième, et dernière couche, réspectivement, et un terme momentum égal a 0.2 pour le premier réseau et 0.4 pour le second (un momentum égal a 1 présente une très mauvaise approximation).L'adaptation des poids s'est faite avec un pas de cinq en utilisant le gradient $\sum_{i=k-4}^{k} e^2(i)$.La figure 9 représente les résultats après un adaptation de 72000 itérations pour le premier réseau et 80000 pour le second.

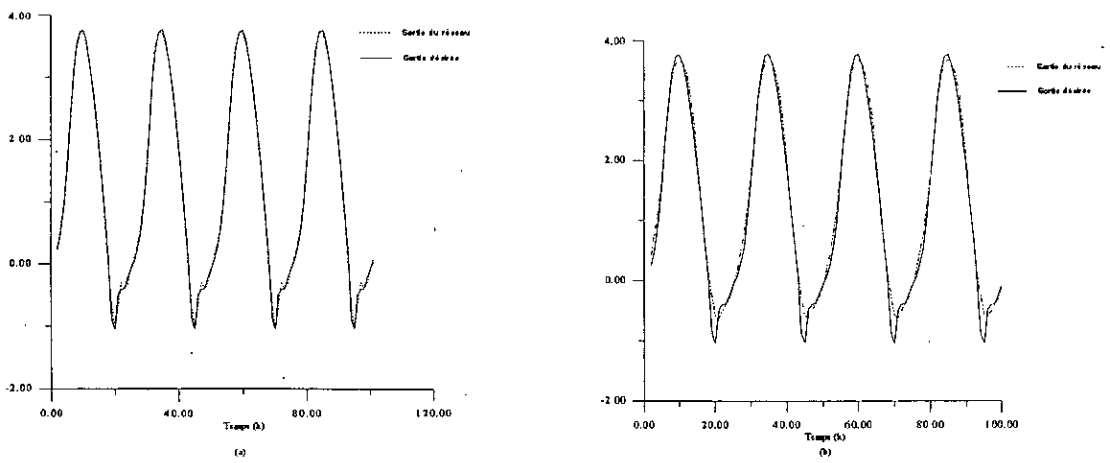


Figure 9 : Sorties du modèle et du système .(a):premier réseau.(b):second réseau.

Dans ce cas le second réseau réussit moins que le premier dans l'approximation.

Maintenant nous allons utiliser le même type de modèles, mais cette fois en changeant le nombre des entrées.En utilisant dans le premier cas $y_p(k-2)$,et dans le second en ne considérant que $y_p(k)$.

Dans le premier cas le modèle aura la forme :

$$\hat{y}_p(k+1) = N_1\left[y_p(k), y_p(k-1), y_p(k-2)\right] + u(k) \qquad \text{(II-13)}$$

Dans le second:

$$\hat{y}_p(k+1) = N_2\left[y_p(k)\right] + u(k). \qquad \text{(II-14)}$$

Le réseau N_1 appartient a la classe $\eta^4_{3,20,10,1,1}$ il converge après 70000 itérations(les mêmes paramètres sont utilisés que précédemment).Le réseau N_2 appartient a la classe $\eta^4_{1,20,10,1,1}$.La figure 10 montre les résultats ,quand l'entrée est $u(k) = \sin\left(3\pi k/10\right)$.

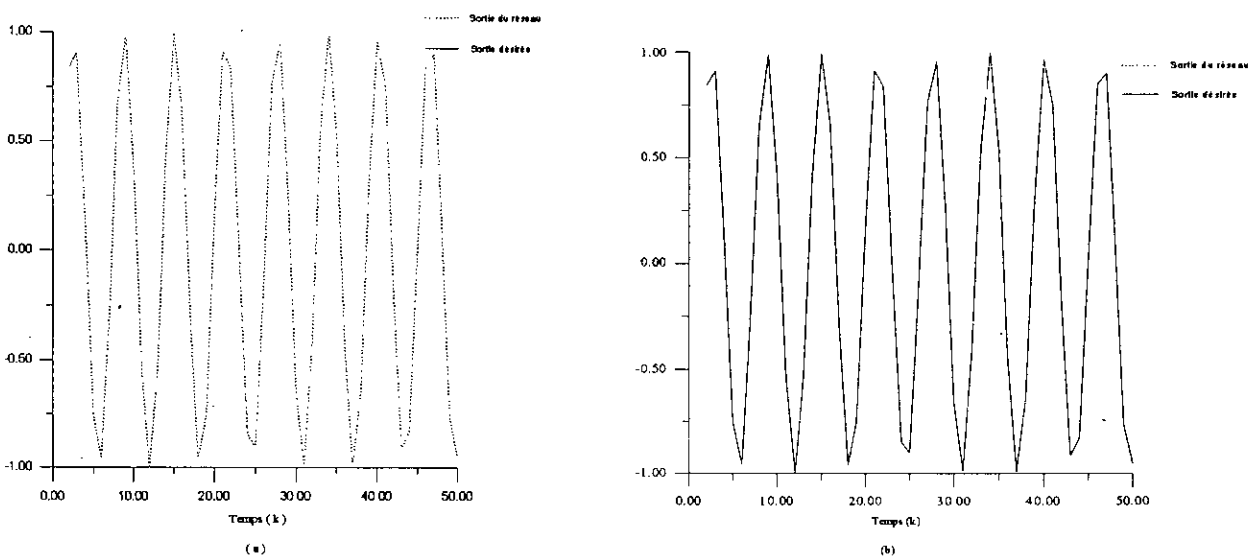


Figure 10: Etude de l'effet du nombre d'entrées.(a) réseau N_1, (b) réseau N_2

On remarque que les sorties du modèle et du système sont indiscernables; ainsi le réseaux n'est pas sérieusement affecté par le nombre d'entrées.

Exemple 3:
Dans ce cas le modèle est celui du modèle 3. Il est décrit par l'équation aux différences:

$$y_p(k+1) = \frac{y_p(k)}{1+y_p(k)^2} + u^3(k). \qquad \text{(II-15)}$$

Qui correspond a $f[y_p(k)] = y_p(k)/((1+y_p(k)^2)$ et $g[u(k)] = u^3(k)$.
Deux modèles série-parallèle ont été utilisés. Le premier consiste en deux réseaux de neurones N_f et N_g qui appartiennent à la classe $\eta^4_{1,20,10,1,1}$, ce système sera donc décrit par l'équation aux différences:

$$\hat{y}_p(k+1) = N_f[y_p(k)] + N_g[u(k)]. \qquad \text{(II-16)}$$

Les deux réseaux N_f et N_g sont en fait utilisés pour déterminer les estimées $\hat{f}$ et $\hat{g}$ de f et g
Le second consiste en un seul réseau qui appartient a la classe $\eta^4_{2,20,10,1,1}$, il peut être décrit par l'équation aux différences $\hat{y}_p(k+1) = N(y_p(k), u(k))$.
Les poids des deux réseaux ont été ajustés a chaque instant pendant 50000 itérations , avec un pas de 0.3 pour la première couche, 0.25 pour la seconde, 0.2 pour la troisième et 0.15 pour la dernière. Le momentum étant égal a 0.4.
Pour le premier réseau, l'entrée utilisée était une entrée aléatoire dont l'amplitude est uniformément distribuée dans l'intervalle [-2,2] dans le réseau N_f et [-10,10] dans N_g Les figures 11-a et 11-b montrent les estimées des fonctions f et g. Tandis que la figure 12 montre les sorties du modèle et du système quand l'entrée est $u(k) = \sin(2\pi k/25) + \sin(2\pi k/10)$. La figure 13 montre les sorties du modèle et du système dans le cas du second réseau qui a été entraîné avec une entrée aléatoire, dont l'amplitude est uniformément répartie dans l'intervalle [-2,2].

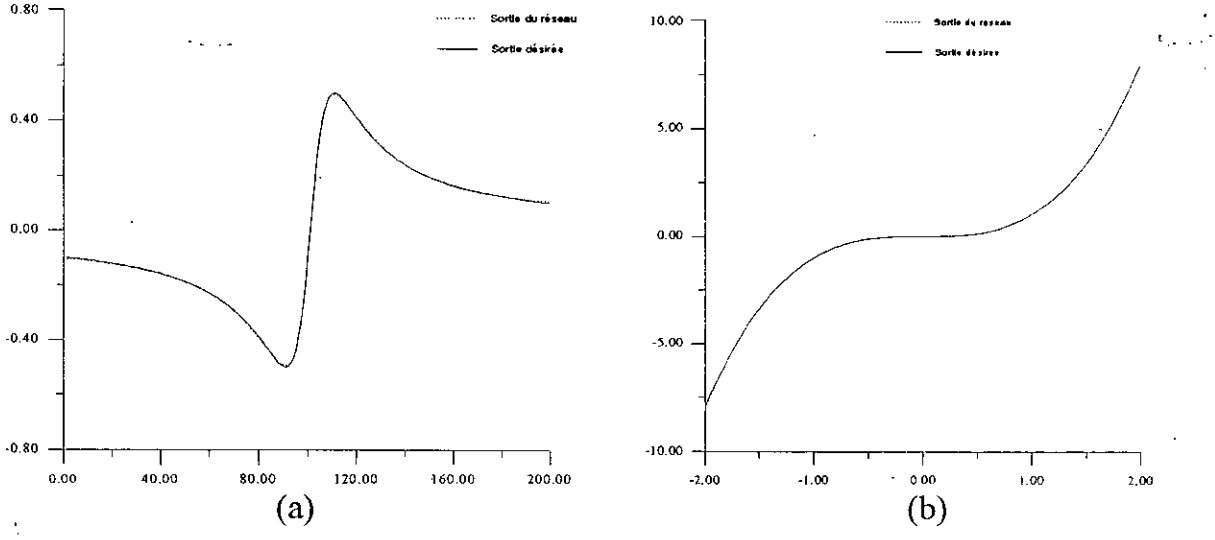


Figure 11 : Approximation des fonctions : a) $f(y) = y/(1+y^2)$; b) $g(u) = u^3$

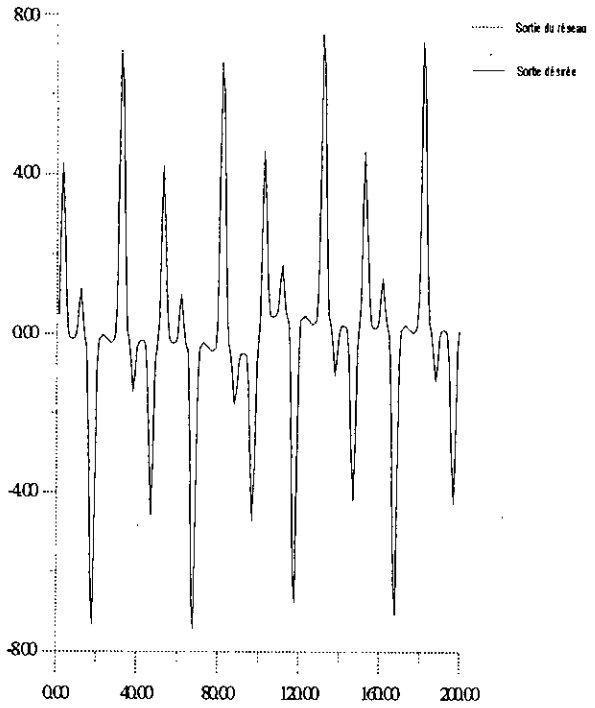


Figure 12: Cas du premier réseau

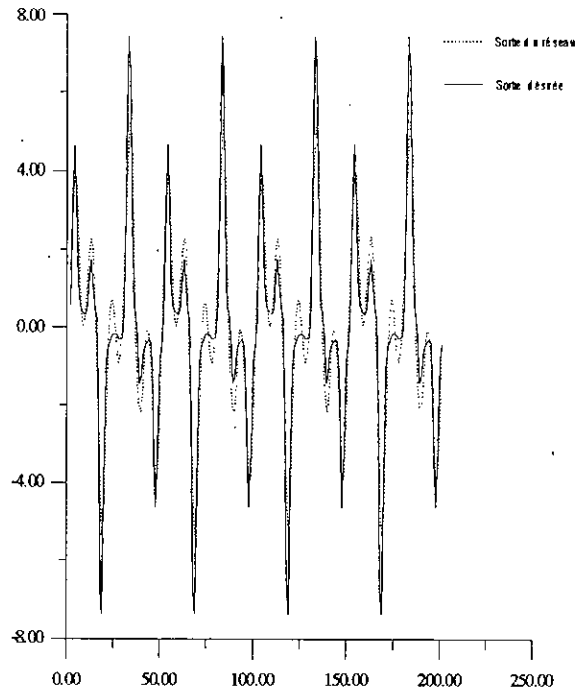


Figure 13 : Cas du second réseau

On remarque que le second réseau n'approxime pas très bien ce système. Contrairement au premier ou les sorties étaient indiscernables.

Exemple 4:

Les mêmes méthodes peuvent être utilisées quand le système est celui du modèle 4. Dans cet exemple on va identifier un système qui appartient au quatrième modèle à l'aide des modèles 1,2 et 4. On expliquera ensuite l'importance de l'identification de ce système par le modèle 1.

Le système est décrit par l'équation aux différences :

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)\right]. \qquad \text{(II-17)}$$

Avec:

$$f\left[x_1, x_2, x_3, x_4, x_5\right] = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_3^2 + x_2^2}. \qquad \text{(II-18)}$$

Dans un premier temps on identifiera ce système par le modèle IV, dans ce cas le réseau appartient a la classe $\eta^4_{5,20,10,1,1}$.La figure 14-a montre les sorties du modèle et du

système lorsque l'adaptation a été effectuée pendant 60000 itérations en utilisant un signal aléatoire uniformément distribué dans l'intervalle [-1,1],avec des pas de 0.3,0.25,0.15,0.1 pour les couches 1, 2, 3, 4, le momentum étant égal a 0.4. Comme indiqué précédemment l'identification se fait par le modèle série-parallèle et la test par le modèle parallèle. Dans ce cas le signal du test est :

$$u(k)=\begin{cases}\sin\left(2\pi k/250\right) & \text{si} \quad k\le 250\\ 0.8\sin\left(2\pi k/250\right)+0.2\sin\left(2\pi k/25\right) & \text{si} \quad k>500\end{cases} \qquad \text{(II-19)}$$

Maintenant utilisons le modèle II, ce modèle est particulièrement souhaitable en commande. Le modèle possédera l'équation au différences:

$$\hat{y}_p(k+1)=N\left[y_p(k),y_p(k-1),y_p(k-2)\right]+\alpha_0 u(k)+\alpha_1 u(k-1). \qquad \text{(II-20)}$$

N est un réseau de neurones qui appartient a la classe précédente. Les coefficients α_0 et α_1 sont inconnus et sont déterminés après fin d'adaptation qui durera 80000, la figure 14-b montre les résultats de cette adaptation ,on remarque que l'erreur entre le modèle et le système diminue.

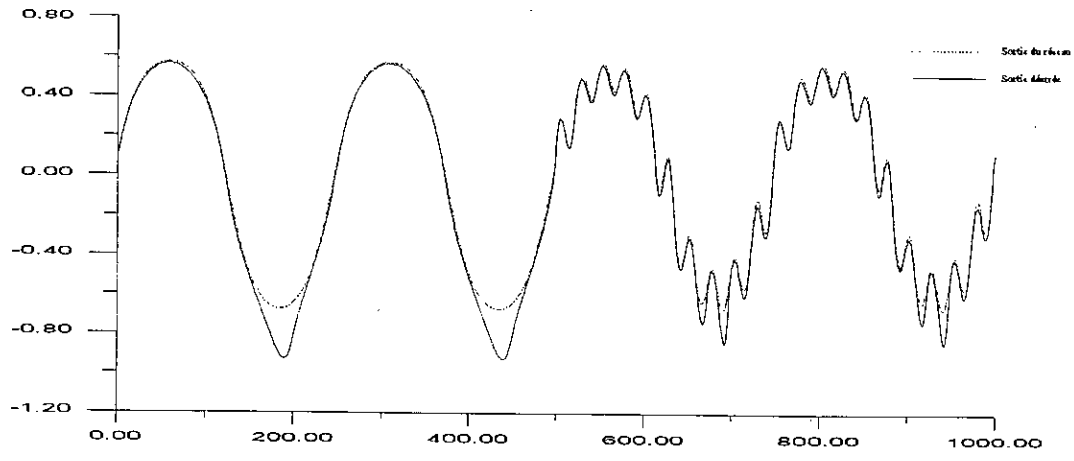
Dans le cas du modèle I, qui est important, car il permet d'étudier la stabilité en régime libre des systèmes non-linéaires. Le modèle possédera l'équation aux différences.

$$\hat{y}_p(k+1)=\alpha_0 y_p(k)+\alpha_1 y_p(k-1)+\alpha_2 y_p(k-2)+N\left[u(k),u(k-1)\right] \qquad \text{(II-21)}$$

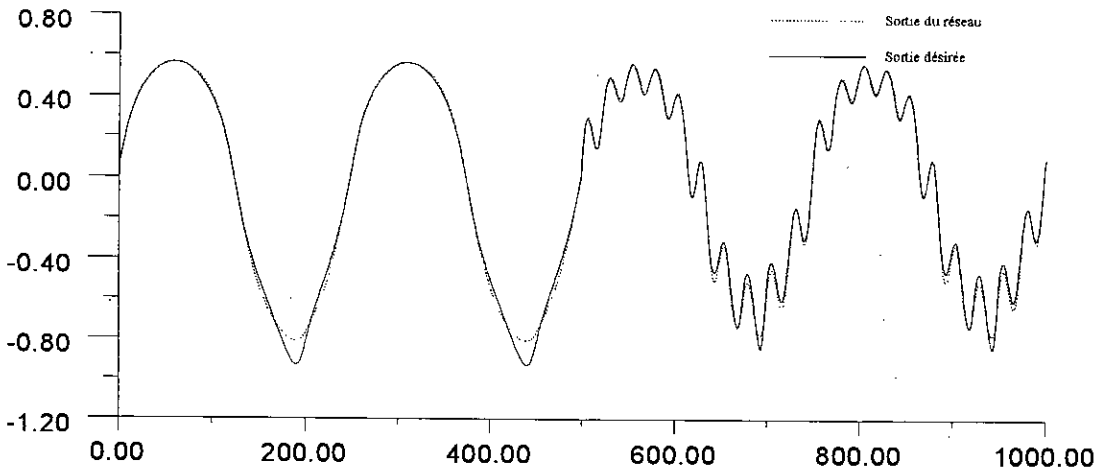
Nous allons étudier l'influence de N sur les coefficients α_i,i=1,3.

Si N appartient a la classe $\eta^4_{5,20,10,1,1}$,après 84000 itérations on trouve $\alpha_0=-0.041263,\alpha_1=0.042922$ et $\alpha_2=0.802252$.

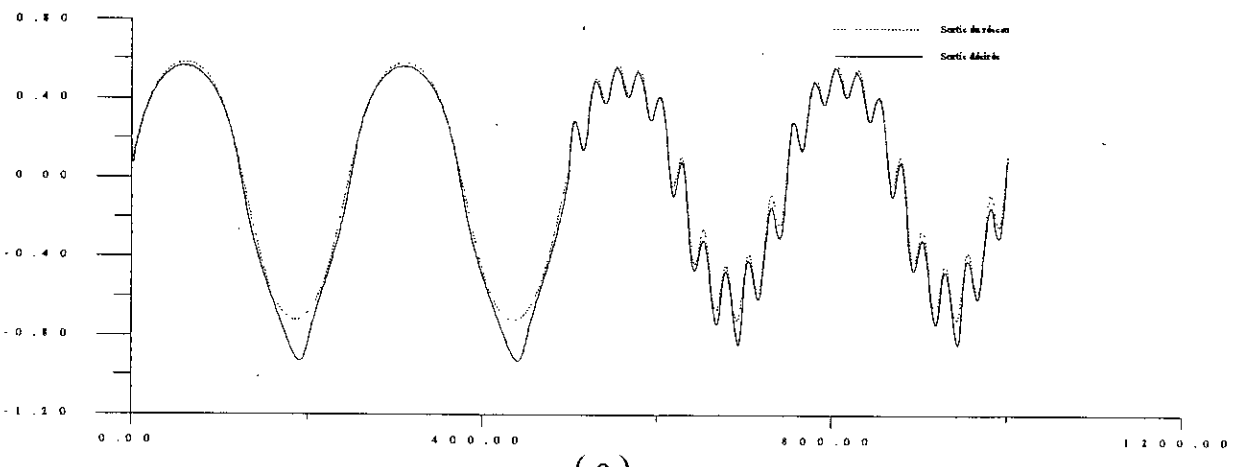
Si N appartient a la classe $\eta^4_{5,20,20,1,1}$,on trouve après 84000 itérations $\alpha_0=-0.048959$, $\alpha_1=0.042407$ et $\alpha_2=0.800385$, les coefficients sont à peu prés les mêmes. La figure 14-c montre les résultats de l'identification dans le premier cas, le second cas présente la "même" réponse.



(a)



(b)



(c)

Figure 14 : (a)Identification par le modèle IV. (b)Identification par le modèle II (c)Identification par le modèle I.

Exemple 5:

Dans cet exemple on montrera que les mêmes méthodes appliquées aux systèmes SISO peuvent être généralisées aux systèmes MIMO.

Considérons le système décrit par les équations:

$$\begin{bmatrix} y_{p1}(k+1) \\ y_{p2}(k+1) \end{bmatrix} = \begin{bmatrix} \dfrac{y_{p1}(k)}{1+y_{p2}(k)} \\ \dfrac{y_{p1}(k)y_{p2}(k)}{1+y^2_{p2}(k)} \end{bmatrix} + \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \qquad \text{(II-22)}$$

Ceci correspond a la version multivariable du modèle II.

Le modèle série-parallèle consiste en deux réseaux de neurones N^1 et N^2,il peut être décrit par l'équation:

$$\begin{bmatrix} \hat{y}_{p1}(k+1) \\ \hat{y}_{p2}(k+1) \end{bmatrix} = \begin{bmatrix} N^1\left[y_{p1}(k), y_{p2}(k)\right] \\ N^2\left[y_{p1}(k), y_{p2}(k)\right] \end{bmatrix} + \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \qquad \text{(II-23)}$$

L'adaptation des poids s'est faite pendant 80000 itérations avec des pas de 0.3,0.25,0.2 et 0.15 pour chaque couche correspondante, et deux entrées $u_1(k)$ et $u_2(k)$ uniformément réparties dans l'intervalle [-1,1].Les réponses du modèle et du système au vecteur d'entrées $\left[\sin\left(2\pi k/25\right), \cos\left(2\pi k/25\right)\right]^T$ sont montrées dans la figure 15.

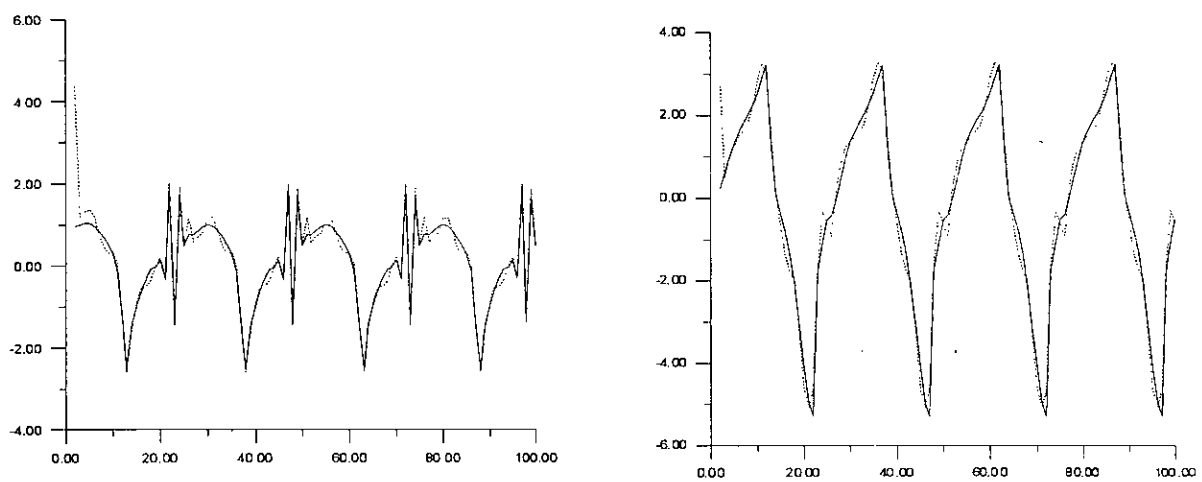


Figure 15 : Identification d'un système MIMO; a) y_1 et $\hat{y}_1$, b) y_2 et $\hat{y}_2$

Exemple 6 :

Dans cet exemple, on va identifier le système :

$$y(k+1) = 1.1\sin\left(\cos\left(y(k)\right)\right) + 1.5u(k) \qquad \text{(II-24)}$$

Les poids du réseau ont été ajustés en utilisant l'algorithme " Fixed point Learning" qui minimise l'erreur instantanée. Le réseau utilisé possède 2 entrées, 2 neurones dans la couche cachée et une sortie, les pas d'apprentissage utilisés étaient $\mu_1 = 0.01, \mu_2 = 0.02, \mu_3 = 0.01$; Notons que le choix des poids initiaux et du signal d'excitation est important, car on ne risque pas d'avoir un mauvais apprentissage, mais un réseau instable. Pour cela, on a choisi des poids initiaux $W = I, B = I, h = \left[0.2, 0.2\right]^T$, et signal aléatoire uniformément réparti entre -1 et 1. La figure 16 montre les résultats de l'identification, après seulement 2 cycles, chacun correspondant à un passage sur 100 exemples, l'adaptation s'est faite pendant 2000 itérations.

Le signal de test est défini par:

$$u(k) = \begin{cases} \sin\left(2\pi k/250\right) & \text{si} \quad k \le 50 \\ 0.8\sin\left(2\pi k/250\right) + 0.2\sin\left(2\pi k/25\right) & \text{si} \quad k > 50 \end{cases} \qquad \text{(II-25)}$$

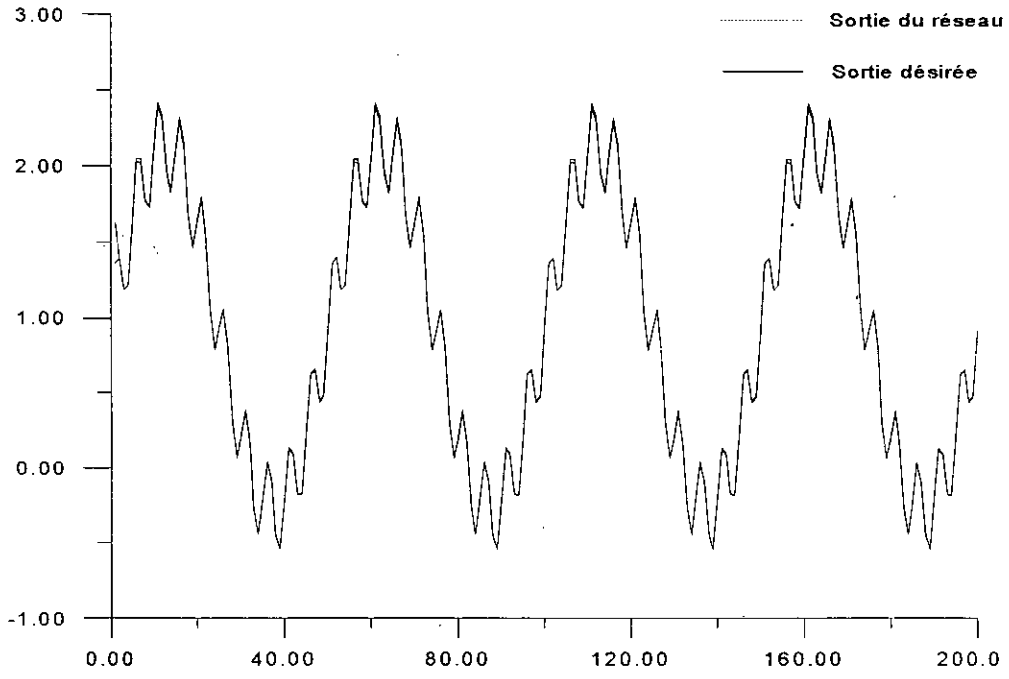


Figure 16 : Sorties du modèle et du réseau.

On remarque que les sorties sont indiscernables. En plus du nombre de neurones qui est faible, l'apprentissage s'est fait rapidement par rapport à un réseau statique appartient à la classe $\eta_{2,20,10,1}$ qui pour les mêmes pérformances a requis 50000 itérations.

Exemple 7 :

Dans cette partie, on fera une comparaison entre les algorithmes FFN-pattern, FFN-Batch, RecN-pattern, dans le cadre de l'identification du système :

$$y(k+1) = \frac{-0.9\,y(k) + u(k)}{1 + y(k)^2} \qquad \text{(II-26)}$$

Le réseau utilisé appartient à la classe $\eta_{2,20,15,1}$. L'adaptation s'est faite pendant 20000 itérations pour FFN-pattern, FFN-Batch et RecN-pattern, en utilisant une entrée aléatoire uniformément répartie entre -1 et 1.

Les figures 18, 19, et 20, montrent les résultats du test en utilisant un signal défini par :

$$u(k) = 0.2\left[\sin(0.5\,r_k) + \cos(2\,r_k) + \sin^2(3\,r_k) + \cos^3(4\,r_k) + \cos(5\,r_k) + \sin(20\,r_k)\right], \qquad \text{(II-27)}$$

$r_k = \pi/40 \cdot k$. dans le cas de l'apprentissage en utilisant, réspectivement, les algorithmes, FFN-pattern, FFN-Batch et RecN-pattern.

On a ensuite testé la robustesse des réseaux statiques entraînés par FFN-pattern et de réseaux récurrent en entrée entraînés par RecN-pattern, en introduisant des bruits de mesure, qui sont des bruits gaussiens, de différentes amplitudes et ceci selon le schéma de la figure 17. Le tableau 1, indique la valeur de l'erreur $e = \frac{1}{T}\sum_{t=1}^{T}\left(\hat{y}(t) - y(t)\right)^2$, avec T = 150, est le nombre d'exemples.. Les mêmes réseaux ont été utilisés, et l'apprentissage à été éffectué durant 12000 itérations dans le cas FFN avec $A_m = 0.1$, pendant 10000 itérations avec $A_m = 0.2$ et pendant 11000 itérations quand $A_m = 0.5$. Dans le cas RecN, l'adaptation s'est faite pendant 13000 itérations quand $A_m = 0.1$, 12000 itérations quand $A_m = 0.2$, et 13000 itérations quand $A_m = 0.5$.

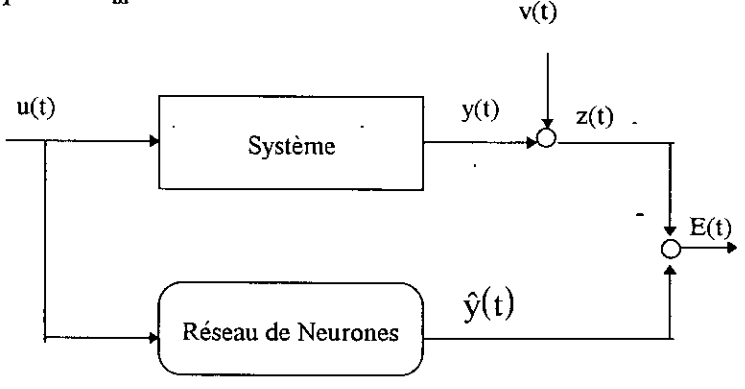


Fig. 17: Identification d'un système dynamique, dont les sorties sont entachées par des bruits de mesure s .

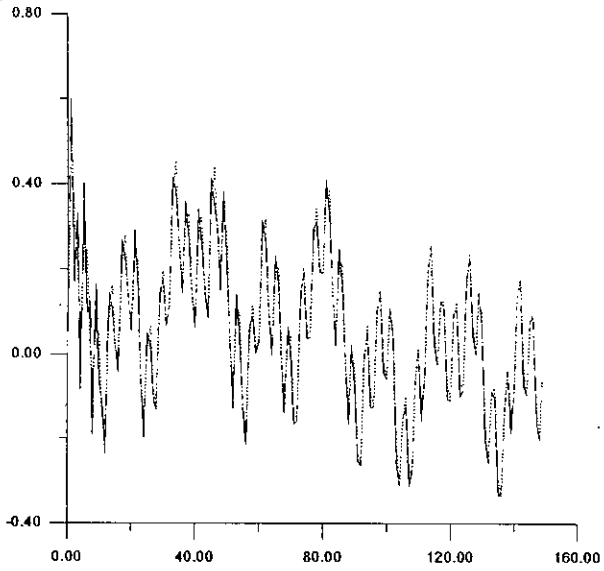


Figure 18

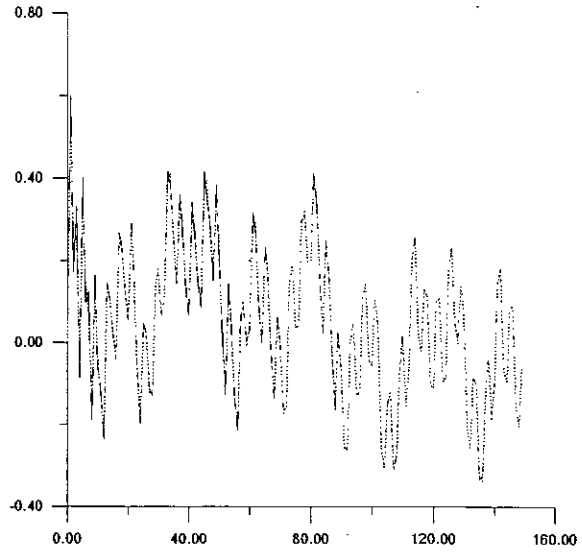


Figure 19

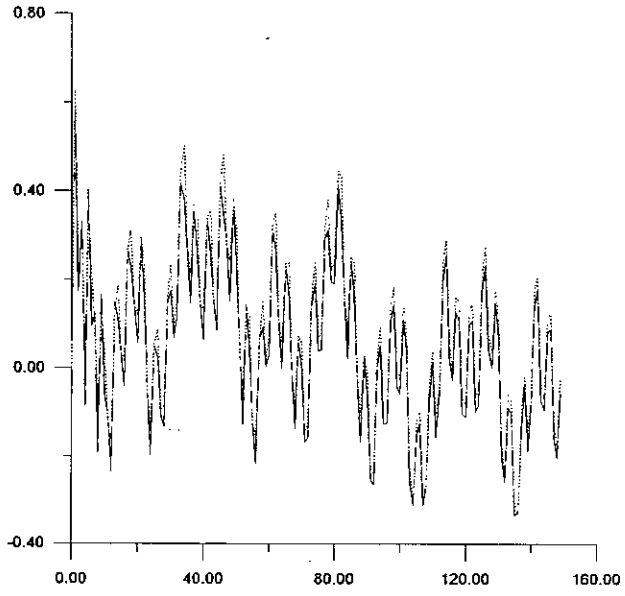


Figure 20

	$A_m = 0.1$	$A_m = 0.2$	$A_m = 0.5$
FFN	2.66×10^{-3}	1.00×10^{-3}	1.21×10^{-3}
RecN	2.66×10^{-4}	8.26×10^{-4}	8.46×10^{-4}

Tableau 1. Valeur de l'erreur $e = \frac{1}{T}\sum_{t=1}^{T}\left(\hat{y}(t) - y(t)\right)^2$, lors de l'introduction d'un bruit de mesure.

On remarque à partir du tableau 1 que, les réseaux réussissent à apprendre la valeur exacte du système y(t) malgré le bruit z(t). On peut, aussi remarquer que les réseaux récurrents en entrée donnent de meilleurs résultats, même quand le bruit possède une large amplitude.

VI. Conclusions :

Dans ce chapitre, on a présenté les méthodes qui permettent l'identification de quatre modèles des systèmes non-linéaires, et ceci en utilisant les réseaux de neurones généralisés qui sont une combinaison entre les réseaux de neurones statiques et des opérateurs linéaires; les entrées de ces réseaux sont les valeurs décalées des entrées et des sorties du système. On a ensuite éfféctué une comparaison entre deux types de réseaux l'un dépend du modèle le second n'en dépend pas; et puis on a étudié l'effet du nombre d'entrées, on a remarqué que le fait de changer le nombre d'entrées n'affectait pas sérieusement les résultats, ceci pourrait être expliqué en utilisant le théorème de Kolmogorov-Sprecher. Puis, on a éffectué l'identification d'un système en utilisant un réseau dynamique, on a remarqué que la convergence était rapide, mais que celle ci dépendait du choix des poids initiaux, du signal d'éxcitation, et des pas d'apprentissage. Et enfin, on a éfféctué une comparaison entre trois algorithmes d'apprentissage des réseaux de neurones, et on a ainsi pu remarquer que le modèle parallèle était plus robuste par rapport aux bruits de mesures.

Chapitre III

COMMANDE PAR RESEAUX DE NEURONES

" La tache suprême du physicien est d'arriver a des lois élémentaires universelles telles que les cosmos puisse être construit a partir d'elles par pure déduction. Aucune voie logique ne conduit a ces lois, mais la seule intuition qui repose sur une intelligence compréhensive."

Albert EINSTEIN (1879-1955)

CHAPITRE III
COMMANDE PAR RESEAUX DE NEURONES

I. Introduction :

Contrairement à la commande linéaire qui est générique[1] , la commande non-linéaire ne l'est pas, c'est pour cela que des nouvelles approches doivent être considérées. Les réseaux de neurones avec leurs capacité d'approximer n'importe quelles fonctions pourraient présenter une solution au problème de généricité de la commande non-linéaire.

L'utilisation des réseaux de neurones dans la commande des systèmes non-lineaires peut être vue comme une évolution naturelle des techniques de commande. Cette évolution est fondée sur plusieurs points: 1) Capacités limitées de la part des régulateurs classiques , 2) Analyse des non-linéarités dures (Il existe de non-linéarités qui ne peuvent être linéarisées (saturation, zone morte...)). 3) Utilisation d'un nombre minimal d'informations sur le système. Les réseaux de neurones possédant les propriétés de la non-linéarité et de généralisation peuvent être utilisés efficacement dans la commande non-linéaire.

Dans le présent chapitre différentes techniques de commande vont être présentées. On commencera par la commande supervisée, on parlera ensuite de la commande inverse qui sera réalisée grâce à quatre techniques; apprentissage indirect, apprentissage généralisé, apprentissage spécialisé et "feedback error learning", ainsi que de la commande adaptative avec modèle de référence, de la commande linéarisante, du régulateur auto-ajustable, du rejet de perturbations dans les systèmes non-linéaires par réseaux de neurones, et enfin de la synthèse de contrôleurs basés sur les fonctions critiques, on terminera ce chapitre avec des simulations et une conclusion.

II. Commande Supervisée :

Cette approche appelée parfois "Copying an existing controller", est basée - comme son nom l'indique - sur l'utilisation d'un régulateur existant (qui pourrait être un opérateur humain), comme fonction inconnue à approximer par un réseau de neurones (Fig.1)[Miller *et al*, 1990]. Les sorties désirées du réseau de neurones pour une entrée donnée sont les valeurs de la commande générés par le contrôleur existant.

Cette approche peut être vue comme une méthode de construction d'un système expert neuronal, par l'acquisition du "savoir-faire" d'un expert [Miller *et al*, 1990][Hunt *et al*, 1992].

[1] La commande d'une classe de systèmes est dite générique, si la solution au problème de la commande ne dépend pas des éléments de cette classe.

La question concernant l'utilité de cette méthode possède plusieurs réponses: Premièrement, le régulateur peut être un élément impraticable lors de l'utilisation (par exemple, l'être humain), deuxièmement , le réseau de neurones étant un approximateur de fonctions, on peut donc avoir une forme explicite (sous forme d'équation mathématique) du régulateur ceci permettrait une analyse. En plus, les réseaux de neurones possédant la capacité de calcul parallèle et distribué, permettra une meilleure implantation du contrôleur existant.

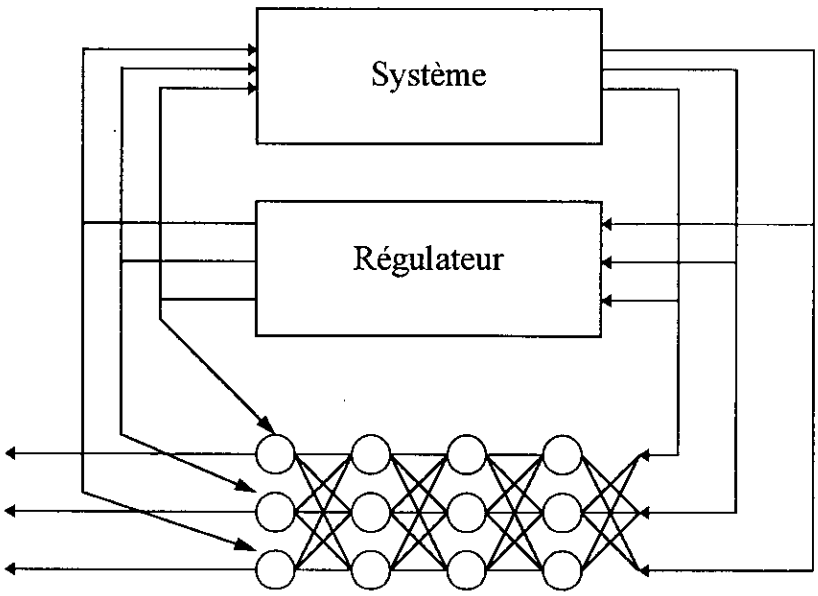


Figure 1: Commande Supervisée

III. Commande Inverse :

La commande inverse est basée sur la détermination du modèle inverse du système à commander. Ce modèle inverse sera simplement mis en cascade avec le système afin que le système résultant soit une fonction identité entre l'entrée et la sortie du système (Fig.2) [Miller *et al*, 1990][Hunt *et al*, 1992].

Dans la figure 2, les sorties désirées sont notées par y_d, et les sorties réelles du système sont notées par y. Le régulateur neuronal doit approximer l'inverse du système, et produira, ainsi, a partir de la réponse désirée y_d le signal de commande u afin que y soit proche de y_d. Pour approximer l'inverse du système plusieurs approches d'entraînement du réseau de neurones existent. [Miller *et al*, 1990] [Hunt *et al*, 1992] [Fukuda and Shibata, 1992] [Kung and Hwang, 1989].

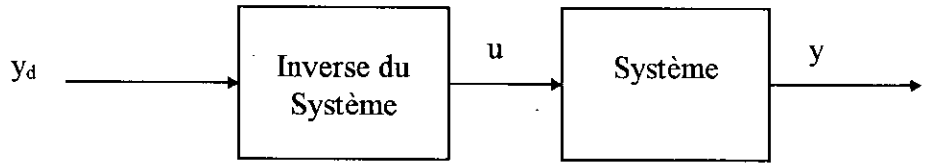


Figure 2: Utilisation d'un Réseau de Neurones pour la Commande Inverse.

1. Méthode d'Apprentissage Indirect :

La figure 3 présente une possibilité pour l'entraînement des réseaux agissant comme l'inverse du système. On doit entraîner le réseau de neurones par l'adaptation de ses poids afin de minimiser l'erreur $e_1 = u - t$, en utilisant l'architecture présentée sur la figure 3 . dans ce cas la minimisation de $e = y_d$-y entraîne la minimisation de e1 .

L'avantage de cette méthode est quelle permet entraîner le réseau uniquement dans le domaine de fonctionnement désiré puisque on commence par la sortie désirée y_d, et tous les autres signaux sont générés a partir d'elle. Notons qu'il est avantageux d'adapter les poids pour minimiser l'erreur e =yd-y a la sortie du système directement.

Malheureusement, cette méthode, n'est pas une procédure valide parce que minimiser e ne revient pas nécessairement a minimiser e_1. Comme cité dans[Psaltis *et al*, 1988], il existe des systèmes ou le réseau de neurones tend a converger vers une solution qui fait correspondre a toutes les entrées une valeur unique $u = u_0$, pour laquelle $e_1 = u - t$ est nulle , mais évidemment e ne l'est pas

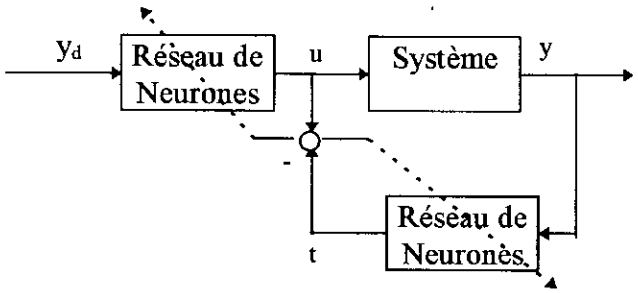


Fig.3 :Apprentissage Indirect

2. Méthode d'Apprentissage Généralisé :

L'architecture présentée sur la figure.4 fournit une méthode d'entraînement des réseaux de neurones qui minimise l'erreur e^2. La procédure d'entraînement est la suivante:

Une entrée du système est sélectionnée et appliquée au système pour obtenir la valeur de *y* correspondante. Le réseau de neurones est entraîne pour reproduire *u* à sa sortie a partir de *y*. Le réseau de neurones entraîné doit ensuite être capable de

produire les valeurs appropriées de *u* pour un signal de référence y_d donné. Cette méthode ne sera efficace que si la référence y_d est suffisamment proche de l'un des signaux d'entraînement (i.e. des y utilisées).

Cependant, le succès de cette méthode est intimement lié à l'aptitude du réseau de neurones de généraliser aux cas non appris, et de répondre correctement dans le domaine de fonctionnement désiré, normalement on ne connaît pas les entrées du système correspondantes aux sorties désirées. Ainsi, on essaye de distribuer uniformément l'espace de entrée du système par les exemples d'entraînement afin que le réseau puisse interpoler. Dans ce cas, la méthode d'apprentissage généralisé ne peut être efficace puisque le réseau de neurones est entraîne pour apprendre les réponses du système sur un espace large plus que le nécessaire.

Une solution est de combiner l'apprentissage général avec l'apprentissage spécialisé présente ci-dessous.

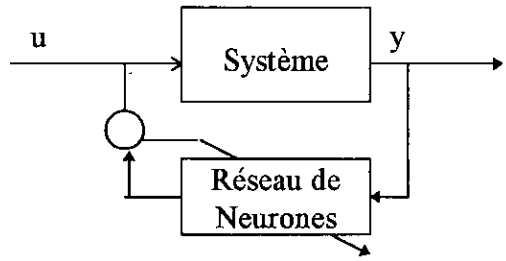


Fig. 4 : Apprentissage Généralisé

3. Méthode d'Apprentissage Spécialisé :

La figure.5 présente une architecture pour entraîner un réseau de neurones pour opérer proprement dans le domaine de fonctionnement seulement.

L'entraînement implique l'utilisation de la réponse désirée y_d comme entrée pour le réseau. Le réseau est entraîné pour trouver les entrées du système ,u, qui conduit la sortie du système, y, aux valeurs désirée y_d. cette opération est accomplie par l'utilisation de l'erreur entre la réponse désirée et les sorties réelles du système pour ajuster les poids du réseau de neurones par l'algorithme de la "backpropagetion", à chaque itération les poids du réseau sont ajustes pour minimiser l'erreur. L'avantage de cette méthode est qu'elle permet entraîner le réseau dans le domaine de fonctionnement désiré, et de produire un modèle inverse particulier du système quand l'inverse du système est mal définie. Le problème de la commande inverse directe est l'instabilité du système global, parce que le réseau commande le système directement,
pour pallier ce problème, il est nécessaire de préparer des valeurs initiales des poids du réseau, qui peuvent être obtenues par un entraînement off-line (par exemple, par la méthode d'apprentissage général).

Dans cette méthode, l'erreur entre la sortie du système y et la sortie désirée yd doit être rétropropagée à travers le système afin d'avoir l'erreur a la sortie du réseau de neurones, utilisée dans l'entraînement.

Assumons que le critère a minimiser est le suivant :

$$J = (y_d - y)^2 \tag{III-1}$$

$$\frac{\partial J(k+1)}{\partial u(k)} = -(y_d - y)\frac{\partial y(k+1)}{\partial u(k)} \tag{III-2}$$

Dans le cas ou le système est une fonction inconnue, l'estimation de la dérivée partielle de sa sortie par rapport a son entrée pose des difficultés. Deux solutions sont pour le calcul de cette dérivée partielle existent :.

- *Calcul numérique du Jacobien du Système*:
  Dans cette approche, deux méthodes peuvent être utilisées pour approximer la dérivée partielle :

  a) $$\frac{\partial y(k+1)}{\partial u(k)} = \frac{y(k+1)-y(k)}{u(k)-u(k-1)} \tag{III-3}$$

  b) Variation de entrée autour du point de fonctionnement, et mesurer la variation correspondante de la sortie, i.e.

  $$\frac{\partial y(k+1)}{\partial u(k)} = \frac{f(\bar{u}+\delta u)-f(\bar{u})}{\delta u} \tag{III-4}$$

- *Calcul de la dérivée en utilisant un modèle neuronal du système*:
  Dans ce cas, un modèle du système est placé en parallèle avec le système (fig 6). L'utilité du modèle direct du système est qu'il permet de calculer efficacement la dérivée partielle de la sortie du modèle par rapport a son entrée en utilisant la "backpropagation" , dans ce cas le régulateur neuronal et le modèle peuvent être vus comme un seul réseau.

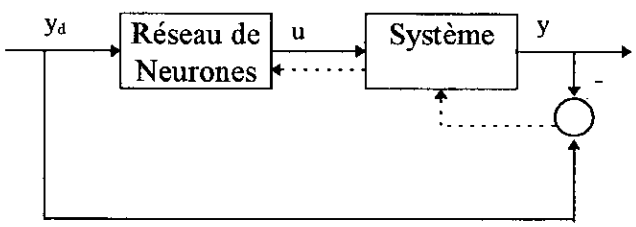


Fig. 5 : Apprentissage Spécialisé

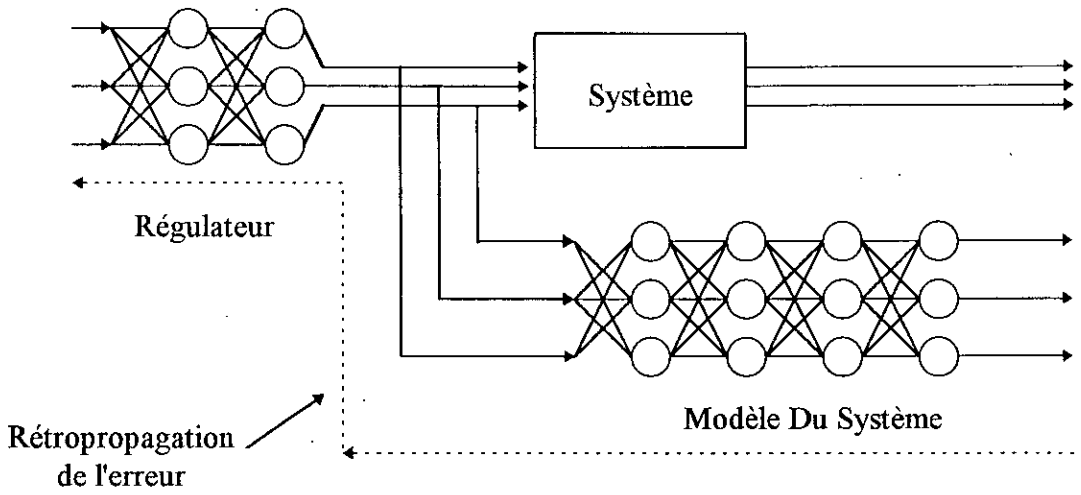


Fig. 6 : Rétropropagation de l'erreur de sortie à travers le modèle du système

4." Feedback Error Learning" :

La figure 7 présente la structure de l'apprentissage par cette méthode. Dans cette méthode le réseau de neurones est utilisé dans la boucle d'anticipation ("feedforward controller"), et est entraîné par l'utilisation du signal d'erreur produit a la sortie du régulateur de la chaîne directe. Le problème de cette méthode est qu'elle utilise des informations à priori sur le système et qui ne sont pas toujours disponibles.

Par exemple, elle utilise dans la construction du réseau les fonctions figurants dans le modèle du système comme les fonctions sinus et cosinus, en plus, elle utilise l'accélération comme entrée pour le réseau et qui ne peut être obtenu correctement à cause du bruit (robots par exemple) . Les performances du modèle inverse résultant par cette méthode sont directement liées au régulateur utilise dans la chaîne directe [Miller *et al* , 1990][Fukuda and Shibata, 1992].

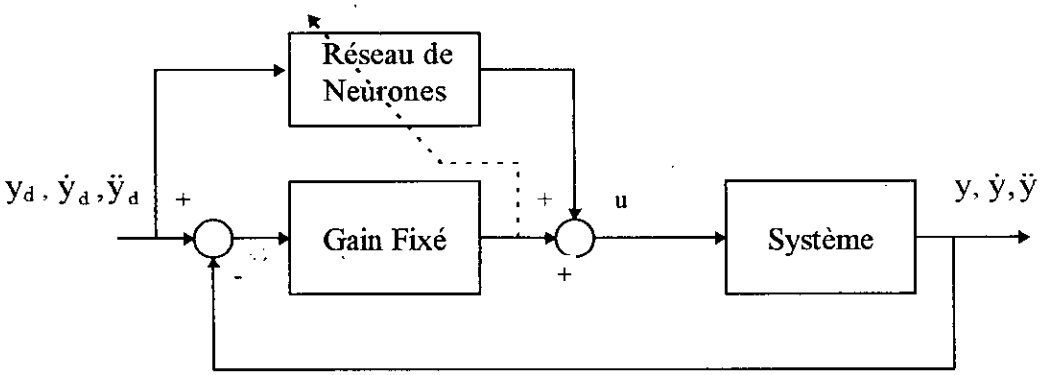


Fig. 7 : "Feedback Error Learning".

IV. Commande Adaptative avec Modèle de Référence :

La commande adaptative a été définie par Astrom [Astrom, 1987], comme suit :

" La commande adaptative est simplement un cas spécial de la commande non linéaire, telle que les états du système puissent être décomposés en deux parties , avec des changements à plusieurs niveaux . Les états qui changent lentement sont considérés comme étant des paramètres".

La recherche en commande adaptative commença dans les années 1950 afin de concevoir des pilotes automatiques possédant de grandes performances. Mais le fondement théorique était inexistant ; l'implantation était impossible ceci est dû au fait que la technologie des ordinateurs était encore a ces débuts, Cette technique fut rapidement oubliée surtout après des tests pratiques non concluants effectués sur des avions [Astrom ,1987].

Durant les années 1960, des développements importants en identification et en commande furent réalisés ,ceci a permis un regain d'intérêt, en ce qui concerne la commande adaptative .Et c'est pendant les années 1970 que la commande adaptative fut redécouverte, grâce aux fondements théoriques solides qui furent développés par les chercheurs .

Dans cette partie nous sommes intéressés par la commande adaptative avec modèle de référence, l'idée de base c'est de faire suivre le système , un certain modèle de référence .L'idée était proposée par Whitaker, Yamron et Kezer en 1958 et fut développée par Parks en 1966, Monopoli en 1974, et Landau en 1974. La question principale qui fut posée était l'étude de la stabilité du système régulé par cette méthode .Cette question était restée ouverte jusqu'à la fin des années 1970, mais fut finalement résolue par Egardt (1978,1980), Feuer et Morse (1978), Fuchs(1980), Goodwin, Ramadge et Caines (1978), Jeanneau et de Larminat (1975), Morse(1980), Narendra et Valvani(1978), et Narendra et Lin en 1980 [Goodwin,1984].

En commande adaptative avec modèle de référence on peut distinguer entre deux différentes étapes .Dans la première étape appelée '*étape algébrique*', on doit démontrer que le régulateur possède suffisamment de degrés de liberté pour pouvoir satisfaire le cahier de charges .Plus précisément ,si nous possédons des informations à priori sur le système , il fût démontré que le vecteur des paramètres du régulateur θ ,existe pour n'importe quelles valeurs du vecteur de paramètres p du système , tel que le système régulé suive asymptotiquement la référence .La seconde étape appelée '*étape analytique*' , concerne la détermination de lois 'stables' d'adaptation du vecteur $\theta(k)$ tel que $\lim_{k\to\infty} \theta(k) = \theta$,et que l'erreur de sortie tende vers zéro.

Durant les vingt dernières années , deux approches ont été utilisées en commande adaptative .La première est la *commande directe ("direct control")*,la seconde est la *commande indirecte ("indirect control")*.

En commande directe , les paramètres du régulateur sont ajustés directement de telle façon à minimiser une certaine norme de l'erreur de sortie .

En commande indirecte ,les paramètres du système $\hat{p}(k)$ sont estimés à chaque instant ,et les paramètres du régulateur $\theta(k)$,sont calculés en supposant que $\hat{p}(k)$ représente la valeur exacte des paramètres du système p [Narendra and Annaswamy, 1989]

Quand le système dynamique est non-lineaire, on peut utiliser les réseaux de neurones comme des régulateurs.

1.Commande Adaptative Directe :

En commande adaptative directe, les paramètres du régulateur sont ajustés, directement de telle façon à minimiser une certaine norme de l'erreur de sortie.

Dans la théorie de la commande adaptative conventionnelle les méthodes d'ajustement des paramètres du régulateur basés sur la mesure de l'erreur de sortie utilisent les concepts de positivité et de passivité [Slotine and Li, 1991]. L'utilisation de la théorie de Lyapounov ou celle basée sur l'hyperstabilite, fournissent des algorithmes d'adaptation des paramètres qui permettent la stabilité du système bouclé [Landau, 1979] [Narendra and Annaswamy, 1989].

Une approche neuronale de cette commande est schématisée dans la figure 8 [Narendra and Parthasarathy, 1990]. Cependant, aucune preuve de la stabilité des lois d'adaptation des paramètres du réseau n'a été établie.

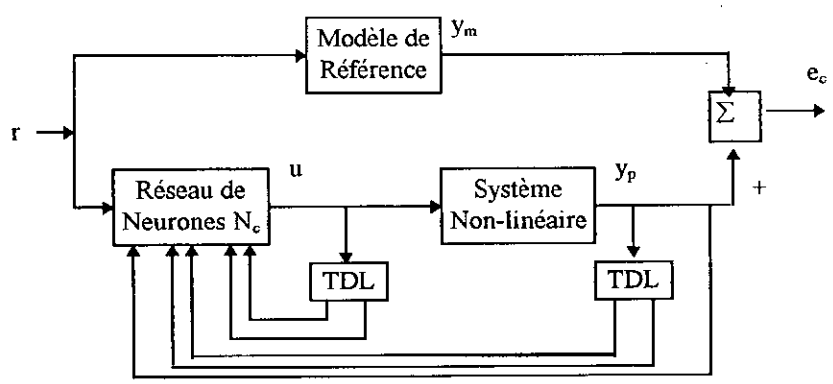


Fig. 8: Commande Adaptative Directe Par Réseaux de Neurones.

2. Commande Adaptative Indirecte :

En Commande Adaptative Indirecte, les paramètres du système sont estimés à chaque instant, et les paramètres du régulateur sont calculés en supposant que les paramètres du modèle sont les mêmes que ceux du système.

Quand la commande indirecte est utilisée pour la commande des systèmes non-lineaires, le système est parametrisé en utilisant l'un des modèles décrits dans le chapitre précèdent et les paramètres du modèles sont ajustes par l'erreur d'identification, les paramètres du régulateur sont ajustes par la rétropropagation de l'erreur entre les sorties du modèle de référence et les sorties du modèle .Un schéma bloc de cette méthode est présente dans la figure 9.

Les paramètres du régulateur et ceux du modèle d'identification peuvent être ajustes à chaque instant ou après un intervalle du temps. Dans le cas de la présence des bruits il est judicieux d'ajuster les paramètres du régulateur avec une cadence moins rapide que celle de l'ajustement des poids du modèles d'identification [Narendra and Parthasarathy, 1990].

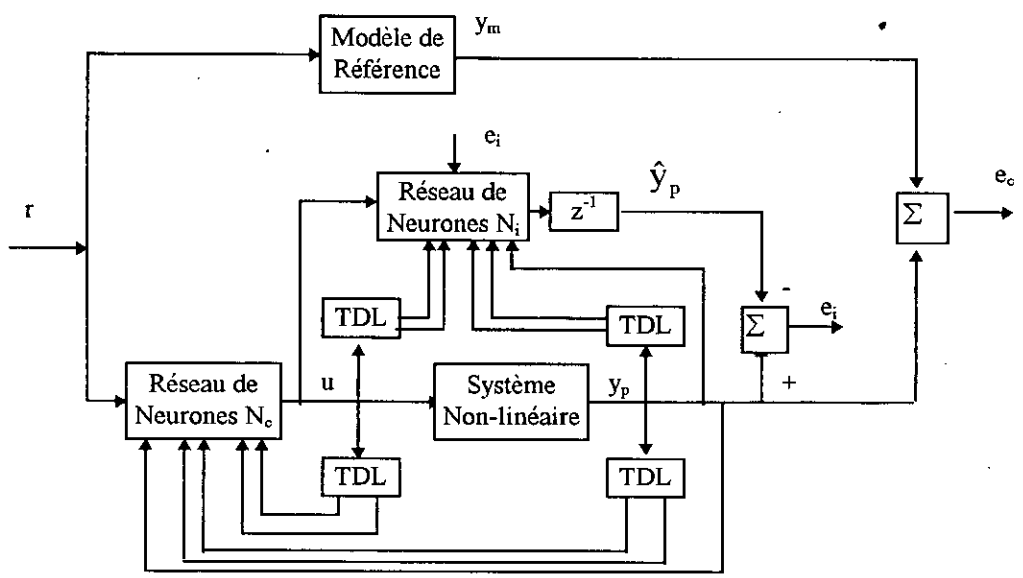


Fig. 9 : Commande Adaptative Indirecte Par Réseaux de Neurones

3. Connaissance à Priori :

Dans la théorie de la commande adaptative conventionnelle, ou le système est linéaire et invariant dans le temps mais avec des paramètres inconnus, les lois de commande stables sont déterminées seulement par la supposition que des informations considérables sur la fonction de transfert du système sont disponibles à priori [Miller *et al*, 1990].

En particulier, on suppose que :

i) L'ordre du système est connu.
ii) Les zéros du système sont stables
iii) le degré relatif de la fonction de transfert est connu.

Il est ainsi raisonnable d'assumer que des informations similaires existent pour les représentations entrée-sortie des systèmes non-lineaires.

4. Modèle de Référence :

Le modèle de référence est toujours assumé linéaire. Parce que la théorie des systèmes linéaires est bien développée et les méthodes de choix des modèles linéaires qui possèdent les propriétés désirées sont bien établies [Miller *et al*, 1990].

En revanche, les méthodes de détermination de modèles de références non-lineaires ne sont pas disponibles. C'est pour cela qu'on utilise des modèles de références linéaires.

Que le modèle de référence soit linéaire ou non-linéaire, une considération pratique est que le système global (système à commander plus le régulateur) doit approximer asymptotiquement le modèle de référence ($\lim_{k\to\infty} e(k) = 0$). Cependant, une question théorique importante concerne la génération des modèles de références qui satisfont ce critère [Narendra and Parthasarathy, 1990].

5. Rejet de Perturbations :

Le rejet de perturbations est un problème qui est important, car c'est à cause des perturbations que le méthodes classiques de régulation ("Feedback") ont été introduites. En fait, ce sont les perturbations qui limitent les performances du système régulé [Astrom and Wittenmark, 1990]. Et c'est, en étudiant le comportement des perturbations qu'il serait possible d'améliorer ces performances.

Dans cette partie le rejet de perturbations dans les systèmes non-linéaires est traité en utilisant les réseaux de neurones.

C'est en considérant d'équations aux différences linéaires, ou non-linéaires telles que l'équation de Van Der Pol ou celle de Duffing, qu'une classe importante de perturbations peut être modélisées [Parker and Chua, 1987] . Et c'est grâce à de tels modèles que le rejet de perturbations est possible. En effet, en considérant le modèle d'un système SISO en l'absence de perturbations :

$$\begin{aligned} x_p(k+1) &= f_1\left(x_p(k), u(k)\right) \\ y(k) &= h_1\left(x_p(k)\right) \end{aligned} \qquad \text{(III-5)}$$

avec : $x_p(k) \in \Re^n$ est le vecteur des états du système à l'instant k, $u(k) \in \Re$ est l'entrée appliquée au système à l'instant k, y(k) est la sortie de ce système à l'instant k. Supposons qu'une représentation entrée-sortie existe, i.e. :

$$y(k+1) = F_1[Y(k), U(k)] \tag{III-6}$$

avec: $Y(k) \in \Re^n = [y(k), y(k-1), .., y(k-n+1)]^T$ et $U(k) \in \Re^n = [u(k), .., u(k-n+1)]^T$

Une perturbation externe est décrite par le système d'équations aux différences suivant:

$$\begin{aligned} x_v(k+1) &= f_2(x_v(k)) \\ v(k) &= h_2(x(k)) \end{aligned} \tag{III-7}$$

avec: $x_v(k) \in \Re^p$ est l'état du système générant la perturbation et $v(k) \in \Re$ est la sortie correspondante. On remarque, que la perturbation et solution d'une équation aux différences libre. Supposons, en plus, qu'une représentation entrée-sortie est possible :

$$v(k+1) = F_2(\overline{V}(k)) \tag{III-8}$$

avec: $\overline{V}(k) = [v(k), v(k-1), \ldots, v(k-p+1)]^T$. En supposant, que v(k) est une autre entrée du système (III-5), le système global sera décrit par :

$$\begin{aligned} x_p(k+1) &= f_3[x_p(k), u(k), v(k)] \\ x_v(k+1) &= f_2[x_v(k)] \\ v(k) &= h_2[x_v(k)] \\ y(k) &= h_1[x_p(k)] \end{aligned} \tag{III-9}$$

ceci est équivalent à un nouveau système "sur-paramétrisé" décrit par :

$$\begin{aligned} x_0(k+1) &= f_4[x_0(k), u(k)] \\ y(k) &= h_1[x_p(k) = h_4[x_0(k)]] \end{aligned} \tag{III-10}$$

avec : $x_0(k) = [x_p^T, x_v^T]^T \in \Re^{n+p}$ est l'état du système global. On remarque, que malgré que l'entrée et la sortie du système est la même, la dimension de l'espace d'état du système global a augmenté . Le problème qui se pose maintenant est le même que celui de la commande adaptative - comme dans la section ci-dessus - , ainsi on devrait déterminer la commande u(k) qui fasse tendre l'erreur de sortie vers zéro.

Malheureusement, cette structure n'est pas directement applicable, et le même

type de modèles utilisés en identification doit être utilisé - pour plus de précisions voir les simulations ci-dessous -.

V. Bouclage Linéarisant ("Feedback Linearization"):

Cette méthode est basée sur la linéarisation et le découplage des systèmes régit par le système d'équations différentielles suivant :

$$M(\theta)\ddot{\theta} + N(\theta,\dot{\theta}) + F = T \qquad (III\text{-}11)$$

Ce modèle décrit, plus particulièrement, un bras de robot qui est caractérisé par une structure arborescente articulée simple ou multiple dont les segments sont mobiles les uns par rapport aux autres. Dans le cas d'un robot les quantités de l'équation (III-11), sont définies par :

T: Vecteur $n \times 1$ des couples appliqués aux articulations.
M : Matrice $n \times n$ représentant la matrice d'inertie du manipulateur.
N : Vecteur $n \times 1$ représentant les forces centrifuges, les forces de Coriolis, l'effet de la gravitation et les frottements.
F: Vecteur $n \times 1$ représentant les termes inconnus résultant des dynamiques non-modélisées et des perturbations externes.
: Vecteur $n \times 1$ représentant les positions des articulations.
: Vecteur $n \times 1$ représentant les vitesses des articulations.

Cette commande exige la connaissance exacte du modèle du système (III-11), c'est pour cela qu'on l'appelle parfois commande linéarisante à paramètres connus. Le comportement du système bouclé avec cette loi de commande est identique à celui d'un système du second ordre. En effet, en choisissant :

$$T = \hat{M}(\theta)\,T' + \hat{N}(\theta,\dot{\theta}) \qquad (III\text{-}12)$$

En injectant (III-12) dans (III-11) on obtient alors :

$$M(\theta)\ddot{\theta} + N(\theta,\dot{\theta}) = \hat{M}(\theta)\,T' + \hat{N}(\theta,\dot{\theta}) \qquad (III\text{-}13)$$

Dans le cas ou le modèle du robot est connu, et que la matrice $M^{-1}(\theta)$ est non singulière l'équation (III-12) devient :

$$\ddot{\theta} = T' \qquad (III\text{-}14)$$

La transformation (III-12), transforme le problème de la commande du système (III-11), en celui de la commande de n doubles intégrateurs découplés. L'entrée T' peut être interprétée comme une commande de la boucle externe qui devra asservir le système. Elle peut être obtenue par un compensateur linéaire proportionnel dérivé (PD)

ou un retour d'état pour le système linéarisé [Craig and Sastry, 1987],[Slotine and Li, 1991] :

$$T' = \ddot{\theta}_d - k_v \dot{e}(t) - k_p e(t) \qquad \text{(III-15)}$$

avec: k_p, k_v sont des matrices de dimensions appropriées.

Les vecteurs $e(t) = \theta_d - \theta, \dot{e}(t) = \dot{\theta}_d - \dot{\theta}$ sont l'erreur de suivi des trajectoires et sa dérivée. En injectant (III-15) dans (III-11), on obtient la dynamique de l'erreur :

$$\ddot{e} + k_v \dot{e} + k_p e = \hat{M}^{-1}\left(\left(M - \hat{M}\right)\ddot{\theta} + N - \hat{N} + F\right) \qquad \text{(III-16)}$$

On remarque, que si les erreurs d'estimation sont nulles, un terme inconnu existe et la convergence vers une erreur nulle n'est pas assurée. La figure 10 schématise cette commande, qui est appelée en robotique "Computed Torque Method".

Une approche neuronale de cette méthode a été proposée dans [Ozaki *et al*, 1991] , on l'appelle *"model learning "*, elle s'effectue en deux étapes :

- *Synthèse de la commande en se basant sur un modèle simple du robot*, i.e. dans ce cas on néglige la dynamique non-modélisée (F=0 dans (III-11)), et entraîner le réseau à apprendre le vecteur des commandes.(Fig.11)
- *Apprentissage spécialisé* en démarrant avec le réseau précédent (Fig.5) .

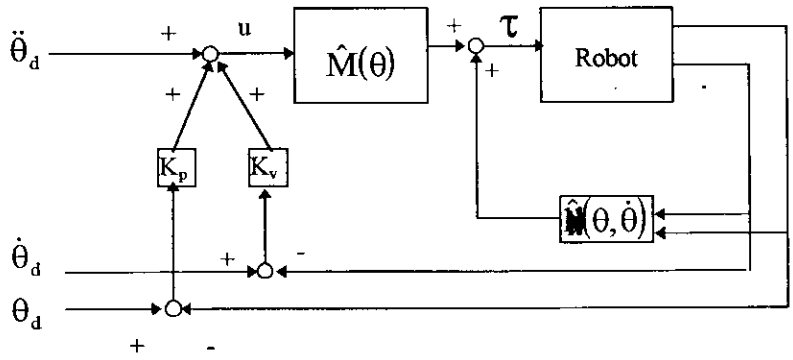


Fig. 10 : Controlleur basé sur le bouclage linéarisant

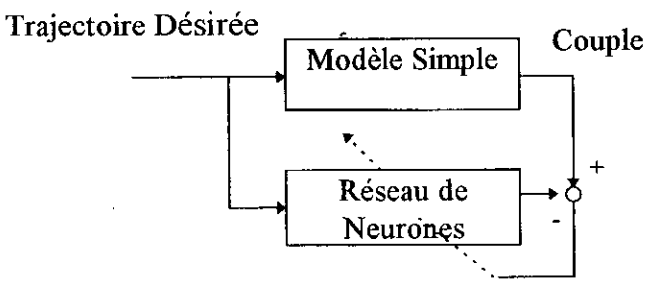


Fig. 11 :"Model Learning"

VI Régulateur Auto-ajustable :

Le régulateur auto-ajustable est essentiellement un retour classique avec des paramètres ajustés en temps réel, ce régulateur est composé de deux boucles, la boucle interne est constituée par le système à commander et d'un régulateur ordinaire, la boucle externe contient l'algorithme qui permet d'ajuster les paramètres du régulateurs [Astrom, 1987]. L'approche neuronale de cette méthode a été proposée par [Chen, 1990], elle est applicable aux systèmes non linéaires qui possèdent la forme :

$$\begin{aligned} y(k+1) = f\big(y(k), y(k-1), \cdots y(k-p), u(k-1), u(k-2), \cdots, u(k-p)\big) + \\ g\big(y(k), y(k-1), \cdots, y(k-p), u(k-1), u(k-2), \cdots, u(k-p)\big)\, u(k) \end{aligned} \qquad \text{(III-17)}$$

Si les fonctions $f(\cdot)$, et $g(\cdot)$ sont connues, alors pour que $y(k+1) = d(k+1)$, on doit avoir :

$$u(k) = -\frac{f(\cdot)}{g(\cdot)} + \frac{d(k+1)}{g(\cdot)} \qquad \text{(III-18)}$$

Un réseau de neurones dont le modèle est décrit par :

$$\begin{aligned} \hat{y}(k+1) = \hat{f}\big(y(k), y(k-1), \cdots y(k-p), u(k-1), u(k-2), \cdots, u(k-p), W(k)\big) + \\ \hat{g}\big(y(k), y(k-1), \cdots, y(k-p), u(k-1), u(k-2), \cdots, u(k-p), W(k)\big)\, u(k) \end{aligned} \qquad \text{(III-19)}$$

permettrait d'estimer les fonctions inconnues et de réguler le système; le schéma de cette commande est montré dans la figure 12. En effet, en supposant que :

$$u(k) = -\frac{\hat{f}(\cdot)}{\hat{g}(\cdot)} + \frac{d(k+1)}{\hat{g}(\cdot)} \qquad \text{(III-20)}$$

et en injectant cette équation dans (III-17), nous obtenons le système bouclé qui possédera la forme :

$$y(k+1) = f(\cdot) + g(\cdot)\left\{-\frac{\hat{f}(\cdot)}{\hat{g}(\cdot)} + \frac{d(k+1)}{\hat{g}(\cdot)}\right\} \qquad \text{(III-21)}$$

Ainsi, il ne nous reste plus qu'à minimiser l'erreur :

$$E(k) = \frac{1}{2}\big(d(k+1) - y(k+1)\big)^2 \qquad \text{(III-22)}$$

en utilisant l'algorithme de "backpropagation".

Pour permettre la minimisation, le système est identifié en temps réel, et la commande sera calculée à partir de (III-20). Si le réseau apprend à approximer le système, l'apprentissage est interrompu.

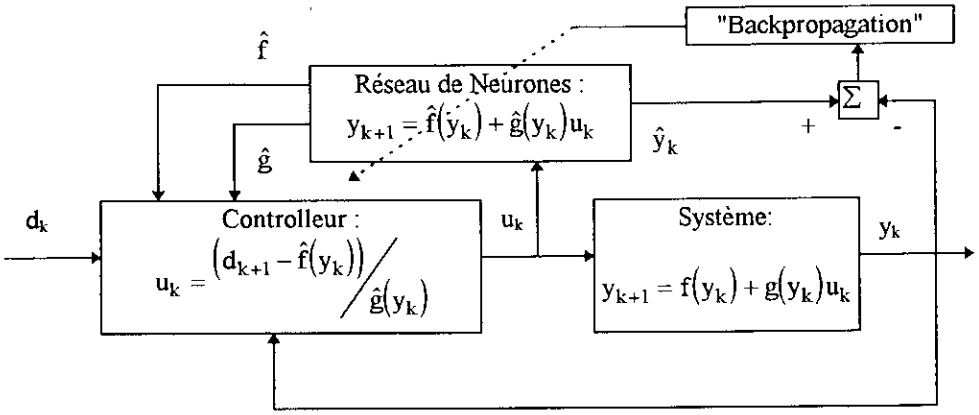


Figure 12 : Schéma d'un régulateur auto-ajustable par réseaux de neurones

VII. Commande par les Fonctions Critiques :

Cet approche diffère des autres approches par le fait qu'elle assume que le système est inconnu, l'erreur utilisée par l'algorithme d'apprentissage est fournie uniquement à travers une fonction critique voir Figure 13, [Hunt *et al*, 1992], [Miller *et al*, 1990], [Fukuda and Shibata, 1992], [Sanner and Akin, 1990].

Dans ce cas, au lieu entraîner le réseau de neurones avec une commande stabilisatrice prédéterminée, on veut évaluer l'aptitude du réseau à produire sa propre stratégie de commande basée uniquement sur la corrélation entre l'évolution du système et la fonction critique [Miller *et al*, 1992], [Sanner and Akin, 1990]. Donc la fonction critique doit être capable d'évaluer les performances du système (cette évaluation n'est pas forcement directement liée aux sorties désirées), et de générer un signal d'évaluation qui est utilisé par l'algorithme d'entraînement et qui permet un apprentissage efficace et une augmentation dans les performances du système [Miller *et al*, 1990], [Hunt *et al*, 1992], [Fukuda and Shibata, 1992].

Cet approche est appropriée quand il y a un véritable manque d'informations sur le système permettant d'appliquer les méthodes les plus spécialisées .

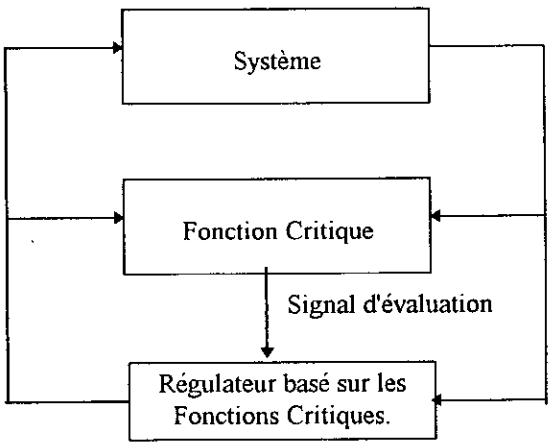


Figure 13 : Régulateur basé sur les fonctions critiques .

VIII. Simulations :

Dans cette partie nous allons présenter des simulations sur quelques méthodes citées dans ce chapitre .

1. Commande Supervisée:

Dans cette simulation une commande supervisée en utilisant les réseaux de neurones dynamiques est appliquée au pendule inversé, l'utilisation de ce type de réseaux a pour but de faire un comparaison avec les résultats de Guez et Selinsky [Guez and Selinsky, 1988] ou la même commande a été appliquée mais en utilisant les réseaux de neurones statiques . Le pendule inversé est le plus simple des systèmes intrinsèquement instable. Il a d'ailleurs toujours été utilisé pour tester les nouvelles commandes car il possède plusieurs caractéristique attrayantes :
1) Le système est non-linéaire et couplé, 2) Le système est instable en boucle ouverte, 3) Il a plusieurs implications pratiques (stabilisation des satellites, démarrage des fusées...).

Le pendule inversé possède une longueur L, une masse m, la masse du chariot, sur lequel il est monté, est M (Fig.14). Le système opère dans le plan vertical et est contrôlé en utilisant la force horizontale u. Le but du contrôleur est d'avoir $\theta = 0$ et $X = 0$. Les équation qui régissent ce système sont données par [Guez and Selinsky, 1988] :

$$\ddot{\theta} = \frac{3}{4L}\left(g \sin(\theta) - \ddot{X} \cos(\theta)\right) \qquad \text{(III-23)}$$

$$\ddot{X} = \frac{m\left(L \sin(\theta)\dot{\theta}^2 - \frac{3}{8} g \sin(2\theta)\right) - f\dot{X} + u}{M + m\left(1 - \frac{3}{4}\cos^2(\theta)\right)} \qquad \text{(III-24)}$$

On définit le vecteur d'état de ce système est la position et la vitesse du chariot, et la position et la vitesse du pendule :

$$Z = \left[X, \dot{X}, \theta, \dot{\theta}\right] \qquad \text{(III-25)}$$

La commande utilisée est celle basée sur la linéarisation et le découplage en utilisant la géométrie différentielle (FLDT). Dans ce cas, on doit transformer le variables d'état de telle façon à ce que le système "vu" par ces états soit linéaire commandable. En réécrivant, les équations (III-23) et (III-24) sous la forme :

$$\ddot{\theta} = h_1 - h_2 \ddot{X} \qquad \text{(III-26)}$$

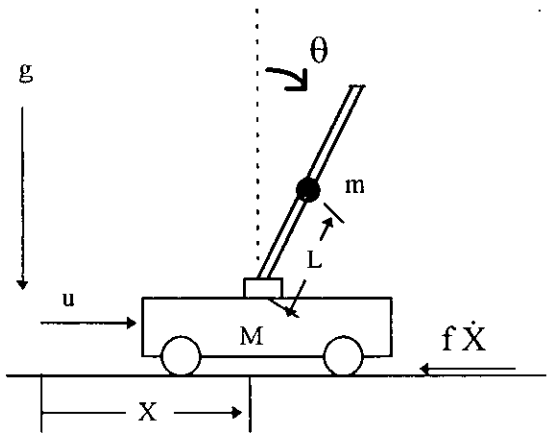


Fig. 14 :Pendule Inversé

$$\ddot{X} = \frac{f_1 + u}{f_2} \qquad \text{(III-27)}$$

avec :

$$h_1 = \frac{3}{4L} g \sin(\theta) \qquad \text{(III-28)}$$

$$h_2 = \frac{3}{4L} \cos(\theta) \qquad \text{(III-29)}$$

$$f_1 = m\left(L \sin(\theta)\dot{\theta}^2 - \frac{3}{8} g \sin(2\theta)\right) - f\dot{X} \qquad \text{(III-30)}$$

$$f_2 = M + m\left(1 - \frac{3}{4} \cos^2(\theta)\right) \qquad \text{(III-31)}$$

La commande u qui sera utilisée est donnée par [Guez and Selinsky, 1988] :

$$u = \frac{f_2}{h_2}\left[h_1 + k_1(\theta - \theta_d) + k_2\dot{\theta} + c_1(X - X_d) + c_2\dot{X}\right] - f_1 \qquad \text{(III-32)}$$

avec : $k_1 = 25$, $k_2 = 10$, $c_1 = 1$ et $c_2 = 2.6$.

Un réseau dynamique à 4 entrées, 3 neurones dynamiques et une sortie, a été entraîné pendant 10000 itérations, selon l'algorithme "Fixed Point Learning" dans sa version qui minimise l'erreur instantanée. Les résultats de l'apprentissage sont montrés dans les figures 15, 16. U_n est la valeur normalisée de la commande . On remarque que les sorties sont indiscernables, ce n'est pas le cas lors de l'utilisation d'un réseau statique.

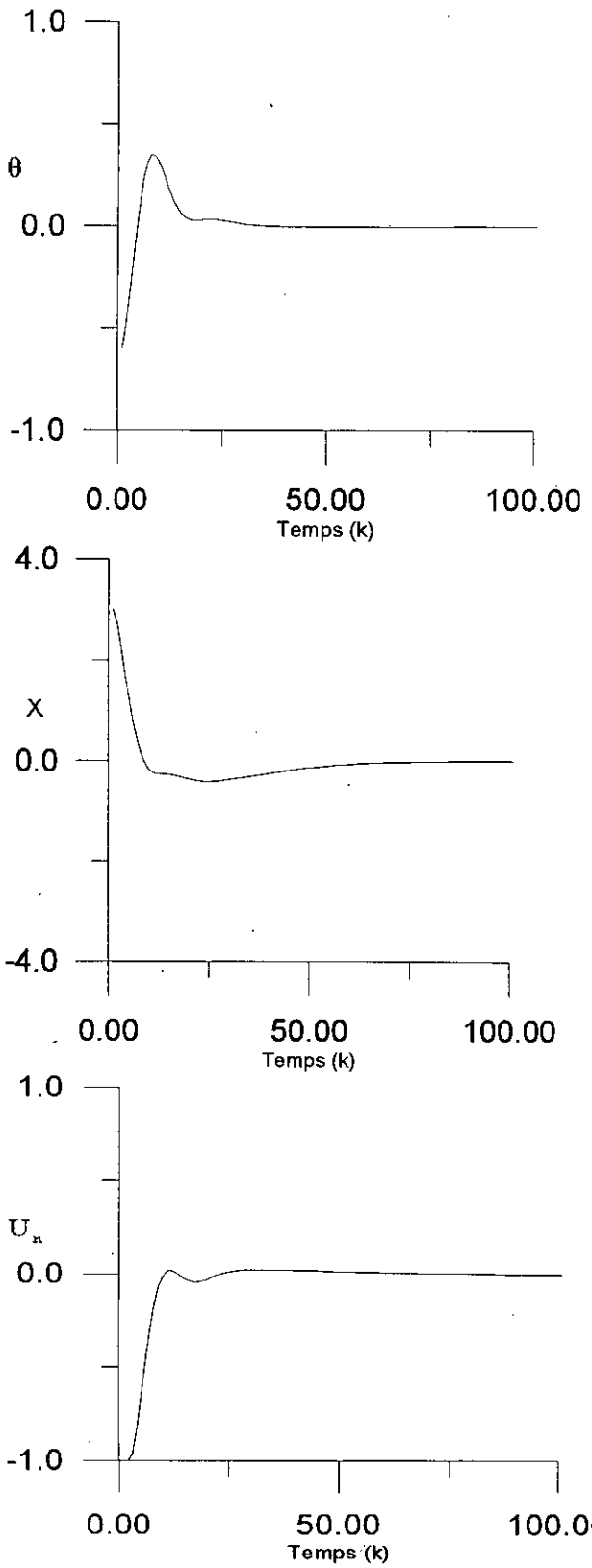


Figure 15 : Résultats de l'apprentissage avec FLDT
$Z(0) = (3,0, -0.6, 0)^T$

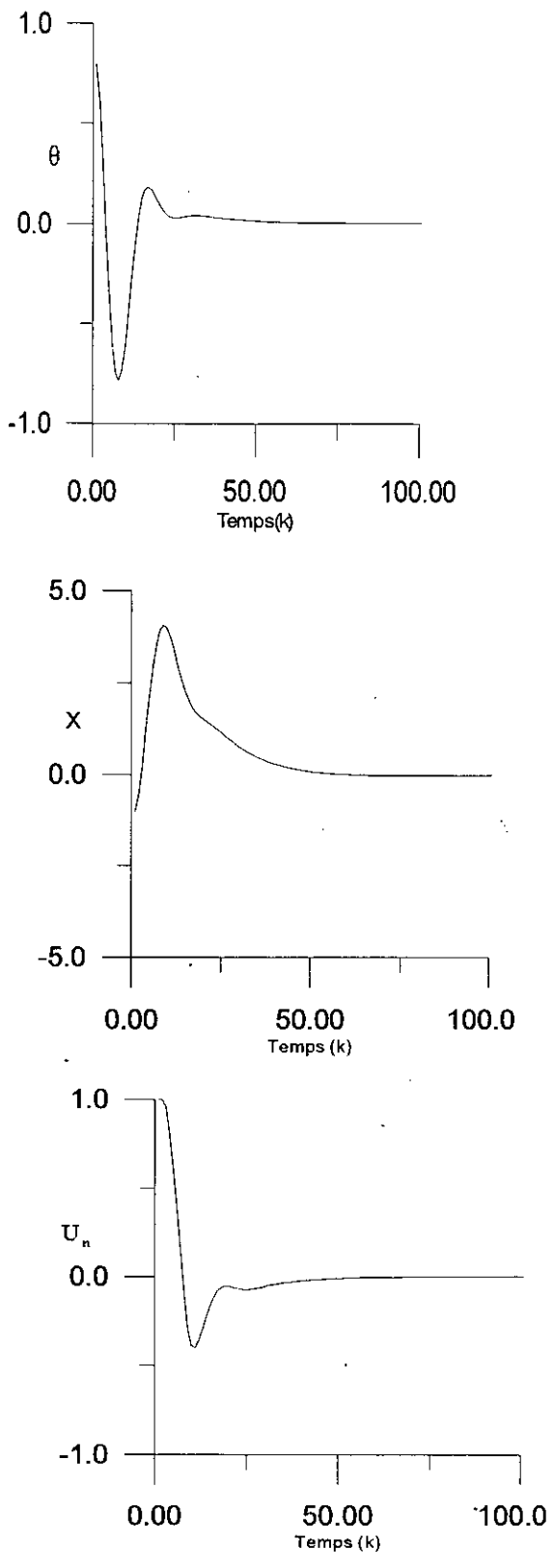


Figure 16: Résultats de l'apprentissage avec FLDT
$Z(0) = (-1, 0 , 0.8, 0.)^T$

On a ensuite effectué des tests de robustesse, le premier est basé sur le changement de la masse du pendule m, le second sur le changement de la valeur maximale de la commande U_n , le troisième est basé sur le changement du coefficient de frottement f , le quatrième sur le changement de la longueur du pendule, le cinquième sur la masse du chariot. On remarque que les sorties sont indiscernables excepté dans le cas du changement de la longueur (car elle intervient directement dans la linéarisation, le contrôleur neuronal peut au mieux implanter le régulateur donné).D'après l'article de Guez et Selinsky la commande supervisée en utilisant les réseaux statiques n'était pas si robuste. Tandis que le réseau dynamique l'est.

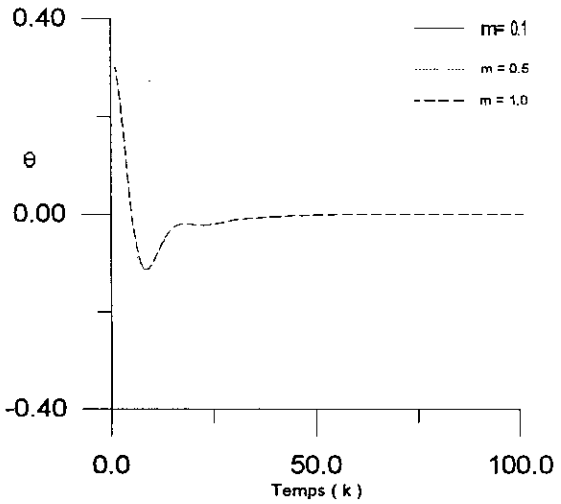


Figure 17 : Test de la robustesse du contrôleur neuronal face à la variation de la masse du pendule

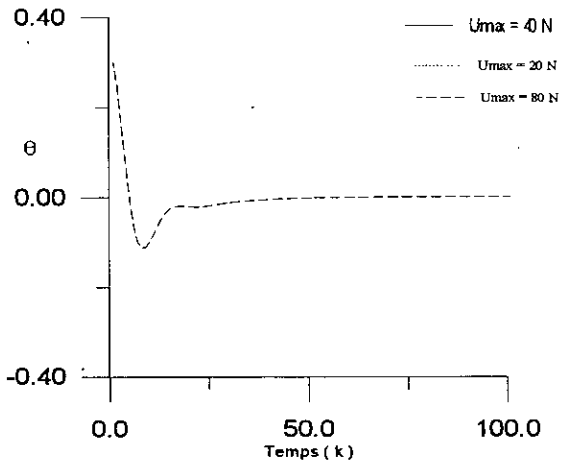


Figure 18: Test de la robustesse du contrôleur neuronal face à la variation de la valeur maximale de la commande

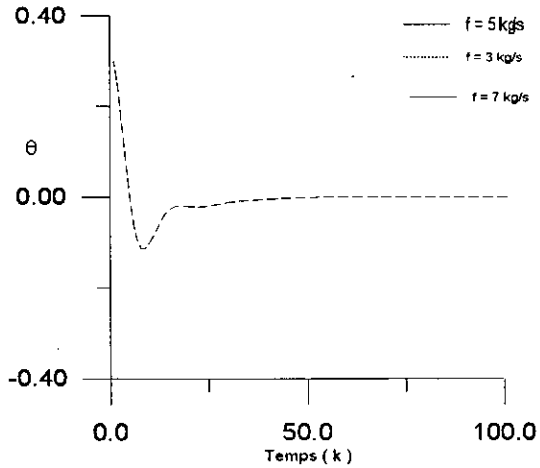


Figure 19 : Test de la robustesse du contrôleur neuronal face à la variation du coefficient de frottement f .

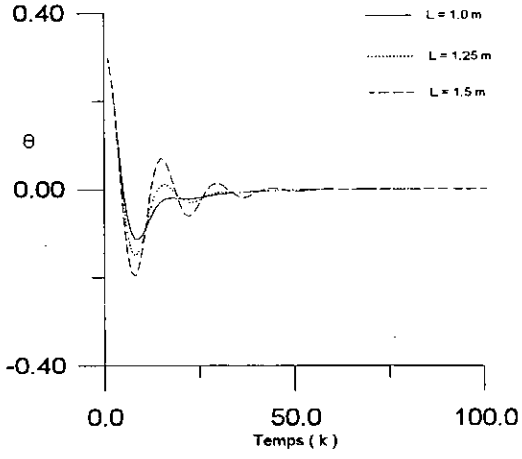


Figure 20 : Test de la robustesse du contrôleur neuronal face à la variation de la longueur du pendule .

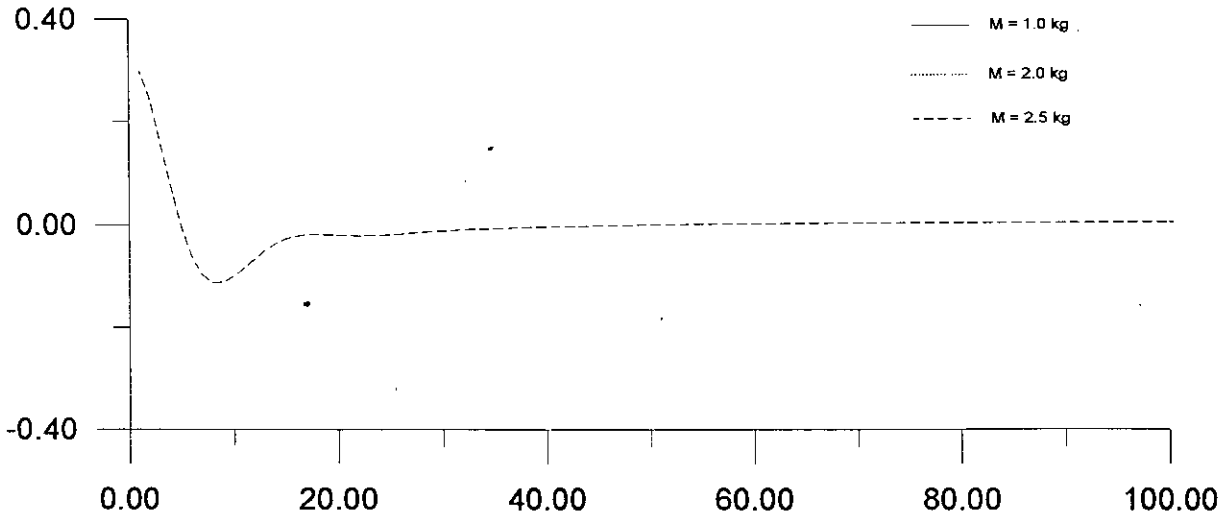


Figure 21 : Test de la robustesse du contrôleur face à la variation de la masse du chariot.

2. Commande Inverse :

a . Apprentissage Généralisé :

Dans cette partie, la commande inverse basée sur l'apprentissage généralisé a été appliquée au système :

$$\ddot{y}(t) = 2.\sin(y(t)) - 3y(t) - 1.6\dot{y}(t) + u(t) \qquad \text{(III-33)}$$

l'apprentissage d'un réseau statique appartenant à la classe $\eta_{6,20,20,1}$ a été effectuée en utilisant 350 points (le pas d'échantillonnage étant égal à 0.1 sec.), avec un signal d'excitation :

$$u(k) = 0.2\left[\sin(0.5\,r_k) + \cos(2\,r_k) + \sin^2(3\,r_k) + \cos^3(4\,r_k) + \cos(5\,r_k) + \sin(20\,r_k)\right] \qquad \text{(III-34)}$$

avec : $r_k = \pi/40 \cdot k$. L'adaptation c'est faite pendant 10000 itérations, en utilisant un pas variable. Aprés l'apprentissage un référence définie par :

$$r(t) = 0.7\left[\sin(2\pi \cdot 0.04\,t) + \sin(2\pi \cdot 0.07 \cdot t)\right] \qquad \text{(III-35)}$$

a été utilisée, la figure 22, montre les résultats de la commande. On remarque, que le système suit la référence.

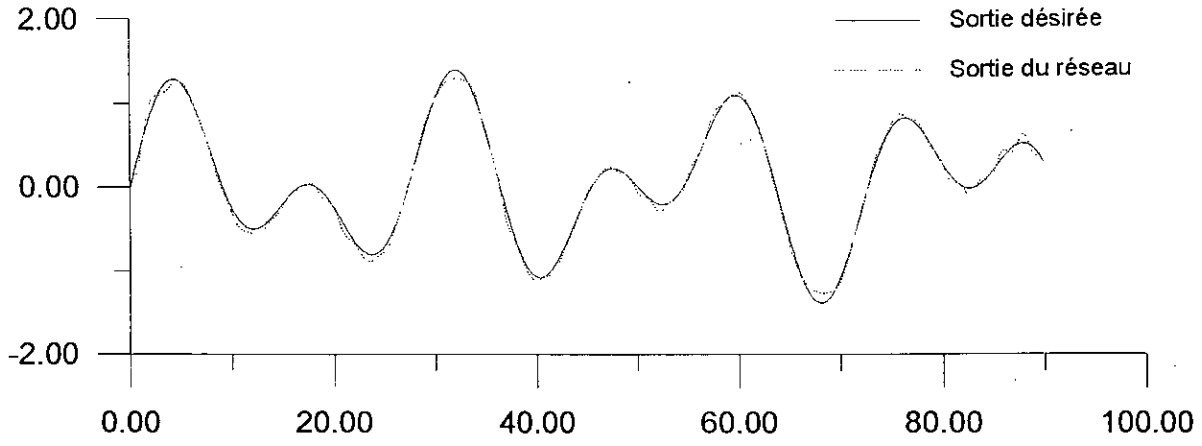


Figure 22 : Commande Inverse basée sur l'apprentissage généralisé.

b. Apprentissage Spécialisé :

Dans cette partie, la commande inverse basée sur l'apprentissage généralisé a été appliquée au système :

$$\ddot{y}(t) = 2.\sin(y(t)) - 3y(t) - 1.6\dot{y}(t) + u(t) \qquad \text{(III-34)}$$

l'apprentissage d'un réseau statique appartenant à la classe $\eta_{6,20,20,1}$ a été effectuée en démarrant avec le réseau précédent en utilisant 350 points (le pas d'échantillonnage étant égal à 0.1 sec.).
Le Jacobien a été défini par :

$$\frac{\partial y}{\partial u} = \frac{\Delta y}{\Delta u} = \frac{f(\bar{u} + 0.3) - f(\bar{u})}{0.3} \qquad \text{(III-35)}$$

L'adaptation c'est faite pendant 50000 itérations, en utilisant un pas variable. Aprés l'apprentissage un référence définie par :

$$r(t) = 0.7\left[\sin(2\pi \cdot 0.04\, t) + \sin(2\pi \cdot 0.07 \cdot t)\right] \qquad \text{(III-36)}$$

a été utilisée, la figure 23, montre les résultats de la commande. On remarque que le système suit la référence.

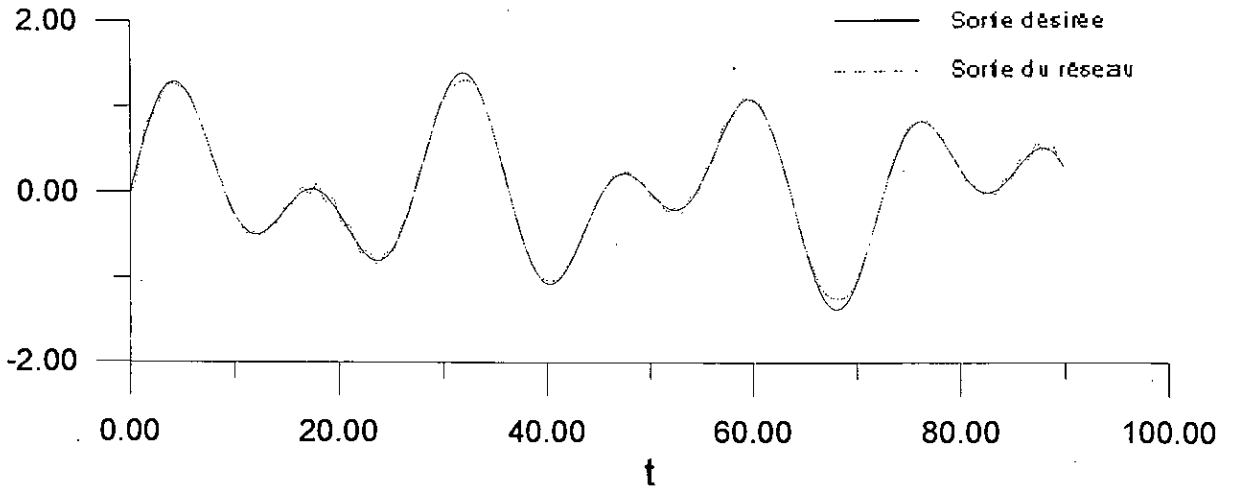


Fig. 23 : Commande Inverse Basée sur l'apprentissage spécialisé.

3. Commande Adaptative avec Modèle de Référence :

a. Modèle II :

Exemple 1:

Dans cet exemple, on considère le cas de la commande adaptative d'un système décrit par le modèle II.

$$y_p(k+1) = f\left(y_p(k)\right) + u(k) \qquad \text{(III-37)}$$

la fonction :

$$f\left(y(k)\right) = \frac{y(k)}{1+y(k)^2} \qquad \text{(III-38)}$$

est supposée inconnue. Le modèle de référence est donné par:

$$y_m(k+1) = 0.6\,y_m(k) + r(k) \qquad \text{(III-39)}$$

avec; r(k) est une référence bornée. Si on définit l'erreur de sortie $e_c(k)$, est définie par :

$$e_c(k) = y_p(k) - y_m(k) \qquad \text{(III-40)}$$

le but est d'avoir la commande bornée u(k) telle que l'erreur définie par (III-41) tende vers zéro. Si la fonction f(.) est connue, on obtient alors u(k) en utilisant les valeurs de $y_p(k)$ et de ses valeurs passées, elle est donnée par :

$$u(k) = -f\left(y_p(k)\right) + y_m(k+1) \qquad \text{(III-41)}$$

l'équation de l'erreur sera donc :

$$e_c(k+1)=0.6\ e_c(k) \qquad \text{(III-42)}$$

cette erreur converge asymptotiquement vers zéro, puisque le modèle de référence est stable. Mais, puisque f(.) est inconnue, elle est estimée en temps réel en utilisant un réseau de neurones (voir chapitre II). La commande sera donc égale à :

$$u(k) = -\eta\left(y_p(k)\right) + y_m(k+1) \qquad \text{(III-43)}$$

Le système en boucle fermée sera donc :

$$y_p(k+1) = f\left(y_p(k)\right) - \eta\left(y_p(k)\right) + y_m(k+1) \qquad \text{(III-44)}$$

Dans notre cas, nous avons utilisé un réseau récurrent en entrée (parce que ces réseaux sont plus robustes), et l'adaptation c'est faite en temps réel, en utilisant une référence :

$$r(k) = \sin\left(2\pi k/10\right) + \sin\left(2\pi k/25\right) \qquad \text{(III-45)}$$

Les figures 24 et 25 montrent, l'évolution de l'erreur lors de l'apprentissage d'un réseau appartenant à la classe $\eta_{2,20,10,1}$, et les sorties du système et la référence. On remarque que l'erreur diminue rapidement; on a poussé l'apprentissage malgré les bonnes performances, pour vérifier la stabilité de l'algorithme d'adaptation. Cependant, nous devrions noter que le choix du pas d'apprentissage est très important lors de la conception d'une commande neuronale en temps réel; le pas doit être faible, car sinon le système en boucle fermée risque d'être instable.

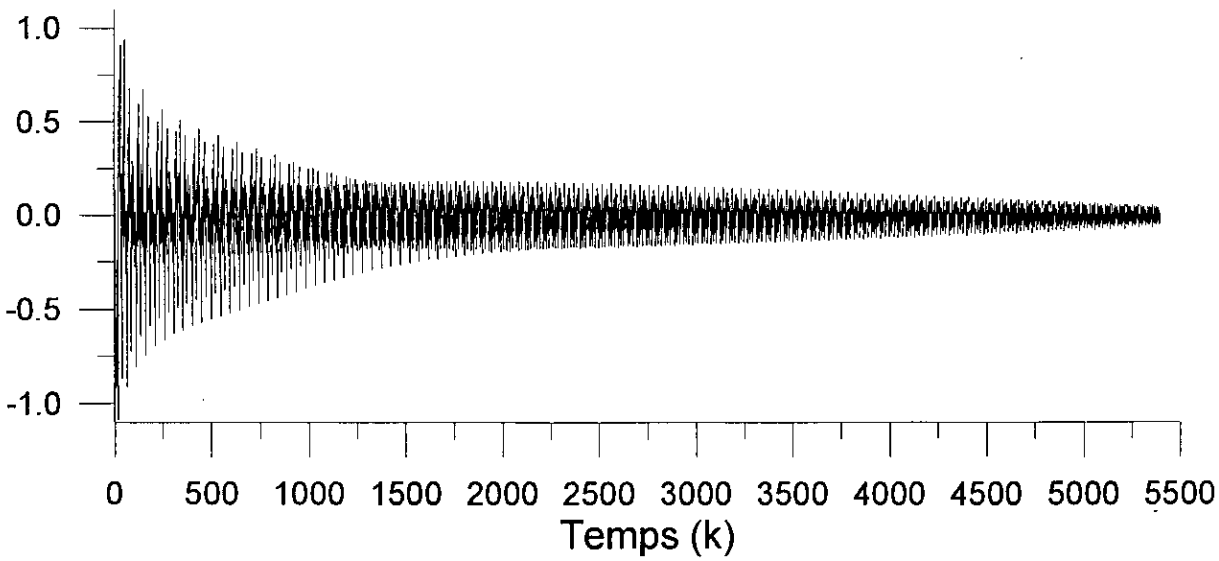


Figure 24: Evolution de l'erreur lors de l'apprentissage en temps réel

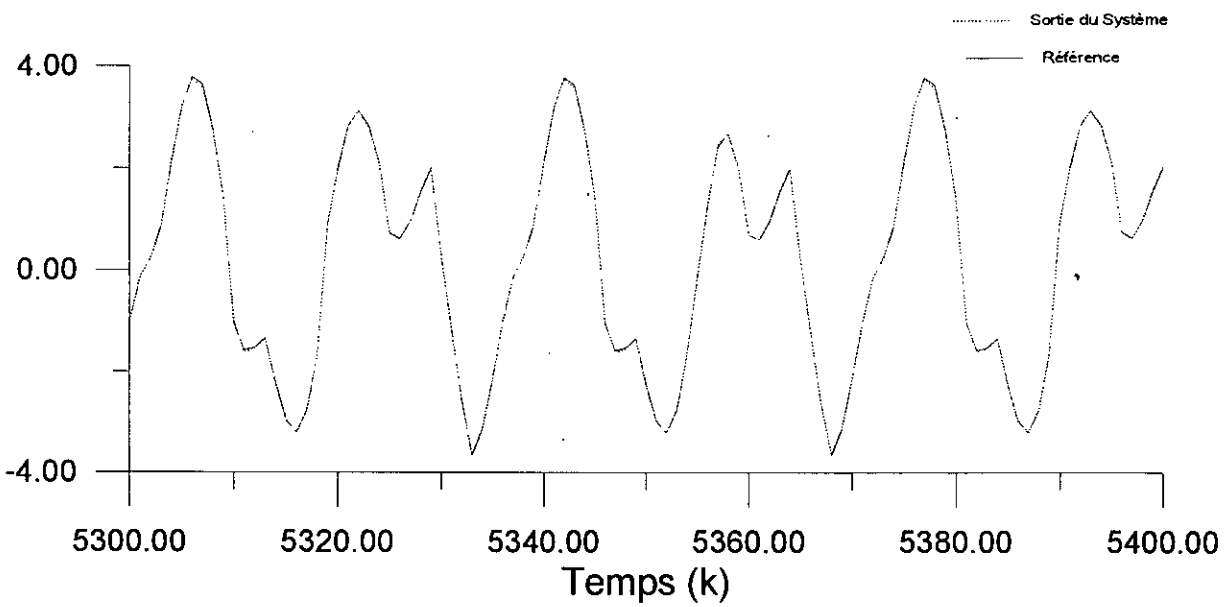


Figure 25 : Référence et Sortie du Système.

Exemple 2:

Dans ce cas, la commande adaptative est appliquée au système suivant :

$$y(k+1) = \frac{\cos(y(k))}{1+\sin^2(y(k))} + \sin(y(k))\, e^{-y^2(k)} + \frac{y(k)}{1+y(k)^2} + u(k) \qquad \text{(III-46)}$$

Le même modèle de référence que l'exemple 1, est utilisée. Les figures 26 et 27 montrent respectivement l'évolution de l'erreur durant l'apprentissage d'un réseau récurrent en entrée appartenant à la classe $\eta_{2,20,10,1}$, et les sorties du système et du modèle de référence.

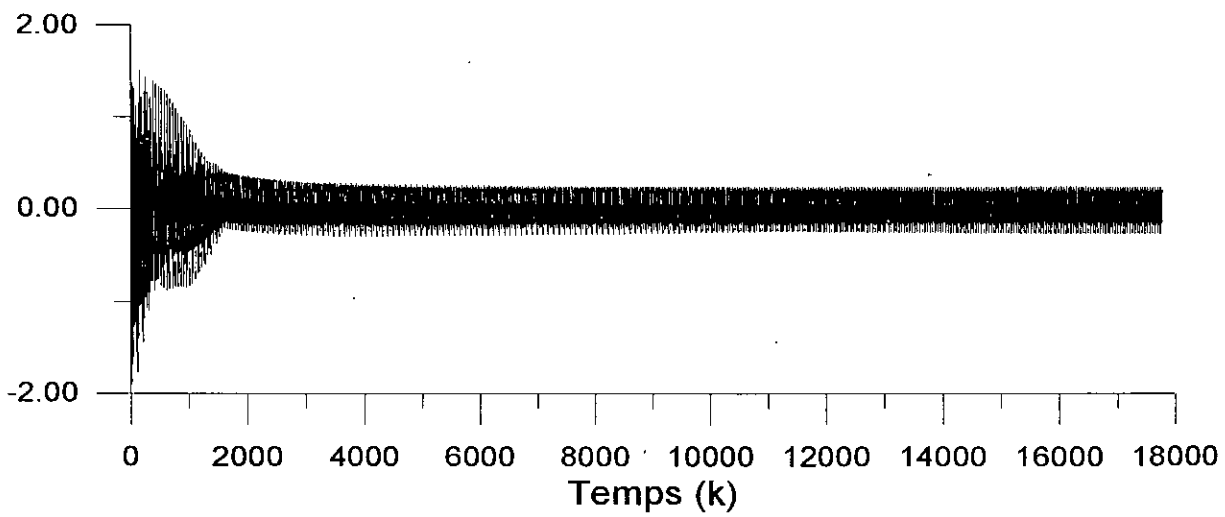


Figure 26 : Evolution de l'erreur lors de l'apprentissage en temps réel.

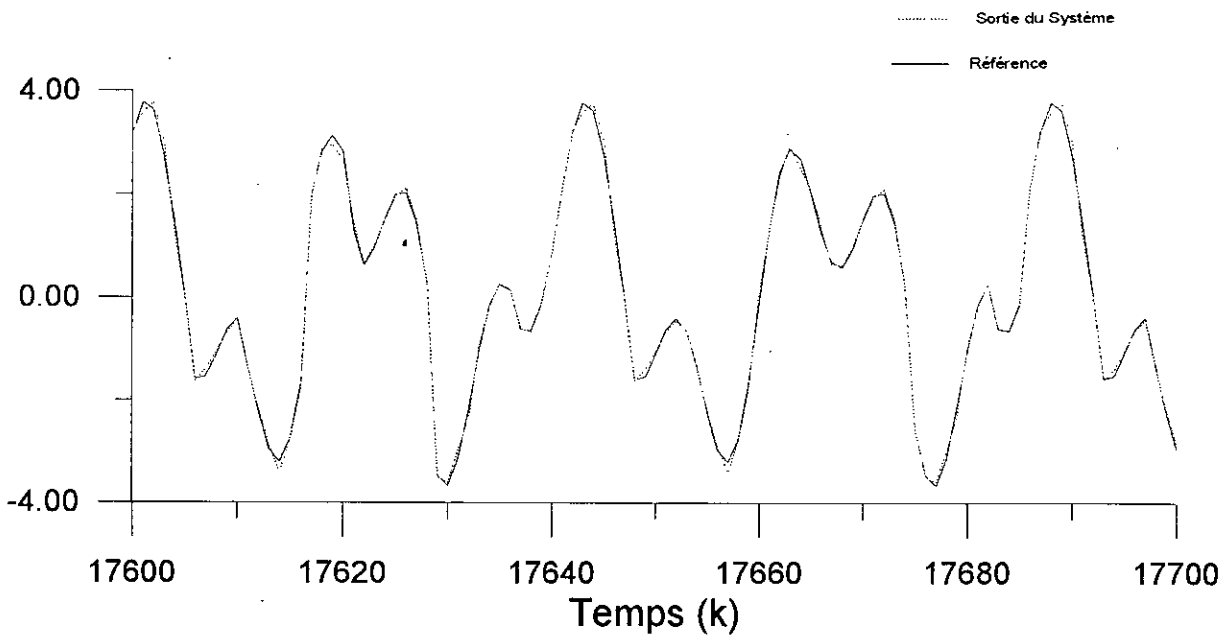


Figure 27 : Référence et Sortie du Système.

Exemple 3 :

La méthode précédente peut être facilement étendue aux systèmes MIMO. Considérons le système :

$$\begin{bmatrix} y_{p1}(k+1) \\ y_{p2}(k+1) \end{bmatrix} = \begin{bmatrix} \dfrac{y_{p1}(k)}{1+y_{p2}(k)} \\ \dfrac{y_{p1}(k)y_{p2}(k)}{1+y^2_{p2}(k)} \end{bmatrix} + \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \qquad \text{(III-47)}$$

Le modèle de référence utilisé est défini par :

$$\begin{bmatrix} y_{m_1}(k+1) \\ y_{m_2}(k+1) \end{bmatrix} = \begin{bmatrix} 0.6 & 0.2 \\ 0.1 & -0.8 \end{bmatrix} \begin{bmatrix} y_{m_1}(k) \\ y_{m_2}(k) \end{bmatrix} + \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} \qquad \text{(III-48)}$$

Les réseaux utilisés appartiennent à la classe $\eta_{2,30,20,1}$. Les figures 28 et 29 montrent les sorties, après un apprentissage en temps réel d'un réseau récurrent en entrée. Les signaux de référence sont $r_1(k) = r_2(k) = \sin\left(2\pi k/25\right)$.

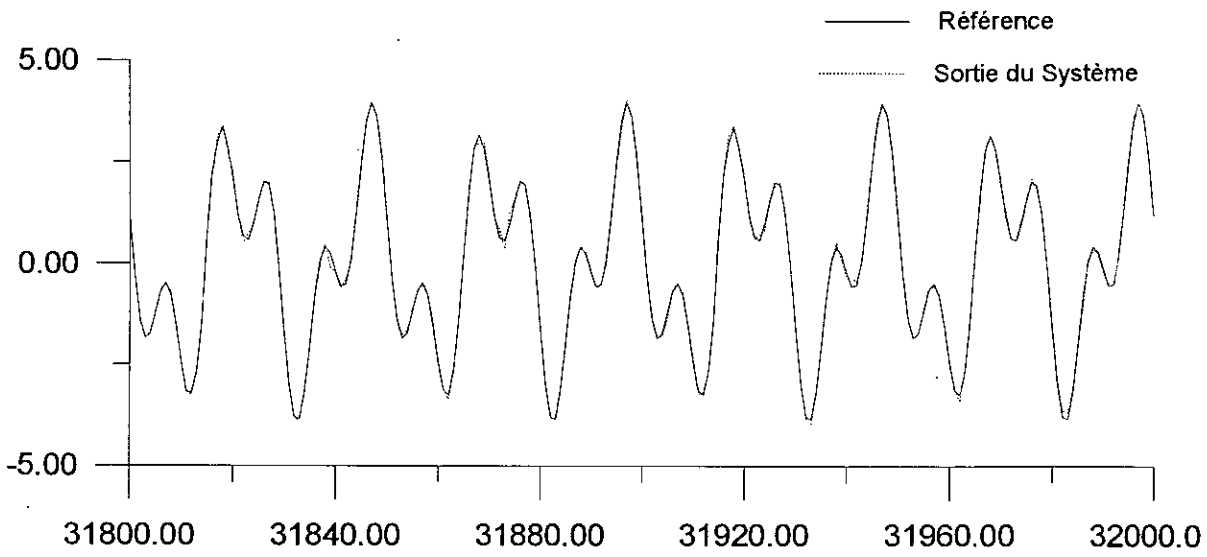


Figure 28 : Sortie y_{p1} et référence y_{m1}.

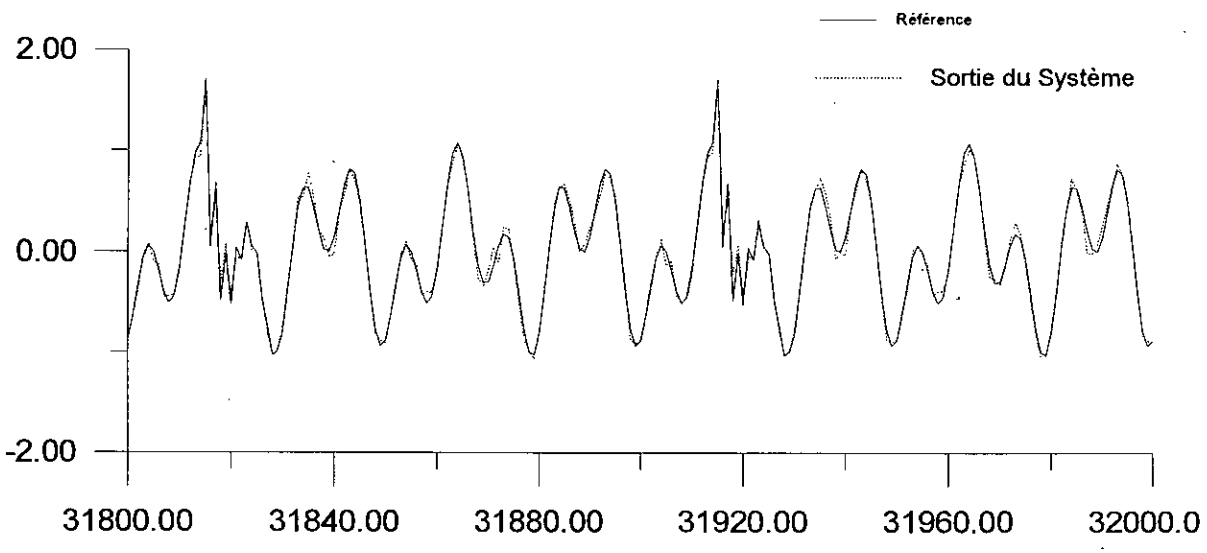


Figure 29: Sortie y_{p2} et référence y_{m2}.

b. Modèle III

Dans cet exemple, on considère la commande adaptative d'un système décrit par le modèle III. i.e. :

$$y_p(k+1) = f\left[y_p(k)\right] + g[u(k)] \qquad \text{(III-47)}$$

les fonctions $f\left[y_p(k)\right] = \dfrac{y_p(k)}{1+y_p^2(k)}$ et $g[u(k)] = u^3(k)$, sont supposées inconnues. En choisissant le modèle de référence :

$$y_m(k+1) = 0.5\sin\left(2\pi k/25\right) + 0.5\sin\left(2\pi k/10\right) \qquad \text{(III-48)}$$

le but est de déterminer u(k) telle que : $\lim_{k\to\infty}\left|y_p(k) - y_m(k)\right| = 0$. La commande u(k) sera calculée à partir de :

$$u(k) = \eta_{g^{-1}}\left[-\eta_f\left[y_p(k)\right] + y_m(k+1)\right] \qquad \text{(III-49)}$$

avec: $\eta_{g^{-1}}(\cdot)$ est un réseau qui approxime la fonction $g^{-1}(.)$, $\eta_f(\cdot)$ est un réseau qui approxime la fonction f(.). Les estimées de f et g sont les réseaux de neurones η_f et η_g, le réseau $\eta_{g^{-1}}$ est entraîné de telle façon à avoir $\eta_g\left[\eta_{g^{-1}}(r)\right] \approx r$.Dans notre cas, les réseaux utilisés sont des réseaux récurrents en entrée et appartiennent à la classe $\eta_{1,20,15,1}$ leurs apprentissage c'est fait hors-ligne; La figure 30 montre la sortie du système et la référence, et ont été ensuite utilisés pour implanter la commande.

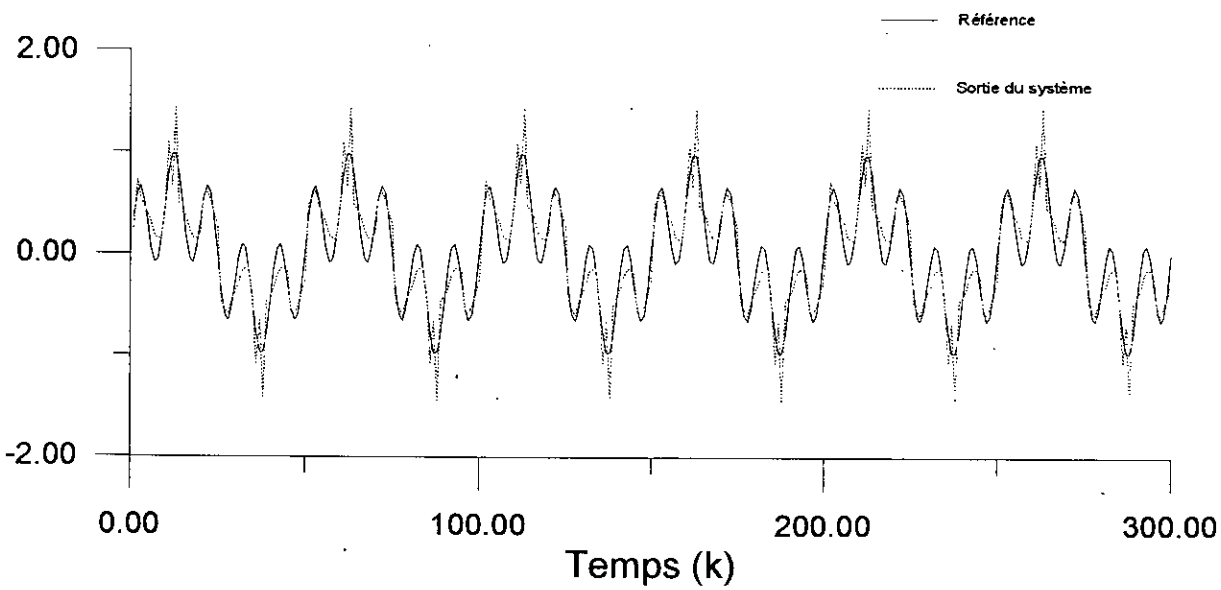


Figure 30 : Sortie du système et référence.

Exemple 2:

Considérons le système régit par l'équation :

$$y_p(k+1) = f\left[y_p(k)\right] + g\left[u(k)\right] \qquad \text{(III-50)}$$

les fonctions $f\left[y_p(k)\right] = \dfrac{y_p(k)}{1+y_p^2(k)}$ et $g\left[u(k)\right] = \cos\left(u(k)\right)$, sont supposées inconnues. En choisissant le modèle de référence :

$$y_m(k+1) = 0.5\sin\left(2\pi k/25\right) + 0.5\sin\left(2\pi k/10\right) \qquad \text{(III-51)}$$

Dans ce cas les réseaux appartiennent à la classe $\eta_{1,20,15,1}$, leurs apprentissage c'est fait hors-ligne, et ensuite ils ont été utilisés pour la synthèse de la commande. La figure 31 montre le résultat de la commande.

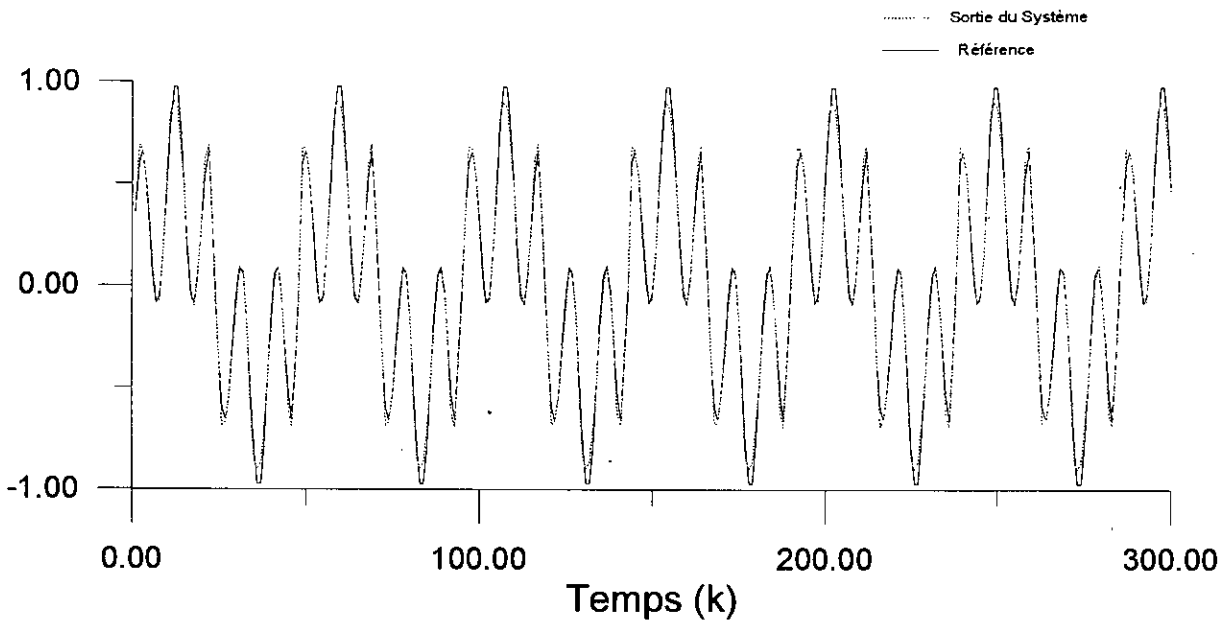


Figure 31 : Sortie du Système et Référence.

4. Rejet de Perturbations :

a. Modèle de perturbation linéaire :

Considérons le système avec une perturbation additive, le système global est régit par les équations aux différences suivantes :

$$\begin{aligned} y(k+1) &= f\left[y(k),\ldots,y(k-n+1)\right] + \sum_{j=0}^{n-1}\beta_j\left[u(k-j) + v(k-j)\right] \\ v(k+1) &= \sum_{i=0}^{p-1}\gamma_i v(k-i) \end{aligned} \qquad \text{(III-52)}$$

Cette équation est équivalente :

$$\begin{aligned} y(k+1) &= f\left[Y(k)\right] + D(z)\left[u(k) + v(k)\right] \\ v(k+1) &= R(z)\,v(k) \end{aligned} \qquad \text{(III-53)}$$

avec u(k) et y(k) sont respectivement l'entrée et la sortie du système. La fonction f(.) est supposée inconnue. Les entiers n et p sont supposés connus.

Les coefficients $\beta_j\,(j=1,\cdots,n-1)$ et $\gamma_i\,(i=0,\ldots,p-1)$ sont inconnus.

Dans ce cas, le système global peut être représenté sans perturbation en éliminant v(k) de (III-53). En notant que :

$$D(z)\,v(k-i) = y(k-i+1) - f[Y(k-i)] - D(z)u(k-i) \qquad \text{(III-54)}$$

En notant que (III-53) peut être mise sous la forme:

$$y(k+1) = f[Y(k)] + D(z)u(k) + \sum_{i=1}^{p}\gamma_{i-1}D(z)v(k-i) \qquad \text{(III-55)}$$

Le système global sera donc :

$$y(k+1) = \bar{f}[\overline{Y}(k)] + \overline{D}(z)u(k) \qquad \text{(III-56)}$$

avec :

$$\begin{aligned}
&\overline{Y}(k) = \left[y(k), y(k-1), \ldots, y(k-p-n+1)\right]^T \\
&\bar{f}(\overline{Y}(k)) = f[Y(k)] + \sum_{i=1}^{p}\gamma_{i-1}\left[y(k-i+1) - f[Y(k-i)]\right] \qquad \text{(III-57)} \\
&\overline{D}(z) = D(z)R_1(z) = D(z)\left[1 - z^{-1}R(z)\right] = \sum_{i=0}^{n+p-1}\bar{\beta}_i z^{-i}.
\end{aligned}$$

On remarque que le système global (III-56) reste toujours linéaire par rapport la commande. On remarque aussi que le nombre de décalages des entrées et des sorties a augmenté de p, par rapport au système sans perturbations [Mukhopadhyay and Narendra, 1993].

Exemple :

Considérons le système non-linéaire décrit par l'équation aux différences suivante :

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u(k) + v(k) \qquad \text{(III-58)}$$

La perturbation est sinusoïdale, elle est décrite par l'équation aux différences suivante:

$$\begin{aligned}
v_1(k+1) &= \cos(\psi)v_1(k) + \sin(\psi)v_2(k) \\
v_2(k+1) &= -\sin(\psi)v_1(k) + \cos(\psi)v_2(k) \qquad \text{(III-59)} \\
v(k) &= v_1(k)
\end{aligned}$$

avec $2\pi/\psi$ est la période de la sinusoïde. Dans la simulation on prendra $\psi = \pi/6$ et $[v_{10}, v_{20}] = [1,0]$ comme conditions initiales du système qui génère la perturbation. En éliminant v(k) de (III-58) en utilisant (III-59) on obtient le système :

$$y(k+1) = \gamma_0 y(k) + \gamma_1 y(k-1) + f[y(k)] - \gamma_0 f[y(k-1)] - \gamma_1 f[y(k-2)] + u(k) - \gamma_0 u(k-1) - \gamma_1 u(k-2) \quad \text{(III-60)}$$

C'est donc le même problème rencontré lors de la commande adaptative d'un système qui est sous la forme du Modèle II. Un réseau de neurones récurrent en entrée appartenant la classe $\eta_{5,20,10,1}$ a été entraîné en temps réel. La référence utilisée est :

$$y_m(k+1) = 0.6 y_m(k) + r(k) \quad \text{(III-61)}$$

Le signal de référence est $r(k) = 0.25\sin(2\pi k/10) + 0.25\sin(2\pi k/25)$

La commande u(k) a été calculée en utilisant l'équation:

$$u(k) = y_m(k+1) - \eta[y(k), y(k-1), y(k-2), u(k-1), u(k-2)] \quad \text{(III-62)}$$

Les figures 32 ,33, 34, montrent l'évolution de l'erreur lors de la commande sans rejet de perturbations, l'évolution de l'erreur lors de l'apprentissage et la sortie du système ainsi que la référence quand le rejet de perturbations est appliqué. On remarque que le rejet de perturbations est efficace, et que l'erreur a été divisée par 10..

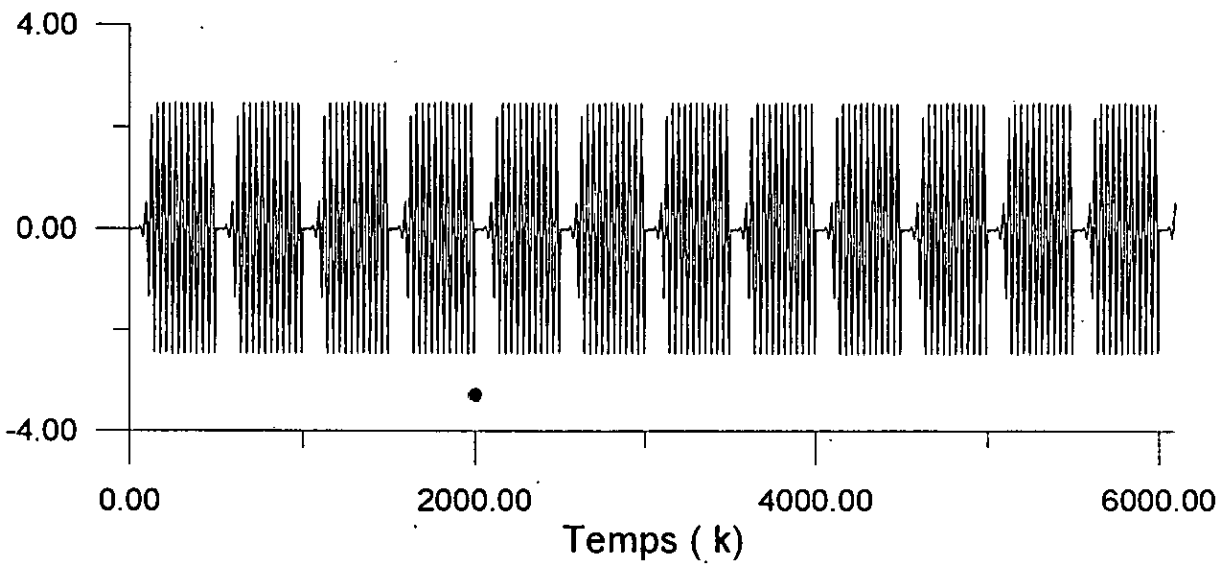


Figure 32 : Evolution de l'erreur lors de la commande adaptative sans rejet de perturbations.

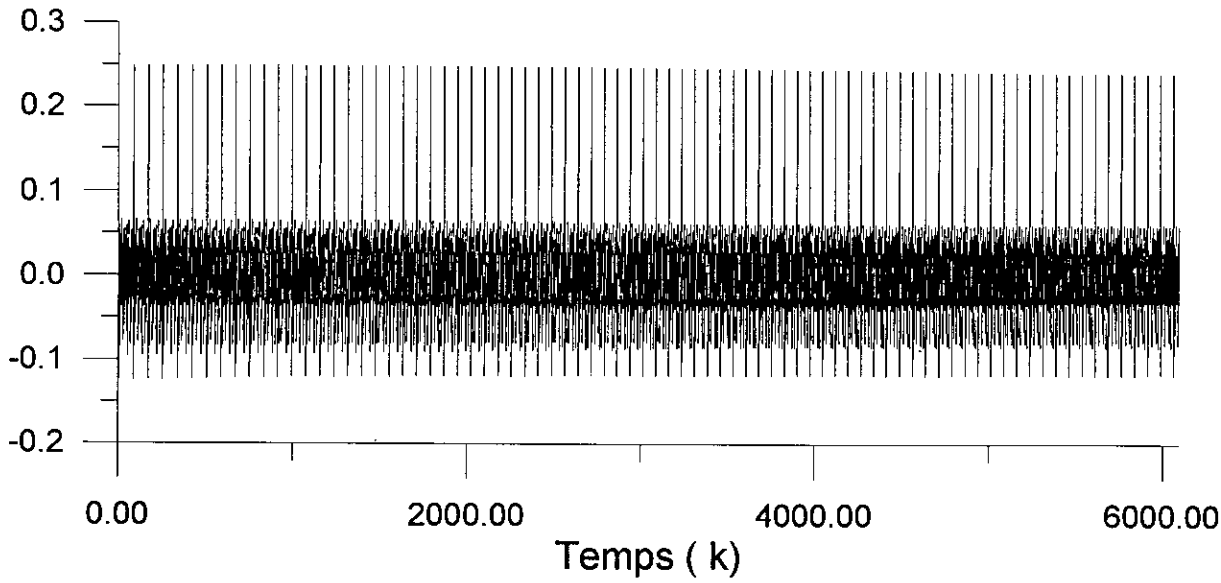


Figure 33 : Evolution de l'erreur lors du rejet de perturbations

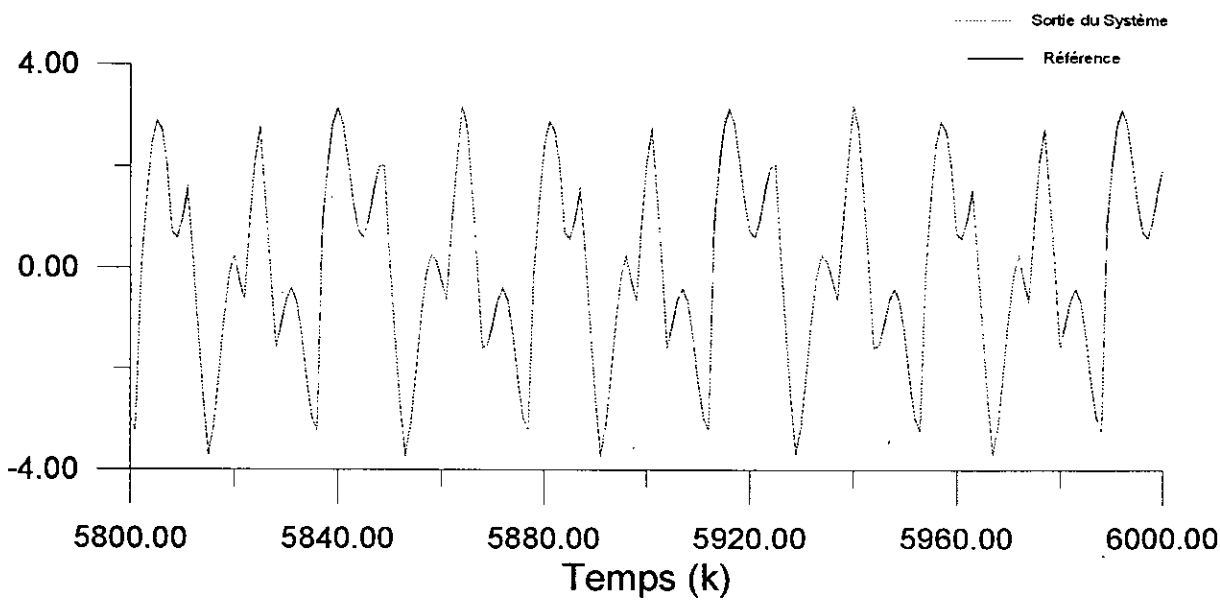


Figure 34 : Référence et sortie du système lors de la commande adaptative avec rejet de perturbations.

b. Modèle de perturbation non-linéaire :

Dans la partie précédente, on a parlé du cas du rejet de perturbations, dans le cas ou les modèles de celles-ci est linéaire. Dans cette partie, on considère un cas plus complexe ou le modèle des perturbations est non-linéaire, la perturbation est ainsi modélisée comme étant la sortie d'un système non-linéaire, stable, et non-forcé. Plusieurs modèles non-linéaires des perturbations existent, on peut citer, l'équation de van der Pol ou l'équation de Duffing [Parker and Chua, 1987].

Considérons un système non-linéaire, qui lorsque les perturbations sont absentes *dépend linéairement de la valeur présente de la commande,* cependant, la discussion ci-dessous peut être étendue en y incluant les valeurs passées de la commande.
Le système global est régit par les équations aux différences suivantes:

$$y(k+1) = f[Y(k)] + u(k) + v(k) \quad \text{(III-63)}$$
$$v(k+1) = g[v(k),\ldots,v(k-p+1)] \quad \text{(III-64)}$$

le système de l'équation (III-64) doit être stable.

En éliminant v(k) de (III-63) en utilisant (III-64) on obtient :

$$y(k+1) = \bar{f}[\overline{Y}(k),\overline{U}(k-1)] + u(k) \quad \text{(III-65)}$$

avec :

$$\overline{Y}(k) = [y(k),\cdots,y(k-n-p-1)]^T$$
$$\overline{U}(k-1) = [u(k-1),\cdots,u(k-p)]^T.$$

Le système de l'équation (III-65) est toujours linéaire par rapport la commande. Une procédure similaire celle utilisée précédemment, doit être utilisée pour l'identification du système global.

Exemple :

Considérons le système non-linéaire d'ordre un :

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u(k) + v(k) \quad \text{(III-66)}$$

la perturbation est régie par la version discrète de l'équation de van der Pol, i.e.:

$$x_{v1}(k+1) = x_{v1}(k) + 0.2\,x_{v2}(k)$$
$$x_{v2}(k+1) = -0.2\,x_{v1}(k) + x_{v1}(k) - 0.1(x_{v1}^2(k)-1)\,x_{v1}(k)$$
$$v(k) = x_{v1}(k) \quad \text{(III-67)}$$

La figure 34. schématise la trajectoire de x_{v2} en fonction de x_{v1} , ainsi que v(k) en fonction du temps.

Le modèle d'identification en présence de la perturbation est :

$$\hat{y}(k+1) = \eta[\hat{y}(k),\cdots,\hat{y}(k-q-1),u(k-1),\cdots,u(k-q)] + u(k) \quad \text{(III-68)}$$

avec q étant un entier qui doit être supérieur 1.

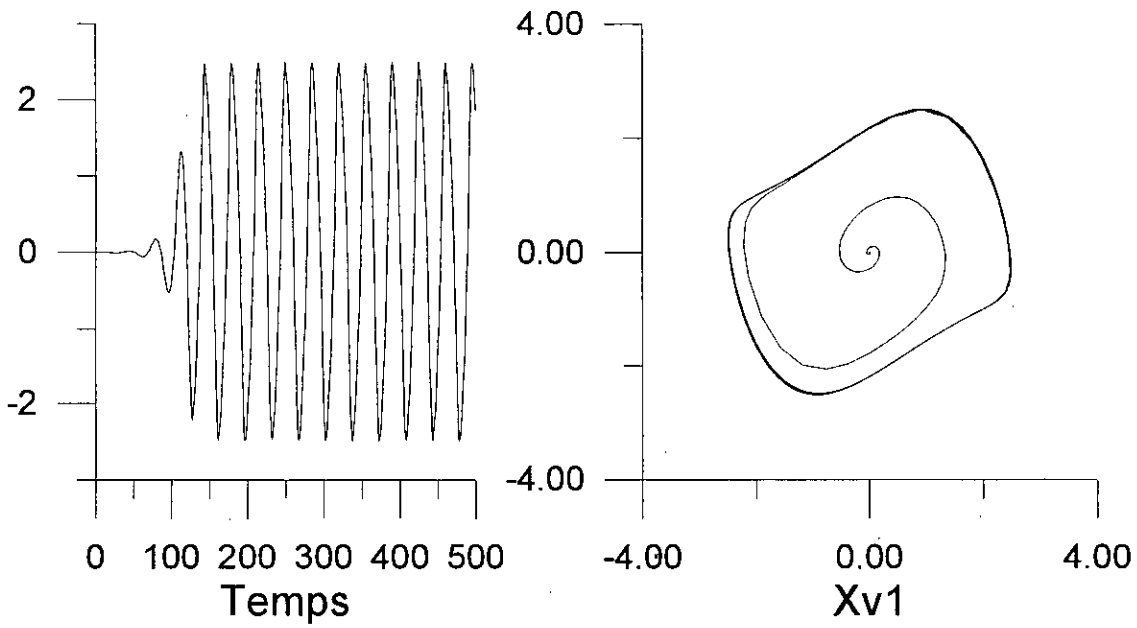


Figure 34 : Caractéristiques de la perturbation.

L'objectif est de suivre la référence définie par :

$$y_m(k+1) = 0.5\sin\left(2\pi k/25\right) + 0.5\sin\left(2\pi k/10\right) \qquad \text{(III-69)}$$

La commande a été calculée en utilisant :

$$u(k) = y_m(k+1) - \eta\left(\hat{y}(k), \hat{y}(k-1), \hat{y}(k-2), u(k), u(k-1)\right) \qquad \text{(III-70)}$$

Un réseau de neurones récurrent en entrée appartenant la classe $\eta_{5,40,20,1}$, a été entraîné en temps réel, pour permettre l'identification par le modèle (III-68). Les figures 35, 36 et 37, montrent respectivement la réponse du système dans le cas ou la commande adaptative sans rejet de perturbations a été utilisée, l'évolution de l'erreur, et les réponses du système quand la commande adaptative avec rejet a été utilisée. On remarque que les sorties du système et la référence sont indiscernables.

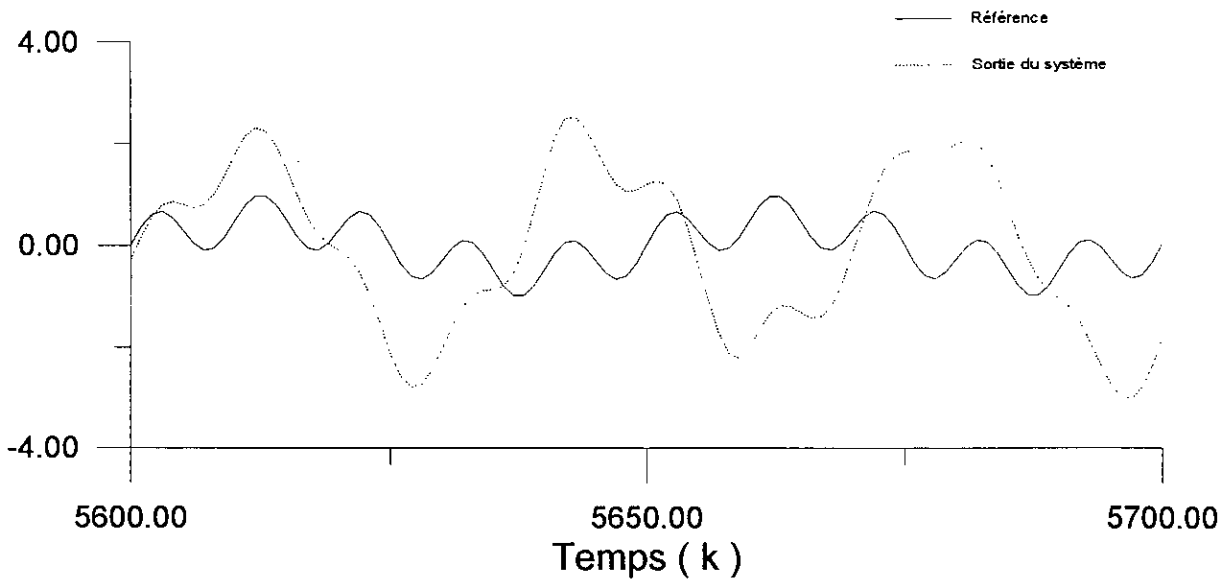


Figure 35 : Commande Adaptative sans rejet de perturbations

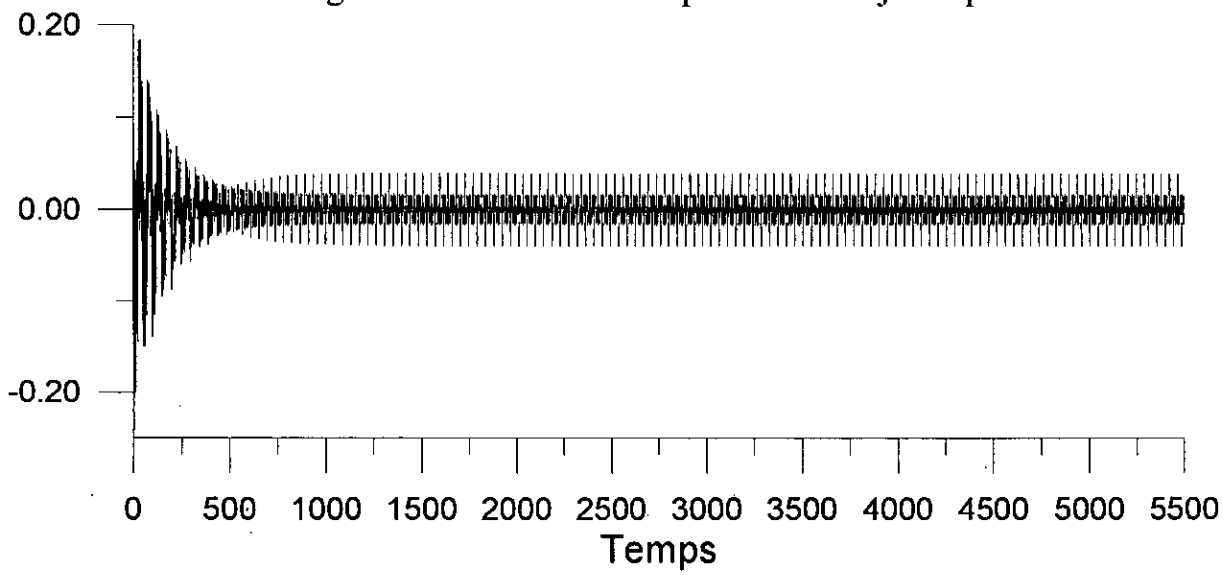


Figure 36 : Evolution de l'erreur lors de l'apprentissage en temps réel.

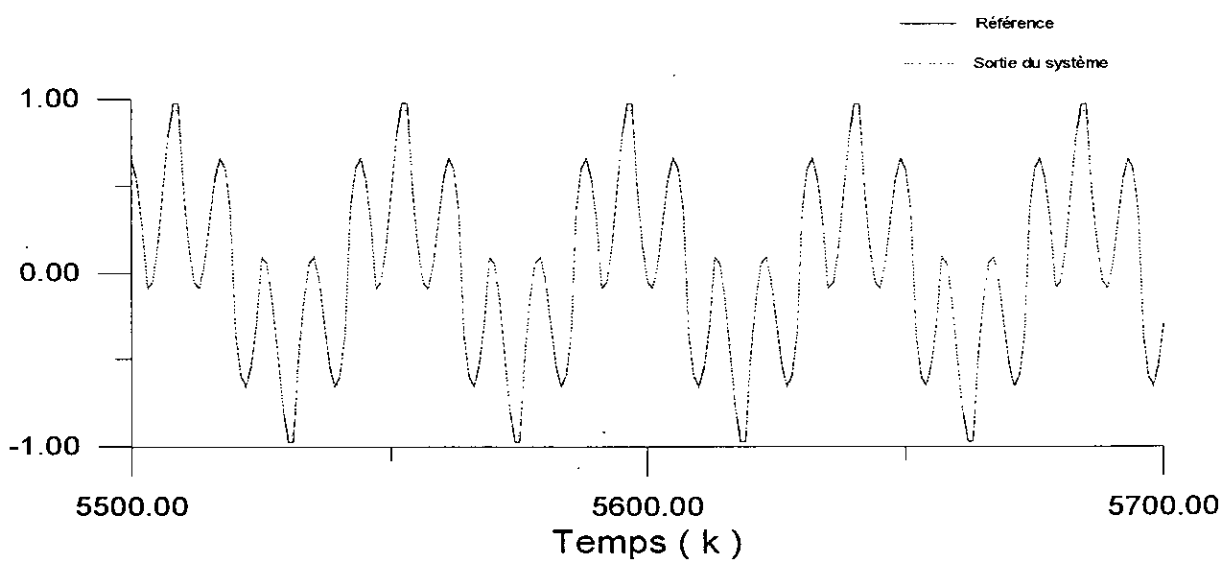


Figure 37 : Résultats de la commande adaptative avec rejet de perturbations.

5. Commande par bouclage linéarisant :

Dans cette partie on va appliquer la méthode du bouclage linéarisant un bras de robot du type PUMA 560 (voir Fig. 38), son modèle est donné par [Seraji, 1989]:

$$T = M(\theta)\ddot{\theta} + N(\theta,\dot{\theta}) + G(\theta) + H(\dot{\theta}) \qquad \text{(III-71)}$$

avec :

$$M(\theta) = \begin{bmatrix} a_1 + a_2\cos(\theta_2) & a_3 + \frac{a_2}{2}\cos(\theta_2) \\ a_3 + \frac{a_2}{2}\cos(\theta_2) & a_3 \end{bmatrix}$$

$$N(\theta,\dot{\theta}) = \begin{bmatrix} -(a_2\sin(\theta_2))\left(\dot{\theta}_1\dot{\theta}_2 + \frac{\dot{\theta}_2^2}{2}\right) \\ (a_2\sin(\theta_2))\frac{\dot{\theta}_1^2}{2} \end{bmatrix}$$

$$G(\theta) = \begin{bmatrix} a_4\cos(\theta_1) + a_5\cos(\theta_1+\theta_1) \\ a_5\cos(\theta_1+\theta_1) \end{bmatrix}$$

$$H(\dot{\theta}) = \begin{bmatrix} V_1\dot{\theta}_1 + V_2\,\mathrm{sgn}(\dot{\theta}_1) \\ V_3\dot{\theta}_1 + V_4\,\mathrm{sgn}(\dot{\theta}_1) \end{bmatrix}$$

$$g = \begin{bmatrix} 0 \\ 9.81 \end{bmatrix}$$

Dans les expressions précédentes, $a_1,\ldots,a_5$ sont des constantes qui dépendent des masses (m_1,m_2) et des longueurs (l_1,l_2) des deux articulations, (V_1,V_3) et (V_2,V_4) représentent respectivement les coefficients des frottements visqueux et des frottements de Coulomb. Lors des simulations on utilisera les paramètres suivants : $m_1 = 15.91$ kg , $m_2 = 11.36$ kg; $l_1 = l_2 = 0.432$ m; dans ce cas on aura :

$$a_1 = 3.82\ ,\ a_2 = 2.12\ ,\ a_3 = 0.71\ ,\ a_4 = 81.82\ ,\ a_5 = 24.06.$$

Les coefficients de frottements utilisés sont :

$$V_1 = V_3 = 1.0\ \text{Nt.m/rad.s}^{-1} \text{ et } V_2 = V_4 = 0.5\ \text{Nt.m.}$$

Dans ce cas nous allons supposer que $F = H(\dot{\theta})$; les trajectoires désirées sont définies par :

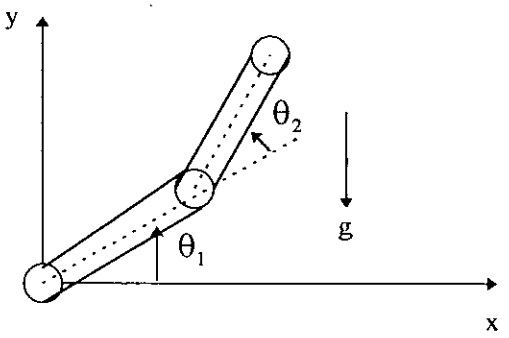


Figure 38 : Bras de robot du type PUMA 560.

$$\theta_{r1}(t) = \begin{cases} -\frac{\pi}{2} + \frac{1}{4}\left[\frac{2\pi t}{3} - \sin\left(\frac{2\pi t}{3}\right)\right] \text{rad}, & 0 \le t \le 3 \\ 0 & 3 < t \end{cases}$$

$$\theta_{r2}(t) = \begin{cases} \frac{1}{4}\left[\frac{2\pi t}{3} - \sin\left(\frac{2\pi t}{3}\right)\right] \text{rad}, & 0 \le t \le 3 \\ \frac{\pi}{2} & 3 < t \end{cases}$$

La commande a été synthétisée comme cité précédemment en utilisant un réseau récurrent en entrée, appartenant la classe $\eta_{4,20,2}$ a été entraîné pendant 50000 itérations, en utilisant des pas de 0.01 pour la seconde et la troisième couche, le pas d'échantillonnage a été choisi comme égal 1 ms. Les figures 39 et 40 montrent les résultats des simulations.

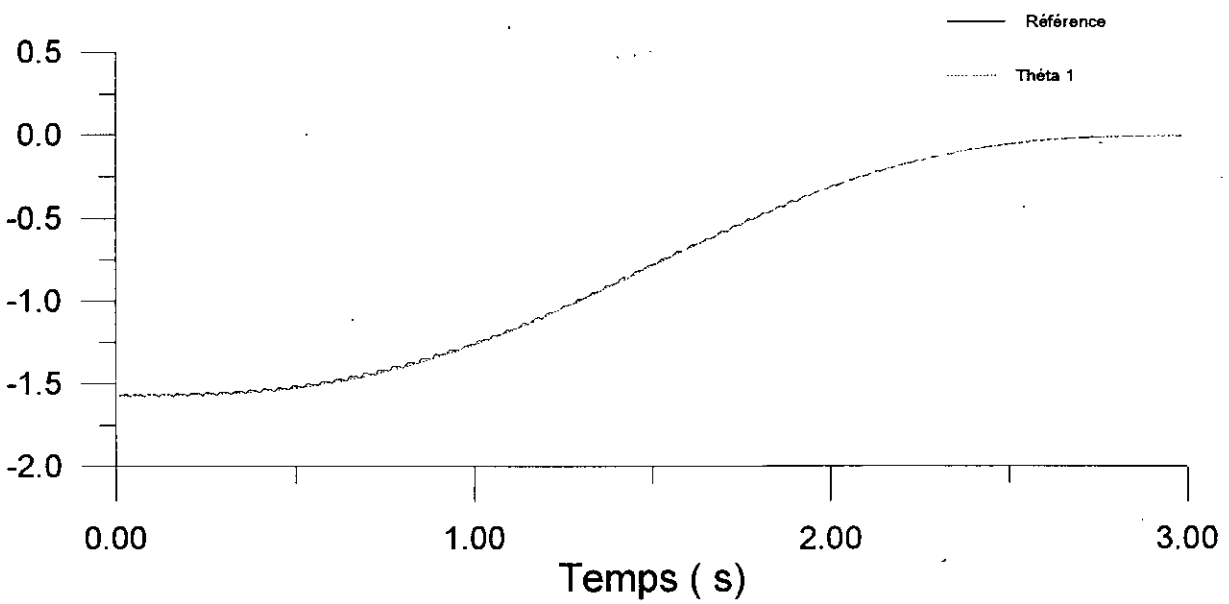


Figure 39 : Position de la première articulation et référence lors de l'utilisation de la commande par bouclage linéarisant.

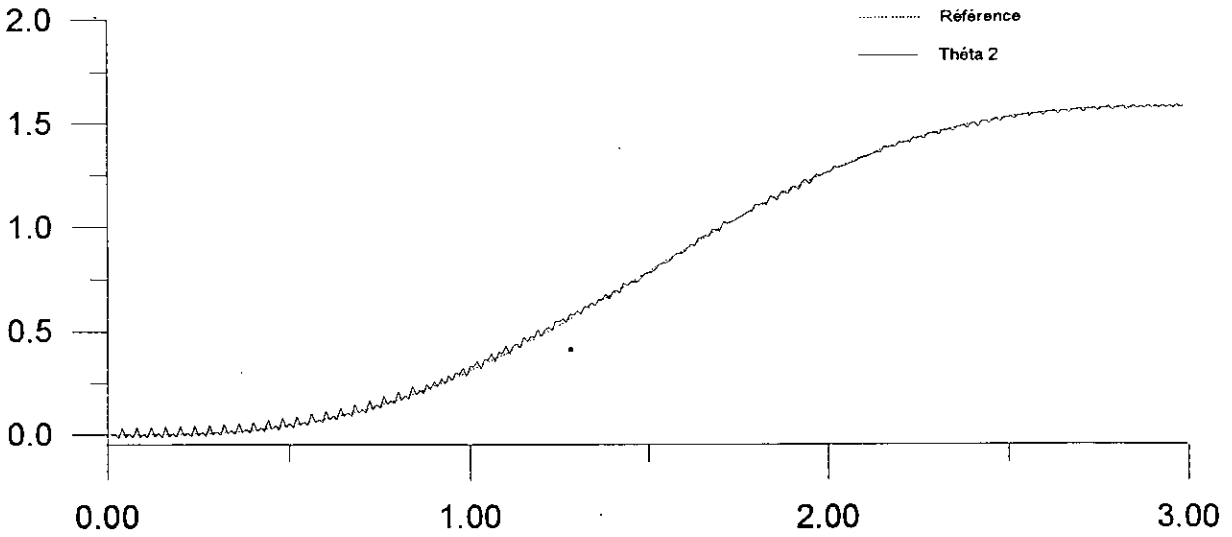


Figure 40 : Position de la deuxième articulation et référence lors de l'utilisation de la commande par bouclage linéarisant.

On remarque que les articulations suivent les références, on en déduit que le réseau de neurones a appris contrôler le bras manipulateur, sans avoir " eu connaissance " de la dynamique non-modélisée, représentée par les frottements.

6. Régulateur Auto-ajustable :

Dans cette partie un régulateur auto-ajustable est synthétisé pour commander le système régit par l'équation aux différences :

$$y(k+1) = 1.1\sin\left[\cos\left[y(k)\right]\right] + 1.5u(k) \qquad \text{(III-72)}$$

Pour réaliser ce contrôleur, un réseau dynamique entraîné par l'algorithme "Fixed Point Learning" dans sa version qui minimise l'erreur instantanée a été utilisé. Il était constitué par 2 neurones dans la première couche, 3 neurones dans la couche cachée, et un sortie. Son apprentissage a été effectué pendant 20 cycles, chaque cycle correspondant un passage sur 100 exemples. Des pas variables ont été utilisés, dont les valeurs initiales étaient 0.05, 0.9, 0.25. La figure 41 montre le résultat après la phase d'apprentissage; lorsqu'une référence définie par :

$$r(k) = \begin{cases} \sin\left(2\pi k/50\right) & k < 50 \\ 1 & 50 \leq k < 75 \\ -1 & k \geq 75 \end{cases} \qquad \text{(III-73)}$$

On remarque que la référence et la sortie du système sont indiscernables.

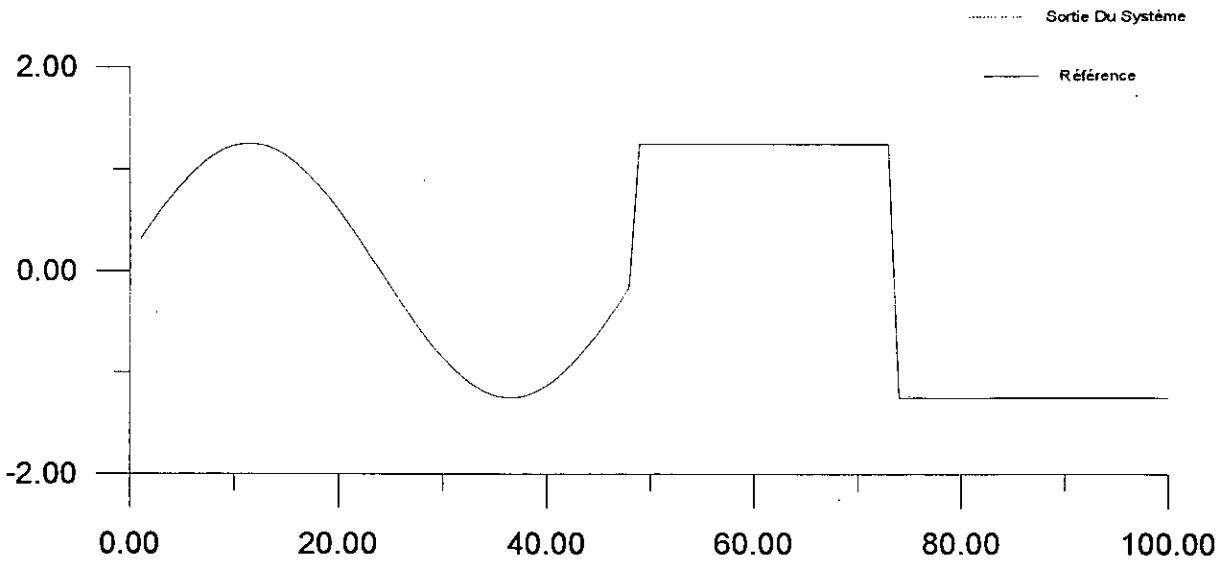


Figure 41 : Sortie du système et référence lors de l'utilisation d'un régulateur auto-ajustable..

7. Contrôleur basé sur les fonctions critiques :

Dans cette partie deux contrôleurs sont appliqués un double intégrateur et un modèle simplifié d'un robot d'exploration des fonds sous-marins; Leurs synthèse est basés sur les fonctions critiques.

Dans cette technique, le réseau de neurones apprend minimiser la fonction critique :

$$e(t) = f\big(x(t), u(t)\big) \tag{III-74}$$

avec :

$x(t)$: états du système.
$u(t)$: commande(s).

Cette fonction possède plus particulièrement la forme [Sanner and Akin, 1990]:

$$e(t) = M_x\big[x_d(t) - x(t)\big] + M_u\, u^*(t) \tag{III-75}$$

avec : M_x est une Matrice $m \times n$, M_u est une matrice $m \times m$, $u^*(t)$ est la valeur normalisée de la commande elle est définie par :

$$u^*(t) = -\left(u_i(t) \Big/ u_{ult,i}\right)^n, n = 2k+1, k \in N \tag{III-76}$$

L'effet de ce terme est qu'il permet d'accélérer la diminution de la fonction critique si $u_i(t)$ s'approche de - $u_{ult\,i}$, la commande sera donc augmentée, et vice-versa quand $u_i(t)$ s'approche de $u_{ult\,i}$. A partir de l'équation (III-75) que le réseau de neurones tend développer une commande $u^*(t)$ définie par :

$$M_x\left[x_d - x(t)\right] = -M_u u^*(t) \qquad \text{(III-77)}$$

Comme en commande optimale, les performances de cette méthode dépendent du choix de la fonction coût.

Le schéma d'apprentissage de ce régulateur est schématisé dans la fig.42.

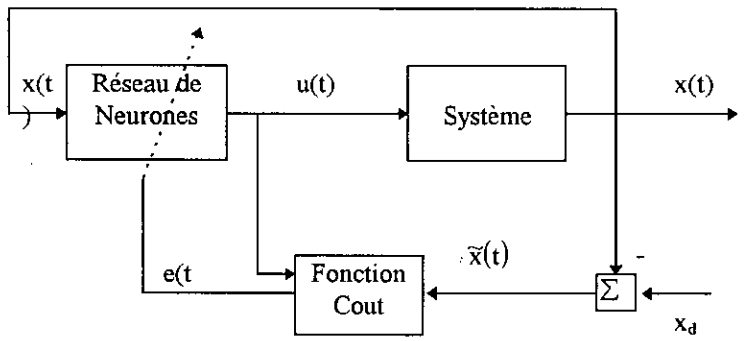


Fig. 42 :Controlleur basé sur les fonctions critiques

Exemple 1:

Dans cet exemple on applique la commande discutée ci-dessus un double intégrateur, qui est un problème simple, mais qui en train d'être réexaminé par les chercheurs car il a plusieurs implications pratiques (Modèle d'un satellite dans l'espace...) [Slotine and Li, 1991]. Son modèle est :

$$\ddot{y} = u \qquad \text{(III-78)}$$

La référence dans ce type de contrôleurs est fixe, dans notre cas: $\mathbf{x}_d = [1.0\ ,\ 0.0]^T$.

L'apprentissage d'un réseau statique appartenant la classe $\eta_{2,3,1}$, a été effectué en utilisant 10 conditions initiales (avec un période d'échantillonnage égale 0.02 sec.). Les figures 43 et 44 montrent les évolutions de la vitesse et de la position, en utilisant respectivement [0, 0] et [2, 0] comme conditions initiales, et comme paramètres de la fonction critique $M_x = [1.5\ ,\ 1]$ et $M_u = 0.5$ et $n = 3$. On remarque que le système converge vers la référence fixe.

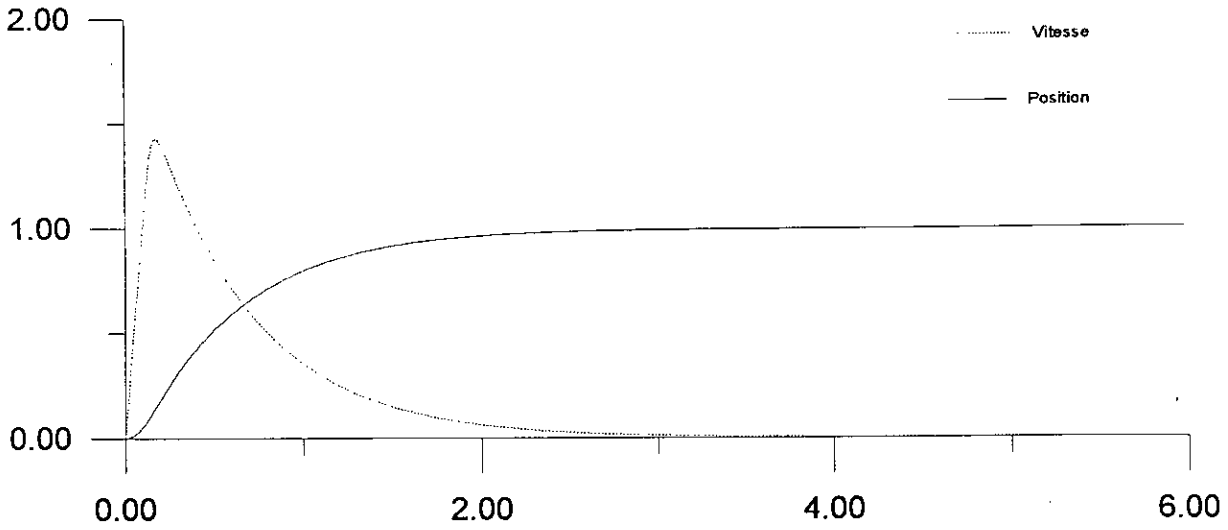


Fig. 43: Réponse du double intégrateur quand $\mathbf{x}_0=[0,0]$

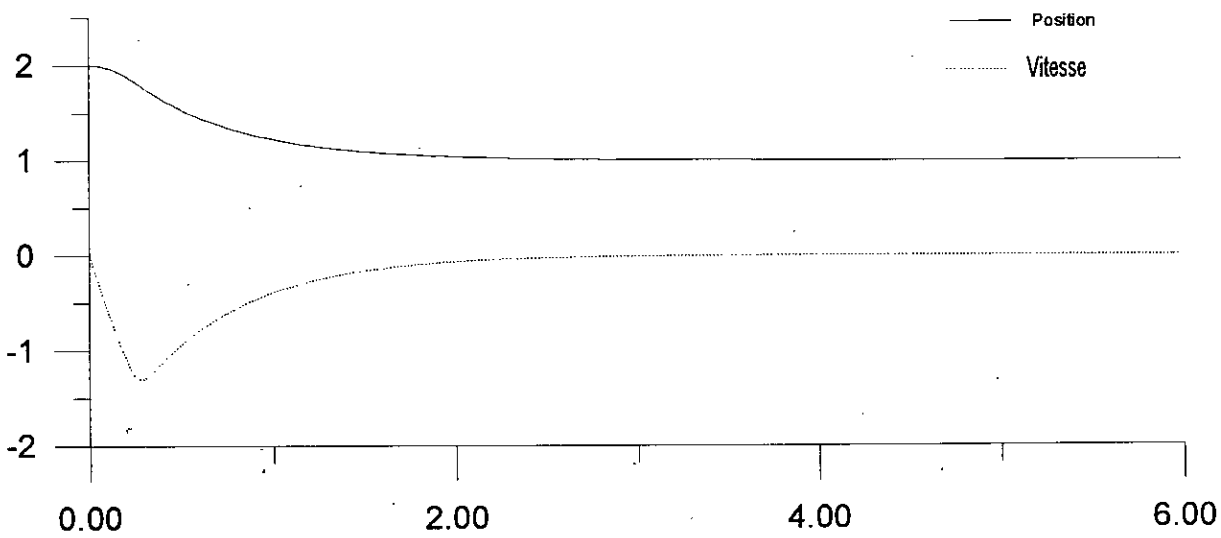


Fig. 44: Réponse du double intégrateur quand $\mathbf{x}_0=[1,0]$

Exemple 2:

Dans cet exemple on applique la technique précédente la commande d'un robot d'exploration des fonds sous-marins, son modèle simplifié est donné par [Slotine and Li, 1991]:

$$I_\theta \ddot{\theta} = -C_{d,\theta}\dot{\theta}|\dot{\theta}| + \tau_\theta \qquad \text{(III-79)}$$

avec : $I_\theta = 300\ \text{kg}-\text{m}^2, C_{d,\theta} = 750\ \text{kg}-\text{m}^2$ et $|\tau_0| \le 288\,\text{N}-\text{m}$.

Dans ce cas: $\mathbf{x}_d = [1.0\ ,\ 0.0]^T$.

L'apprentissage d'un réseau statique appartenant la classe $\eta_{2,3,1}$, a été effectué en utilisant 15 conditions initiales (avec un période d'échantillonnage égale 0.05 sec.). Les figures 45 et 46 montrent les évolutions de la vitesse et de la position, en utilisant respectivement [0, 0] et [2, 0] comme conditions initiales, et comme paramètres de la fonction critique $M_x = [1.5 , 0.5]$ et $M_u = 0.3$ et $n = 3$. On remarque que le système converge vers la référence fixe.

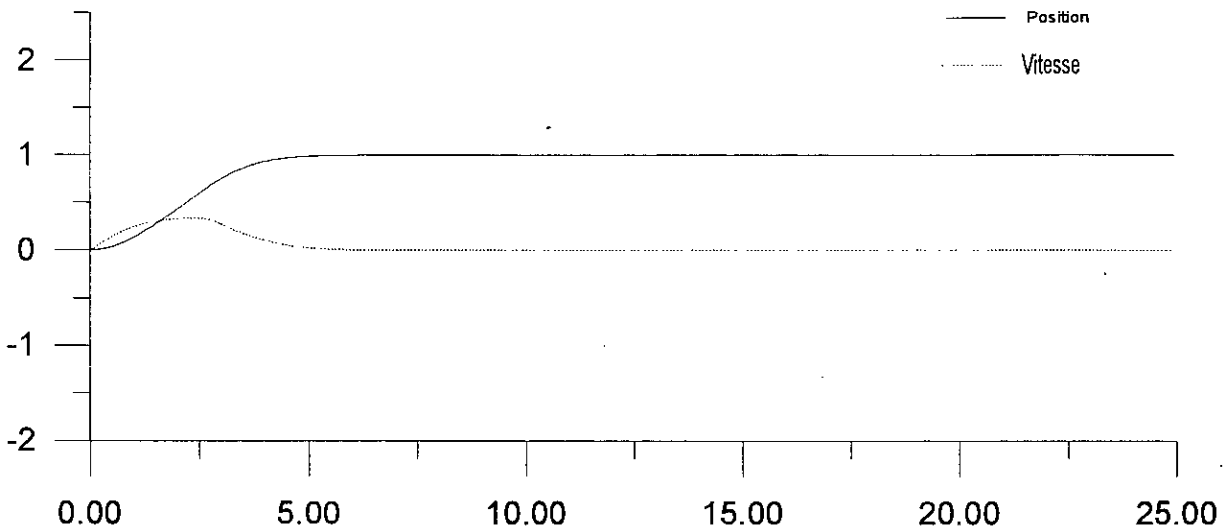


Fig. 45: Réponse du robot quand $\mathbf{x_0}$=[0,0]

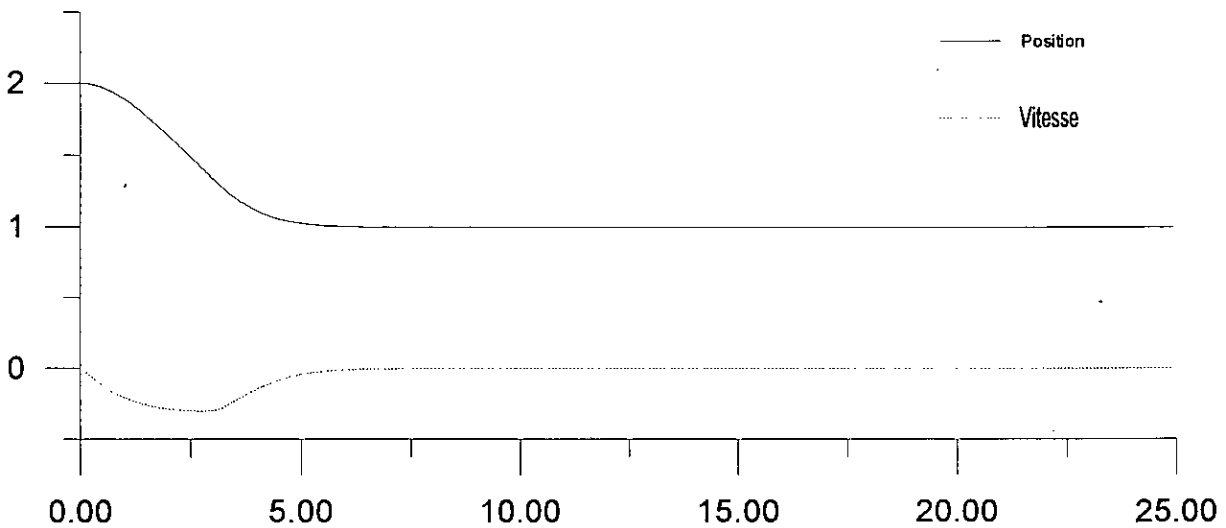


Fig. 46: Réponse du robot quand $\mathbf{x_0}$=[1,0]

On remarque que le système suit la référence. Notons que la référence est toujours fixe, on ne peut guère spécifier une dynamique désirée.

IX. Conclusion :

Dans ce chapitre nous avons proposé des méthodes qui peuvent être utilisées d'une manière efficace dans la commande des systèmes non-linéaires.

On a d'abord commencé par la commande supervisée qui a été appliquée la stabilisation d'un pendule inversé, et ceci grâce un réseau de neurones dynamique, on a remarqué que ce réseau présentait de bons résultats lors de son implantation comme contrôleur; ensuite on a effectué des tests de robustesse, et il c'est avéré que, contrairement au contrôleur implanté par un réseau statique qui n'était pas robuste, ce contrôleur l'est; Cependant ce type de régulateur ne peut qu'au mieux implanter le régulateur appris, avec ses avantages et ses inconvénients.

Ensuite, on a parlé de la commande inverse qui est basée sur la détermination de la fonction inverse du système, quatre schémas d'apprentissage ont été proposés, chacune d'elle possède sa propre vision sur le problème, cette méthode est limitée par l'existence de la fonction inverse.

Dans la troisième section, on a parlé de l'approche neuronale de la commande adaptative avec modèle de référence, cette méthode dépend du modèle auquel appartient le système non-linéaire, s'il le système est linéaire par rapport la commande, la synthèse du régulateur est directe, elle utilise directement les résultats de l'identification; Par contre, si le système est non-linéaire par rapport la commande, un réseau doit être entraîné pour apprendre la caractéristique inverse de cette fonction; Si le système est fortement non-linéaire la commande adaptative avec modèle de référence se confond avec la commande inverse.

Dans la quatrième section on a parlé de l'important problème du rejet de perturbations, le principe de cette méthode est l'augmentation de la dimension de l'espace d'état pour permettre d'absorber la perturbation - considérée comme sortie d'un système non-linéaire en régime libre -, on a pu remarquer lors des simulations que la commande adaptative n'était pas robuste, et qu'il fallait faire appel au "duo de choc", Réseaux récurrents en entrée - Augmentation de l'espace d'état pour avoir un peu de robustesse.

On a ensuite cité la commande par bouclage linéarisant qui découle directement des contrôleurs classiques basés sur la méthode du même nom, on a appliqué cette méthode un bras de robot du type PUMA 560, et on a remarqué que le réseau a pu surmonter la problème de la dynamique non-modélisée.

Ensuite on a parlé du régulateur auto-ajustable dans sa version neuronale, la simulation de ce contrôleur sur un exemple non-trivial, a prouvé son efficacité.

En dernier lieu, on a parlé de la synthèse de contrôleurs basés sur le fonctions critiques, ces contrôleurs ne doivent être utilisés que lorsqu'on ne possède pas beaucoup d'informations sur le système commander; La synthèse de ce type de régulateur ne permet pas d'imposer une dynamique, en plus elle est essentiellement basée sur la forme de la fonction critique.

Comme conclusion, on peut dire que les *réseaux de neurones* permettent de commander des systèmes non-linéaires complexes; Cependant il ne faut *pas en abuser* en les utilisant comme contrôleurs de systèmes linéaires, car cette méthode ne sera pas efficace; Ceci peut être facilement expliqué en utilisant la définition de Hecht-Nielsen, dans cette dernière un réseau de neurones est un approximateur de fonctions, basé sur des éléments non-linéaires appelés neurones, l'utilisation des réseaux de neurones comme contrôleurs des systèmes linéaires revient approximer une fonction linéaire par des fonctions fortement non-linéaires, qui est une tache relativement difficile. Ainsi, avant d'utiliser les réseaux de neurones comme contrôleurs on devrait être surs qu'un contrôleur linéaire ne peut résoudre le problème.

Ainsi, " *les réseaux de neurones, c'est bien mais il ne faut pas en abuser...*"

CONCLUSION GENERALE

"Je me fais l'impression de n'avoir été qu'un enfant jouant sur la plage et s'y amusant a y trouver de temps en temps un galet particulièrement lisse ou un coquillage plus joli que les autres, tandis que s'étendait devant moi, inconnu, le grand océan de la vérité."

Isaac NEWTON (1642-1727)

CONCLUSION GENERALE

Les capacités d'approximer n'importe quelle fonction, d'apprentissage et de généralisation des réseaux de neurones donnent des issues nouvelles pour les problèmes de commande et d'identification des systèmes non-linéaires, dont le traitement se heurte encore aujourd'hui à des difficultés d'ordre pratique et théorique.

Dans notre travail on a pu voir que les réseaux de neurones étant des approximateurs universels, semblent apporter une solution au problème d'identification des systèmes non-linéaires. Leurs propriétés d'adaptabilité et de stockage associatif sont utilisées pour l'entraînement de contrôleurs neuronaux capables de réaliser des commandes adaptatives et optimales, alors que les techniques classiques sont basées soit sur la commande adaptative, soit sur la commande optimale. Les méthodes que nous avons appliquées, ont des performances supérieures à celles des méthodes classiques; les résultats obtenus sont très intéressants, et montrent l'efficacité de l'approche neuronale.

Cependant on devrait noter que :

- Les algorithmes de commande basés sur la descente du gradient doivent être maniés avec précautions, car un mauvais choix des paramètres du contrôleur risque de déstabiliser le système global; Ainsi, on devrait essayer de déterminer des algorithmes d'apprentissage qui permettent la stabilité du système global, on propose pour cela la combinaison entre la théorie de la stabilité de Lyapounov et les RBFN [Chen, 1991] à la place des réseaux multicouches.

- Les réseaux de neurones dynamiques utilisés en identification et en commande ont prouvé leurs efficacité, cependant plusieurs problèmes liés a ces réseaux existent (choix de poids initiaux, choix des pas d'apprentissage, choix du signal d'entraînement); On devrait donc essayer d'approfondir les recherches dans l'étude de la stabilité des réseaux de neurones dynamiques.

- Les réseaux utilisés n'ont pas une architecture minimale, on devrait donc utiliser des réseaux de neurones qui possèdent une architecture dynamique qui permettent un nombre minimal de paramètres, on propose d'utiliser les GPFN [Lee and Kil, 1988, 1991].

- Les techniques d'identification et de commande n'ont été appliquées qu'a des systèmes déterministes, on devrait donc les appliquer à des systèmes stochastiques.

- Lors de l'application de la commande par bouclage linéarisant, on a pu remarquer, que pour commander une articulation on devait utiliser des données qui dépendent de l'autre articulation, on devrait donc essayer de synthétiser des contrôleurs neuronaux locaux.

- Les réseaux de neurones lorsqu'ils sont appliqués a des systèmes linéaires ne donnent pas d'excellents résultats, on devrait donc essayer de synthétiser des contrôleurs neuronaux pour les systèmes linéaires, on propose pour cela l'identification des paramètres du système linéaire en utilisant les réseaux de Hopfield en minimisant une certaine fonction d'énergie, dans le cas continu, ou l'utilisation des statistiques d'ordres supérieurs pour l'identification des paramètres d'un systèmes linéaire discret; Ces paramètres peuvent ensuite être utilisés pour synthétiser des contrôleurs linéaires adaptatifs.

ANNEXES

"Si j'ai appris une chose au cours de ma longue vie
c'est que toute notre science confrontée a la réalité
apparaît comme primitive et enfantine, et pourtant
c'est ce que nous possédons de plus précieux."

Albert EINSTEIN (1879-1955)

Annexe

CONSIDERATIONS PRATIQUES

Dans cette partie quelques "recettes" vont être données, qui permettraient au lecteur de concevoir un réseau de neurones qui s'adapte à son problème. Cette partie sera subdivisées en questions auxquelles on répondra .

- *Le problème peut il être résolu en utilisant les réseaux de neurones ?*

Avant de concevoir un réseau de neurones, la question concernant l'applicabilité des réseaux de neurones au problème considéré se pose. Les problèmes auxquels les réseaux de neurones sont applicables, sont ceux ou une réponse analytique n'existe pas. Dans le cas de réseaux de neurones entraînés en utilisant la "Backpropagation", des sorties désirées doivent exister. Pour pouvoir exploiter la capacité de généralisation des réseaux de neurones, on devrait se souvenir que les réseaux de neurones sont des approximateurs de fonctions; Ainsi on devrait être sur qu'une fonction qui relie les entrées aux sorties existe; sinon on se trouverait en train d'essayer de prédire l'imprévisible; Par exemple, Un réseau de neurones qui apprend la correspondance entre le nom des gens et leurs numéros de téléphone, ne nous aidera guère à connaître le numéro de téléphone d'un personne dont le numéro n'a pas été appris (il pourrait ne pas avoir un téléphone.)

- *Comment traiter les exemples ?*

  Les réseaux de neurones, dépendent beaucoup de la qualité des données qu'on leurs présente, ces données doivent être mises à l'échelle des fonctions d'activation utilisées; Par exemple il suffit d'observer la fonction tanh(.), pour voir qu'elle est "linéaire" entre -1 et 1, dans ce cas la sortie varie entre -.8 et +.8; On devrait donc normaliser les entrées entre -1 et +1, et les sorties entre -.8 et +.8; Sinon, et s'il arrivait qu'un neurone possède comme entrée, i.e. la somme pondérée, qui soit très élevée, alors la sortie serait constante, car la fonction d'activation serait saturée, dans ce cas la neurone n'apprendra rien, car la dérivée de la fonction d'activation serait nulle.

- *Quelles sont les dimensions du réseau ?*

Dimensionner le réseau est une tache importante; Des résultats théoriques ont pu être obtenu dans le cas d'un réseau statique à une seule couche cachée; Ce résultat a pu être obtenu par Baum et Haussler [Baum and Haussler, 1989], il pose des limites supérieures et inférieures sur le nombre de poids; Si nous notons par N_p le nombre d'exemples, N_x le nombre d'entrées, N_y le nombre de sorties, et N_w le nombre de poids alors on aura:

$$\frac{N_y N_p}{1+\log_2(N_p)} \le N_w \le N_y\left(\frac{N_p}{N_x}+1\right)(N_x+N_y+1)+N_y,$$

On s'aperçoit rapidement, que pour un nombre élevé d'exemples, la limite inférieure devient grande; pour cela il faut parfois ajouter une seconde couche cachée, cette seconde couche implique, en général, une diminution du nombre de neurones dans la couche précédente, une relation équivalente a la limite de Baum, dans le cas d'un réseau qui possède plus qu'une seule couche cachée n'existe pas.

En fait, c'est votre expérience qui vous permettra de déterminer les dimensions du réseau, plusieurs "recettes" été établies, sans aucune démonstration, On peut citer quelques unes :

- Plus la relation entre les entrées et les sorties devient complexe, plus il faut augmenter le nombre de neurones par couche, mais apparemment une limite supérieure au delà de laquelle les performances du réseau se dégradent, existe; En plus, *parfois* augmenter le nombre de neurones augmente le temps de calcul.
- Le nombre minimal h de neurones dans la couche cachée, pour un réseau qui ne possède qu'une seule couche cachée est :

$$h = \frac{N_p}{10 * (N_x + N_y)}.$$

- Le nombre de couches cachées est égal au produit du nombre d'entrées par le nombre de sorties.
- Le nombre minimal h de neurones dans la couche cachée est :

$$h = 2 * \sqrt{(m+n)} .$$

- *Quand faudrait-il arrêter l'apprentissage ?*

Cette question est importante, car si on arrête l'apprentissage trop tôt le réseau n'aura rien appris, et si on arrête l'apprentissage trop tard le réseau risque d'apprendre le bruit qui entache les données, ce phénomène s'appelle *sur-spécialisation*, pour mieux comprendre ce phénomène, considérons le problème de l'approximation de la fonction $y = f(x)$, supposons que ces données soient entachées de bruit; Ainsi, au fur et à mesure de l'apprentissage, les réseau améliore ses performances jusqu'à réaliser une fonction proche de la fonction désirée. En poussant l'apprentissage, le réseau va apprendre les données bruitées, car il est capable d'approximer des fonctions plus complexes. D'aprés [Gérard, 1992], pour éviter la sur-spécialisation, il faudrait que le réseau possède une architecture adaptée aux données, ce réseau doit posséder la propriété que le rapport entre les poids et le nombre d'exemples soit de 1/10. D'autre part, afin d'éviter la sur-spécialisation, on conseille des réseaux de neurones qui possèdent une architecture dynamique, qui convergent vers un nombre minimal de neurones tels que les "Gaussian Potential Function Network " GPFN [Lee and Kil, 1988] (dans le cas d'une seule sortie) .

REFERENCES BIBLIOGRAPHIQUES

"Pour faire de grandes choses il ne faut pas être un si grand génie, il ne faut pas être au dessus des hommes, il faut être avec eux."

Charles DE SECONDAT MONTESQUIEU (1689-1755)

REFERENCES
BIBLIOGRAPHIQUES

Abou-Mustafa, Y. (1989). Information Theory, Complexity, and Neural Networks. *In Penkaj Mehra and Benjamin W. Wah, Artificial Neural Networks: Concepts and Theory.*, pp.474-478. IEEE Computer Society Press Tutorial, 1992.

Astrom, K. J. (1987). Adaptive Feedback Control. *Proc. IEEE*, 75, 185-222

Astrom, K. J. and B. Wittenmark (1989). *Adaptive Control.* Addison-Wesley, New York.

Astrom, K. J. and B. Wittenmark (1990). *Computer Controlled Systems*, Prentice-Hall.

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, **9**, 31-37.

Barron, A. R. (1991). Approximation and estimation bounds for artificial neural networks. *In Penkaj Mehra and Benjamin W. Wah, Artificial Neural Networks: Concepts and Theory.*, pp.500-506. IEEE Computer Society Press Tutorial, 1992.

Barto, A. G. (1990). *Neural Networks for Control*, Chapter 1, pp. 5-58. MIT Press, Cambridge, MA.

Barto, A. G., R. S. Sutton and Ch. W. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on System, Man, and Cybernetics*, **13**, 834-846.

Baum, E. B. (1989). What Size net gives valid generalization ?. *Neural Computation*, **1**, 151-160.

Bhat, N. V., P. A. Minderman, T. J. McAvoy and N. S. Wang (1990). Modeling chemical process systems via neural computation. *IEEE Control Systems Magazine*, **10**, 24-29.

Chen, F.C., (1990). Backpropagation Neural Networks for nonlinear self-tuning adaptive control. *IEEE Control Systems Magazine*, **10**, 44-48.

Chen, S., C. F. N. Cowan and P. M. Grant (1991). Orthogonal Least Squares Learning Algorithm for Radial basis function networks. *IEEE Trans. on Neural Networks*, **2**, 302-309.

Chi, S. R., R. Shoureshi and M.Tenorio (1990). Neural networks for system identification. *IEEE Control Systems Magazine*, **10**, 31-34.

Cohen, M. A. and S. Grossberg (1983). Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. on System, Man, and Cybernetics*, **13**, 815-826.

Craig, J. J. and P. H. S. S. Sastry (1987). Adaptive control of mechanical manipulators. *The international Journal of robotics research*, **6**, 16-28.

le Cun, Y. (1988). A theoretical framework for back-propagation. In *Penkaj Mehra and Benjamin W. Wah, Artificial Neural Networks: Concepts and Theory.*, pp.331-338. IEEE Computer Society Press Tutorial, 1992.

Cybenko, G. (1989). Approximation by superposition of a sgmoidal function. *In Penkaj Mehra and Benjamin W. Wah, Artificial Neural Networks: Concepts and Theory.*, pp.488-499. IEEE Computer Society Press Tutorial, 1992.

Dote, Y. (1990). Fuzzy and neural networks controller . *Proc. IEEE*, ?, 1314-1343.

Elnayar, S. V. T. and Y. C. Shin (1994). Radial Basis function neural networks for approximation and estimation of nonlinear stochastic dynamic systems. *IEEE Trans. on Neural Networks*, 5, 594-603

Elsgolc, L.E. (1961). *Calculus of Variations*, Pergamon Press, 1961.

Freeman, J. A. and D. M. Skapura (1992). *Neural Networks: Algorithms, Applications and Programming Techniques*. Addison-Wesley Publishing Company.

Fu, K.S. (1970). Learning Control-review and outlook. *Trans. IEEE on Aut. Control*, **16**, 210-221.

Fukuda, T. and T. Shibata (1992). Theory and applications of neural networks for industrial control systems. *IEEE Trans. on Industrial Electronics*, **39**, 472-489.

Funahashi, K.I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, **2**, 183-192.

Gerard, T. (1992). Application aux réseaux de neurones de la théorie de "minimisation de risque " de V. Vapnik. *Rapport de Stage au Labo. I. A. R. F. A, Université P. et M. Curie-Laforia*, 1-29.

Giles, C.L. and T.Maxwell (1987). Learning, invariance, and generalization in high-order neural networks. *In Penkaj Mehra and Benjamin W. Wah, Artificial Neural Networks: Concepts and Theory*., pp.94-100. IEEE Computer Society Press Tutorial, 1992.

Girosi, F. and T.Poggio (1989). Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation*, **1**, 465-469.

Goodwin, G. C. and K. S. Sin (1984). *Adaptive Filtering Prediction and Control.* Prentice-Hall, Englwood Cliffs, NJ.

Guez, A. and J. Selinsky (1988). A Trainable Neurormorphic Controller. *Journal of Robotic Systems*, **5**, 363-388.

Hammerstrom, D. (1993). Working with neural networks, IEEE Spectrum, 46-53.

Hunt, K. J., D. Sbarbaro, R. Zbikwoski, P. J. Gawthrop (1992). Neural Networks for Control Systems - A survey. *Automatica*, **28**, 1083-1112.

Isidori , A. (1989), Nonlinear Control Systems : An introduction, Springer Verlag, 1989.

IEEE (1990). Special Issue on neural networks. IEEE Control Systems Magazine, 10.

Karakasoglu, A., S. Sudharsanan and M. K. Sundareshan (1993). Identification and decentralized adaptive control using dynamical neural networks with application to robotic manimpulators. *IEEE Trans. on Neural Networks*, **4**, 919-930.

Karayiannis, N. B. and A. N. Venetsanopoulos (1993). *Artificial Neural Networks: Learning Algorithms, Performance Evaluation and Applications*, Kluwer Academic Press.

Kosko, B. (1992).*Neural Networks and Fuzzy Systems: A dynamical approach to machine intelligence*. Prentice-Hall.

Kung, S.Y. and Hwang J. N. (1989). Neural Network Architectures for Robotic Applications, *IEEE Trans. on Robotics and Automation*, **5**, 641-657

Kolmogorov, A. and S. Fomine (1973): Elements of the theory of functions and functional analysis.

Lee, S. and Kil, R. M. (1988). Multilayer Feedforward Potential Function Network. *In Penkaj Mehra and Benjamin W. Wah, Artificial Neural Networks: Concepts and Theory.*, pp.83-93. IEEE Computer Society Press Tutorial, 1992.

Lee, S. and R. M. Kil (1991). A Gaussian Potential function network with hierarchically self-organizing learning. *Neural Networks*, **4**, 207-224.

Levin. A. U., and K. S. Narendra (1993). Control of nonlinear dynamical systems using neural networks: controllability and stabilization. *IEEE Trans. on Neural Networks*, **4**, 192-206.

Li, J. H., A. N. Michel and W. Porod (1988). Qualitative analysis and synthesis of a class of neural networks. *IEEE Trans. on Circuits and Systems*, **35**, 976-986.

Lippman, R. P. (1987). An introduction to computing with neural nets. *In Penkaj Mehra and Benjamin W. Wah, Artificial Neural Networks: Concepts and Theory.*, pp.13-31. IEEE Computer Society Press Tutorial, 1992.

Ljung, L. (1987). *System Identification-Theory for the User.* Prentice Hall, Englewood Cliffs, NJ.

Ljnug, L. and T. Soderstrom (1983). *Theory and Practice of Recursive Identification.* MIT Press, London.

Mehra, P. and B. W. Wah, *Artificial Neural Networks: Concepts and Theory.*, IEEE Computer Society Press Tutorial, 1992.

Michell, A. N., J. A. Farrell and W. Porod (1989). Qualitative analysis of neural networks. *IEEE Trans. on Circuits and Systems*, **36**, 229-243

Miller, W. T., R. S. Sutton and P. J. Werbos (1990). *Neural Networks for Control.* MIT Press, Cambridge, MA.

Mukhopadhyay, S. and K. S. Narendra (1993). Disturbance rejection in nonlinear systems using neural networks, *IEEE. Trans. on Neural netwoks*, **4**, 63-72

Nagata, S. , M. Sekigushi, and K. Asakawa. (1990). Mobile Robot control by a structured hierarchical neural network. *IEEE. Control Systems Magazine*, **10**, 69-76.

Narendra, K. S. (1990). *Neural Networks for Control,* Chapter 5, pp. 115-142. MIT Press. Cambridge.

Narendra, K. S. and A. M. Annaswamy (1989). *Stable Adaptive Systems.* Prentice-Hall, Englwood Cliffs, NJ.

Narendra, K.S. and K. Parthasarathy (1990). Identification and control of dynamical s ystems using neural networks. *IEEE Trans. on Neural Networks*, **1**, 4-27.

Narendra, K.S. and K. Parthasarathy (1991). Gradient methods for the optimization of d-ynamical systems containing neural networks. *IEEE Trans. on Neural Networks*, **2**, 252-262.

Ozaki, T. , Suzuki, T. ,Furuhashi, T., Okumu, S. and Uchikawa, Y. (1991). Trajectory Control of robotics manipulators using neural networks. *IEEE Trans. on Industrial Electronics*, **3**,195-202

Parker, T. S. and L.O. Chua (1987). Chaos: A tutorial for engineers. *Proc. of the IEEE*, **75**, 982-1008.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, **1**, 263-269.

Poggio, T. and F. Girosi (1990). Networks for approximation and learning: *Proc. of IEEE*, **78**, 1481-1497.

Psaltis, D., A. Sideris and A. A. Yamamura (1988). A multilayered neural network controller. *IEEE Control Systems Magazine*, **8**, 17-21.

Qin, S.Z. , Su. H.T. and T. J. McAvoy (1992). Comparison of neural net learning methods for dynamic system identification. *IEEE Trans. on Neural Networks*, **3**, 122-130.

Saad, M., A. Dessaint, P. Bigras, and K. Al-Haddad. (1994). Adaptive versus neural adaptive control: Applications to robotics. *International Journal of Adaptive Control and Signal Processig*, **8**, 223-236.

Sanner, R. M. and D. L. Akin (1990). Neuromorphic pitch attitude regulation of an underwater telerobot. *IEEE Control Systems Magazine*, **10**, 62-67.

Sbarbaro, D., D. Neumerkel, and K. Hunt (1993). Neural Control of a Steel Rollong Mill. *IEEE Control systems Magazine*, June, 69-75.

Sbarbaro, D., K. J. Hunt and P. J. Gawthrop. (1991). Designing Nonlinear Controllers Using Connectionist Networks. 13 *th IMACS World Congress on Computation and Applied Mathematics*, July 22-26, 1991, Trinity College, Dublin, Ireland.

Seraji, H. (1989). Decentralized adaptive control of manipulators: Theory, Simulation and experimentation. *IEEE Trans. on robotics and automation*, **5**, 183-201.

Slotine, J. J. and W. Li.(1988). Adaptive Manipulator Control: A case Study. *IEEE. Trans. on Automatic Control*, **11**, 995-1003.

Slotine, J. J. and W. Li (1991). *Applied Nonlinear Control*. Prentice Hall.

Sudharsanan, S. and Sundareshan, M. K. (1991). Equilibrium Characterisation of Dynamical Neural Networks and a Systematic Synthesis Procedure for Associative Memories. *IEEE Trans. on Neural networks*, **2**, 509-521.

Tank, D. and J. J. Hopfield (1986). Simple "Neural" Optimization Networks: An A/D Converter, Signald Decision Circuit and a Linear programming Circuit, *IEEE Trans. on circuits and systems*, **CAS-33**, 533-541.

Venugopal, K. P. and S. M. Smith (1993). Improving the Dynamic response of neural networks controllers using velocity reference feedback. *IEEE Trans. on neural networks*, **4**, 355-357.

Werbos, P. J. (1989). Neural Networks for control and system identification. *Proceedings of the 28th conference on decision and control*. Tempa. Florida, December 1988.

Werbos, P.J. (1990a). Backpropagation through time: what it does and how to do it? *Proc. of IEEE*, **78**, 1550-1560.

Werbos, P.J. (1990b). *Neural Networks for Control*, Chapter 3, pp. 67-95. MIT Press. Cambridge, MA.

Widrow, B. and M. A. Lehr (1990). 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation, *Proc. of the IEEE*, **78**, 1415-1441.

Williams, R. J. and D. Zipser (1989a). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, **1**, 87-111.

Williams, R. J. and D. Zipser (1989b). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, 270-280.

Willis, M. J. , G. Montague, C. Di Massimo, M. T. Tham and A. J. Morris (1992). Artificial neural networks in process estimation and control, *Automatica*,**78**, 1181-186.