DEA DE PRODUCTIQUE:
Automatique et Informatique industrielle

LES RESEAUX DE NEURONES UTILISES EN COMMANDE INTELLIGENTE DE PROCESSUS

MEMOIRE DE DEA

présenté le 3 juillet 1992 par:

Bruno FRANÇOIS

sous la direction de:
Pr. P. BORNE

EC LILLE (anciennement IDN)
Ministère de l'Education Nationale
Cité Scientifique
BP 48
F 59651 Villeneuve d'Ascq Cedex
Tél. : (33) 20.33.53.53
Fax : (33) 20.33.54.99
Télex : 283 155 F

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE
U.F.R. IEEA - Bât. P2
Cité Scientifique
F 59655 Villeneuve d'Ascq Cedex
Tél. : (33) 20.43.45.65
(33) 20.43.47.43
Fax : (33) 20.43.49.95

DEA DE PRODUCTIQUE:
Automatique et Informatique industrielle

LES RESEAUX DE NEURONES UTILISES EN COMMANDE INTELLIGENTE DE PROCESSUS

MEMOIRE DE DEA

présenté le 3 juillet 1992 par:

Bruno FRANÇOIS

sous la direction de:
Pr. P. BORNE

Je déclare que ce mémoire contient des travaux de recherche originaux réalisés pour la préparation d'un diplome d'Etudes Approfondies. Toutes les informations contenues ont été obtenues et présentées conformément aux règles académiques et aux principes d'éthique en vigueur. Tous les éléments et résultats qui ne sont pas propres à ce travail ont été cités et référencés 26/06/1992

CENTRALE LILLE

EC LILLE (anciennement IDN)
Ministère de l'Education Nationale
Cité Scientifique
BP 48
F 59651 Villeneuve d'Ascq Cedex
Tél. : (33) 20.33.53.53
Fax : (33) 20.33.54.99
Télex : 283 155 F

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE
U.F.R. IEEA - Bât P2
Cité Scientifique
F 59655 Villeneuve d'Ascq Cedex
Tél. : (33) 20.43.45.65
(33) 20.43.47.43
Fax : (33) 20.43.49.95

USTL

SOMMAIRE :

PARTIE 2: PRESENTATION DU RESEAU DE NEURONES UTILISE EN COMMANDE

PARTIE 3: UN EXEMPLE DE COMMANDE PAR RESEAU DE NEURONES: LA COMMANDE ADAPTATIVE

PARTIE 4: APPROXIMATION D'UNE FONCTION PAR RÉSEAU DE NEURONES

PARTIE 1:

CONTRIBUTION DES RESEAUX NEURONAUX A LA COMMANDE INTELLIGENTE

1) La commande intelligente: essai de définition

Une très forte association existe entre le mot INTELLIGENCE et l'habilité humaine. Selon la définition du dictionnaire, l'intelligence signifie l'habilité à comprendre, raisonner et apprendre. Une commande possédant ces 3 facultés peut être considérée comme une commande intelligente.
Dans l'industrie, la plupart des machines nécessitent encore à divers niveaux l'intervention de l'homme pour être commandées. Ceci peut s'expliquer par le fait qu'elles ne possèdent pas ces 3 facultés et que c'est l'homme qui, en quelque sorte, les lui apporte. Dans cette démarche, l'intervention humaine est influencée par l'information sensorielle (audio,vidéo) captée lors de l'opération et par le degré d'expertise (expérience) de l'opérateur.
A partir de cette observation, on pourra qualifier d'intelligente, toute commande qui sera capable:
_ de tenir compte d'un très grand nombre d'informations
_ de les interpréter même lorsqu'elles sont incomplètes
_ d'en déduire une connaissance
_ de réagir très vite (en temps réel).

La connaissance peut représenter les caractéristiques d'un processus comme, par exemple, son comportement statique et dynamique, les caractéristiques des perturbations et de l'environnement ...
De plus, il est souhaitable que cette connaissance puisse être mémorisée afin, bien sûr, de pouvoir être utilisable pour commander un processus, mais, aussi, pour améliorer les performances de la commande elle-même au fur et à mesure que l'expérience s'enrichit.

2) Champ d'action de la commande intelligente

2.1) Positionnement des problèmes à résoudre

La commande intelligente est essentiellement utilisée pour le contrôle de systèmes à dynamiques complexes.
Les systèmes complexes sont souvent caractérisés par des modèles pauvres, une grande dimension de l'espace de décision, des niveaux de bruits importants, de multiples sous-systèmes, des échelles de temps, des critères de performance, des formes d'informations complexes (par leur nombre, leur corrélation, leur redondance) ...
et la liste est loin d'être exhaustive.

Les difficultés qui surviennent lors de la commande de tels systèmes peuvent être classées en 3 catégories:
- la complexité
- la présence de non-linéarités
- l'incertitude sur les informations manipulées.

Il faut ajouter que ces difficultés ne sont pas nécessairement dues à un manque d'information sur le processus et son environnement, mais aussi, sur les spécifications même du problème.

2.2) Principe de résolution

La commande intelligente surmonte ces difficultés grâce à ses facultés:
- d'être sensible à son environnement
- de manipuler l'information pour réduire l'incertitude
- de planifier, générer des actions de commande

Un des buts primaires des boucles de retour est d'augmenter la robustesse d'un système de commande: c'est à dire permettre au système de conserver ses performances quand il y a un manque de précision sur les variables manipulées.
Les commandes adaptatives sont un choix naturel dans de telles situations. Leurs principes consistent à incorporer une seconde boucle de retour qui va faire varier les paramètres de la commande afin de maintenir des performances acceptables. Elles permettent, donc, de dominer des perturbations sur les paramètres du système à commander, mais aussi, sur les conditions environnantes et sur les signaux d' entrées. En définitive, elles maîtrisent une certaine échelle d'incertitude (pour plus de détail consulter [1]).
L' objectif de la commande intelligente est similaire à cette philosophie.
La différence est, que, ici, l'échelle d'incertitude est très largement plus grande que celle tolérée par les algorithmes adaptatifs. Pratiquement, elle permet de faire face à des données non dimensionnées et/ou à des structures de données complexes.

K.J. ASTRÖM [2] a recensé plusieurs solutions pour réaliser une commande intelligente de processus:
- les systèmes experts comme éléments adaptatifs.
- le calcul flou pour des éléments produisant ou décidant à partir de connaissances incertaines.
- les réseaux de neurones, présentant les deux avantages mentionnés ci-dessus, apportent en plus un élément compensateur par l'apprentissage.

3) Situation des réseaux neuronaux dans la commande intelligente

Les réseaux de neurones étant des systèmes complètement adaptatifs, leur utilisation dans la résolution de problèmes est originale par rapport aux démarches des méthodes classiques.
En effet, pour une résolution par réseaux de neurones, on part d' un modèle évolutif que l'on cherche à optimiser en vue d'un traitement donné. Ceci a l'avantage de nécessiter peu de connaissance sur le problème contrairement aux méthodes classiques où on cherche d' abord à formaliser le problème pour obtenir un modèle figé.
C'est pourquoi les réseaux de neurones sont très utilisés dans les problèmes de commande n'ayant pas d'expression mathématique précise ou n'ayant pas de solutions par des approches analytiques traditionnelles.
L'adaptation des réseaux neuronaux est basée sur l'ajustement d' un nombre considérable de paramètres (connexions) à partir de mesures.
Cette caractéristique leur apporte les 2 fonctions intelligentes suivantes:
- apprendre par l'expérience, c'est à dire extraire la connaissance à partir de valeurs expérimentales.
- généraliser, c'est à dire utiliser la connaissance acquise lors de l'apprentissage pour synthétiser les conditions de fonctionnement optimal du processus.

Pour une application particulière utilisant un réseau de neurones, on peut préciser ces concepts. Dans le cas où ce réseau commande un processus:
- L'expérience est représentée sous forme de données entrées-sorties, qui sont mesurées en faisant travailler le processus. Ces données constituent la connaissance du problème à fournir au réseau de neurones.
- L'apprentissage consiste à réaliser la relation entre les données des entrées et des sorties en ajustant les paramètres du réseau de neurones.
- La généralisation permet d'estimer, d'interpoler, d'extrapoler des commandes pour des situations et des conditions non envisagées lors de l'apprentissage.

4) Avantages et inconvénients apportés par les réseaux de neurones

Les réseaux neuronaux ont de nombreux atouts [3] de par:
- leur utilisation d'une large quantité d'information
- leur aptitude au calcul parallèle intensif qui permet, en outre, leur utilisation en temps réel
- leur adaptation qui permet d'extrapoler les commandes pour des conditions de fonctionnement changeantes
- leur faculté de commander des systèmes sans connaissance a-priori.

Ils sont particulièrement intéressants dans la commande d'un système qui ne pourrait être commandé à cause du coût de la mise en oeuvre des algorithmes de commande et de la difficulté de trouver ces algorithmes de commande pour des problèmes non-linéaires et/ou complexes.

En commande de processus, le réseau de neurones permet d'approcher la loi de commande qui ne pouvait être déterminée théoriquement ou pratiquement. D'un point de vue général, il est utilisé comme l'outil de résolution des problèmes d'approximation de fonction. Cette fonction pouvant être la relation entrée-sortie du processus (-> Identification), une fonction de choix (-> Classement). L'estimation de cette fonction est faite à partir des données d'entrée et de sortie de la fonction qui, lors d'une phase d'apprentissage, vont mettre en place un schéma particulier de l'organisation des connexions du réseau.

_ Soit que l'on possède une certaine connaissance a-priori du système.
Le réseau de neurones estime alors les paramètres d'une fonction (Voir la commande adaptative auto-ajustable 3eme partie 3)).
_ Soit que le système est totalement inconnu.
Le réseau de neurones estime la fonction entièrement (Voir la commande adaptative avec modèle de référence 3eme partie 2)).

Le manque d'information nécessite une certaine combinaison de l'identification et du contrôle du processus. Plus précisément, on ne peut pas contrôler un processus sans connaître ses caractéristiques, par contre, on peut l'étudier en le commandant. C'est ce que réalise finalement le réseau de neurones.

PARTIE 2:

PRESENTATION DU RESEAU NEURONAL UTILISE EN COMMANDE

1) Introduction

Actuellement, il existe un très grand nombre de structures ou architectures de réseaux qui présentent plus ou moins d'avantages selon leurs utilisations envisagées.
En commande de processus, le plus utilisé est le réseau multicouche (multilayer feedforward neural network). Ceci peut s'expliquer par sa relative facilité d'emploi par rapport aux autres réseaux (avantages apportés par l'algorithme d'apprentissage voir 3.3) et par le fait qu'il puisse résoudre des problèmes qui ont des solutions non linéairement séparables alors que d'autres (perceptron, adaline) ne le peuvent pas.
En effet, on peut considérer que ce réseau effectue une transformation statique non linéaire d'un vecteur d'entrée en un vecteur de sortie. Ceci est réalisé en configurant le réseau par apprentissage de façon à approcher une fonction donnée.

2) Description

2.1) Topologie du réseau

Le "feedforward multilayer neural network" est composé d'éléments de base, appelés neurones (représentés par des cercles et des carrés sur la figure 1), groupés par couches et reliés entre eux par des connexions pondérées chacune d'un poids noté W.

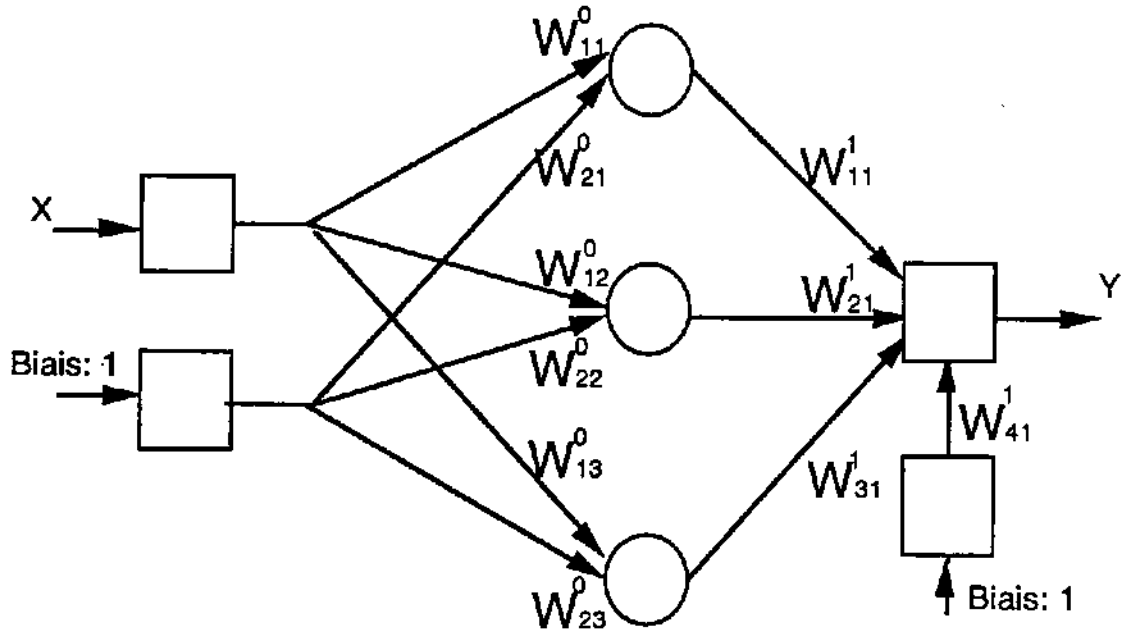

figure 1: Exemple d'un réseau de neurones à 3 couches,
1 entrée et 1 sortie

Il a été prouvé, [4] [5] et [6], qu'un réseau à 3 couches comportant:
_ 1 couche d'entrée (qui permet de mémoriser les valeurs des variables d' entrée)
_ 1 couche cachée
_ 1 couche de sortie (fournissant les valeurs des variables de sortie)
permet d'estimer n'importe quelle fonction continue.

Le nombre de neurones dans les couches d'entrée et de sortie correspond respectivement au nombre de variables dépendantes ou indépendantes d'entrée et de sortie de la fonction à estimer.
Pour ce qui est des neurones de la couche cachée, il n' y a pas de règles pour déterminer a-priori leur nombre. Cependant, il faut savoir que ce nombre doit être supérieur à
(2 x Nombres de neurones de la couche de sortie + 1) et que plus ce nombre est grand, plus la précision du modèle est grande mais, en contre partie, la vitesse de calcul se dégrade, autrement dit, le réseau met plus de temps pour répondre, étant donné l'augmentation de paramètres à calculer.

2.2) Description d'un neurone

Chaque neurone est excité par plusieurs entrées et répond par une seule sortie (figure 1).
Il effectue deux fonctions élémentaires:
_ une fonction de combinaison, qui consiste à effectuer une somme des entrées pondérées par les poids (scalaires) des connexions
_ une fonction d'activation, qui transforme cette somme pour produire la valeur de la sortie du neurone.

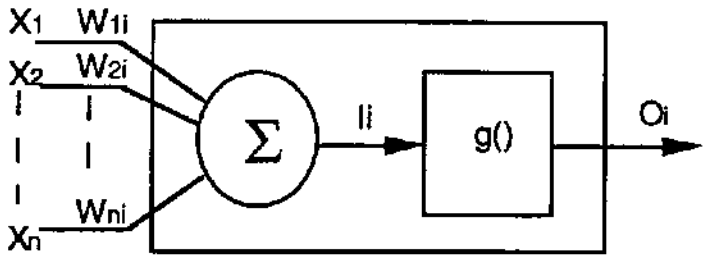

figure 2: fonctionnement d'un neurone

Divers fonctions d'activation existent (voir figure 3).

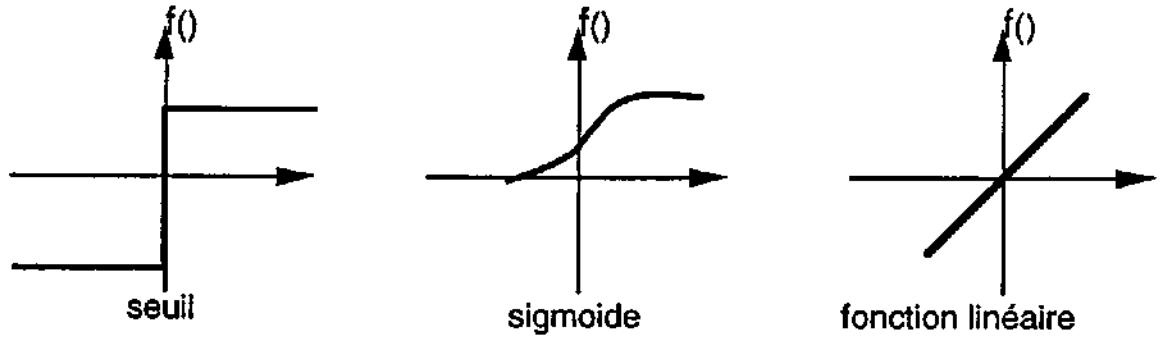

figure 3: Exemples de fonctions d'activation

Les couches d'entrée et de sortie sont constituées de neurones linéaires (représentés par des carrés figure 1); c'est à dire possédant une fonction d'activation linéaire.

En ce qui concerne les fonctions d'activation des neurones de la couche cachée, on peut s'inspirer des résultats expérimentaux suivant:

- si la fonction à approcher est linéaire, les neurones à fonction d'activation linéaire donnent de meilleurs résultats.
- si la fonction est non linéaire, une architecture comprenant 2 parties déconnectées, l'une linéaire et l'autre non linéaire est préférable (voir figure 4).

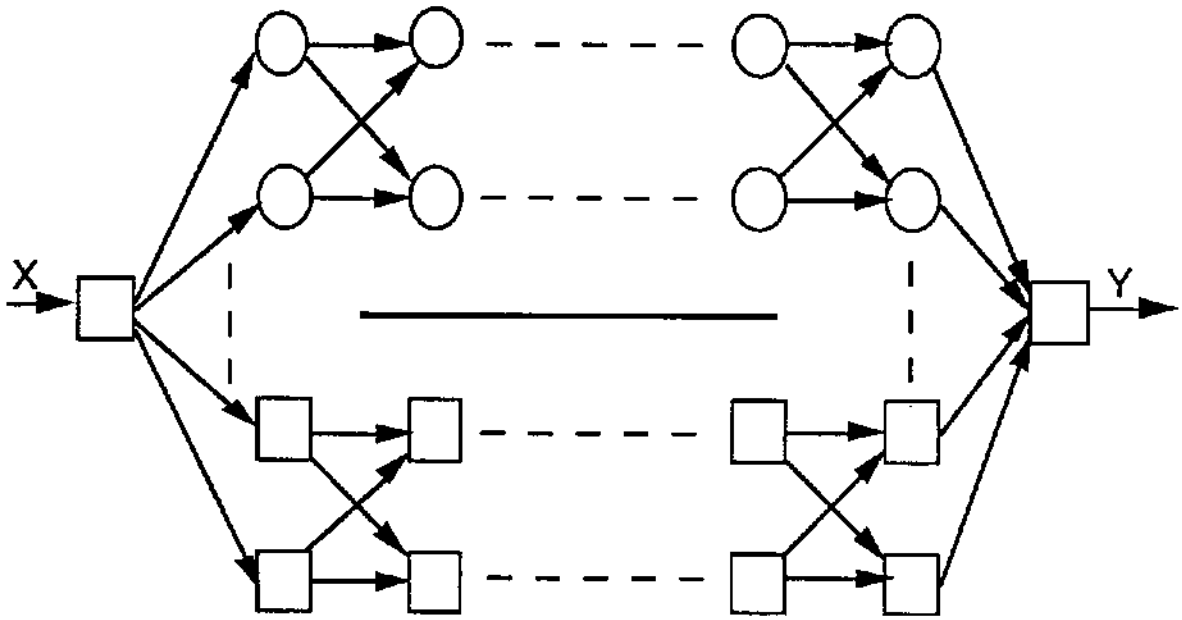

figure 4: Exemple de réseau de neurones se présentant bien pour l'estimation d'une fonction non linéaire

2.3) Formulation mathématique

Pour le réseau de la figure 1, les sorties des neurones de la couche cachée s'écrivent:

$$O_1 = f(W^0_{11}.X_1 + W^0_{21}.1)$$

$$O_2 = f(W^0_{12}.X_2 + W^0_{22}.1)$$

$$O_3 = f(W^0_{13}.X_3 + W^0_{23}.1)$$

et la sortie du réseau: $Y = W^1_{11}.O_1 + W^1_{21}.O_2 + W^1_{31}.O_3 + W^1_{41}.1$
où O1, O2, O3 sont les sorties des neurones de la couche cachée, et, X et 1 les entrées.

Ces équations s'écrivent sous forme vectorielle:

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = f\left(\begin{bmatrix} W^0_{11} & W^0_{21} \\ W^0_{12} & W^0_{22} \\ W^0_{13} & W^0_{23} \end{bmatrix} . \begin{bmatrix} X \\ 1 \end{bmatrix} \right)$$

$$[Y] = \begin{bmatrix} W^1_{11} & W^1_{21} & W^1_{31} & W^1_{41} \end{bmatrix} . \begin{bmatrix} O_1 \\ O_2 \\ O_3 \\ 1 \end{bmatrix}$$

Plus généralement, pour un réseau quelconque (voir figure 5), il est commode de regrouper:

- les N entrées du réseau dans un vecteur entrée $X = \{ X_1, X_2, \ldots, X_N \}$,
- les M sorties dans un vecteur sortie $Y = \{ Y_1, Y_2, \ldots, Y_M \}$
- les poids W_{ij}, reliant le neurone i de la couche l au neurone j de la couche l+1 dans une matrice.

Ceci permet une implantation informatique plus aisée du réseau (voir partie 4: 4.1)).

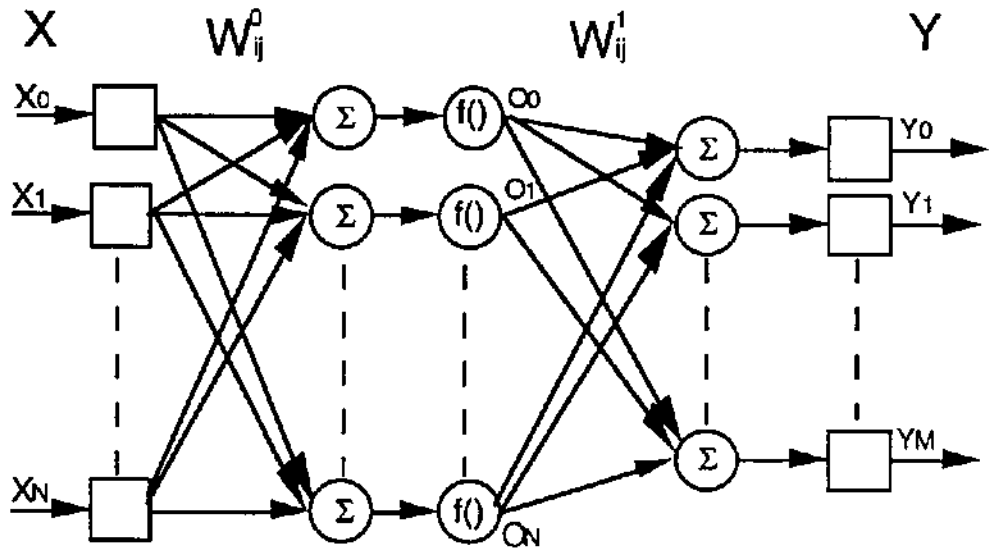

figure 5: Représentation du réseau de neurones à 3 couches

On a alors les équations suivantes:

$$\begin{bmatrix} O_1 \\ O_2 \\ \vdots \\ O_N \end{bmatrix} = f\left(\begin{bmatrix} W^0_{11} & W^0_{21} & \dots & W^0_{N1} \\ W^0_{12} & W^0_{22} & \dots & W^0_{N2} \\ \vdots & \vdots & \dots & \vdots \\ W^0_{1N} & W^0_{2N} & \dots & W^0_{NN} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \right) \quad (1)$$

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_M \end{bmatrix} = \begin{bmatrix} W^1_{11} & W^1_{21} & \dots & W^1_{M1} \\ W^1_{12} & W^1_{22} & \dots & W^1_{M2} \\ \vdots & \vdots & \dots & \vdots \\ W^1_{1M} & W^1_{2M} & \dots & W^1_{MM} \end{bmatrix} \cdot \begin{bmatrix} O_1 \\ O_2 \\ \vdots \\ O_M \end{bmatrix} \quad (2)$$

Si l'on regroupe les poids des deux matrices de connexions dans W^0 et W^1, on obtient:

$[Y] = W^1 \,.\, f(W^0.X)$ noté aussi $W^1 \,.\, f(net)$

3) Apprentissage

3.1) Principe

La phase d'apprentissage consiste d'abord à présenter à la couche d'entrée du réseau un vecteur d' entrée $X = \{ X_1, X_2, \dots, X_N \}$. Les sorties des neurones sont calculées, couche par couche, de la couche d'entrée à la couche de sortie par les formules (1) et (2).

Le vecteur de sortie $Y = \{ Y_1, Y_2, \dots, Y_M \}$ qui apparaît sur les M neurones de la couche de sortie du réseau est comparé au vecteur regroupant les M sorties de la fonction à estimer $S = \{ S_1, S_2, \dots, S_M \}$ et l'erreur obtenue est utilisée pour modifier chaque poids de la matrice de connexion du réseau. L'apprentissage consiste à minimiser l'erreur quadratique obtenue sur l'ensemble des exemples présentés (couples vecteur entrée-vecteur sortie) selon un certain critère.

3.2) Choix des données présentées

Le choix des exemples, leur ordre de présentation et le nombre de présentations à effectuer doit permettre d'obtenir un modèle approprié de la fonction. Le réseau de neurones va apprendre à partir des exemples représentatifs que l'on aura choisi, la relation entre le vecteur entrée X et sortie S de la fonction. Il convient donc de prendre ces exemples dans le domaine de définition de la fonction à estimer, ou, d'un point de vue automatique, dans le domaine de fonctionnement du processus à commander afin que le réseau soit capable de généraliser sa connaissance aux situations différentes de celles envisagées lors de l'apprentissage.

3.3) Algorithme d'apprentissage

Le critère mesurant la qualité des réponses du réseau par rapport à un ensemble d'exemples de la fonction à estimer qui lui est soumis est le critère des moindre carrés.

S_m étant la valeur d'une des sortie de la fonction à estimer, Y_m la sortie effective du neurone correspondant, l'erreur totale commise par la couche de sortie (comportant M neurones) du réseau est:

$$E_t = \frac{1}{2} . \sum_{m=0}^{M} (S_m - Y_m)^2 \quad (3)$$

Cette erreur va être minimisée en ajustant les poids du réseau par l'algorithme de rétropropagation du gradient (encore appelé Generalized Delta Rule). L'idée de cet algorithme, expliqué par D.E. RUMELHART [7], est que si le réseau commet une erreur sur une sortie (Ym - Sm), alors celle ci doit provoquer un réajustement des poids, proportionnellement à cette erreur commise, et de manière à la diminuer. Le gradient s'écrit:

$$W_{ij}(k+1) = W_{ij}(k) - \eta . \frac{\partial E_t}{\partial W_{ij}} \quad (4)$$

$W_{ij}(k+1)$, étant la valeur du poids d'une connexion entre deux neurones i et j à l'itération k+1.

η est le pas de convergence et permet d'augmenter la vitesse d'apprentissage.

Cependant, pour des valeurs grandes de η, il y a un risque d'oscillation. C'est pourquoi, en pratique, on préfère ajouter un terme supplémentaire (momentum term):

$\beta.(W_{ij}(k) - W_{ij}(k-1))$

où $0 < \beta < 1$ détermine l'effet des variations passées sur les poids.

REMARQUE:

Généralement, les poids W_{ij} sont initialisés aléatoirement.

En développant les calculs, on trouve:

$$\frac{\partial E_t}{\partial W_{ij}} = \frac{\partial E_t}{\partial I_i} . \frac{\partial I_i}{\partial W_{ij}}$$

$$\frac{\partial I_i}{\partial W_{ij}} = \frac{\partial \sum_{p=0}^{P} W_{ip}.O_p}{\partial W_{ij}} = \frac{(W_{io}.O_o + \ldots + W_{ij}.O_j \ldots + W_{iP}.O_P)}{\partial W_{ij}}$$

$$= O_j$$

d'où

$$\frac{\partial E_t}{\partial W_{ij}} = \frac{\partial E_t}{\partial I_i} . O_j$$

On calcule d'abord le gradient sur la couche de sortie en utilisant l'erreur entre les M neurones de la couche de sortie du réseau $Y = \{ Y_1, Y_2, \ldots, Y_M \}$ et le vecteur de la fonction à estimer $S = \{ S_1, S_2, \ldots, S_M \}$ (voir figure 6).

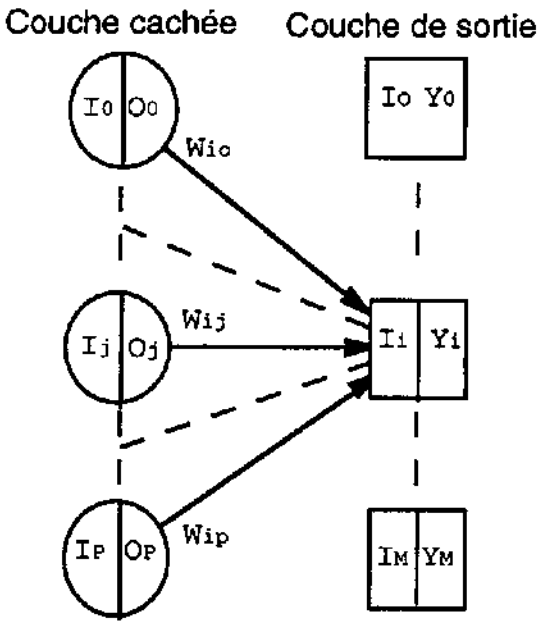

figure 6: Ajustement du poids W_{ij} sur une couche de sortie

$$\frac{\partial E_t}{\partial I_i} = \frac{1}{2} \cdot \frac{\partial \sum_{m=0}^{M} (S_m - Y_m)^2}{\partial I_i}$$

$$= \frac{1}{2} \cdot \frac{\partial \left((S_0 - Y_0)^2 + \dots (S_i - Y_i)^2 + \dots (S_M - Y_M)^2 \right)}{\partial I_i}$$

$$= - (S_i - Y_i) \cdot \frac{\partial Y_i}{\partial I_i} = - (S_i - Y_i) \cdot f'(I_i)$$

d'où

$$\boxed{\frac{\partial E_t}{\partial W_{ij}} = - (S_i - Y_i) \cdot f'(I_i) . O_j} \quad (5)$$

Cette formule implique que la fonction d'activation de chaque neurone, $f()$, soit dérivable.
Les fonctions les plus utilisées sont la sigmoide et la tangente hyperbolique car elles présentent l'avantage d'avoir des dérivées très facilement calculables:

$$O_i = \frac{1}{1+e^{-I_i}} \quad \Rightarrow \quad f'(I_i) = \frac{\partial O_i}{\partial I_i} = [1 - O_i]$$

$$O_i = \frac{e^{I_i} - e^{-I_i}}{e^{I_i} + e^{-I_i}} \quad \Rightarrow \quad f'(I_i) = \frac{\partial O_i}{\partial I_i} = [1 - O_i^2]$$

Les poids de la couche de sortie ayant été ajustés, il faut maintenant répercuter le signal d'erreur (mesuré sur la dernière couche), sur chacune des connexions des couches cachées (problème du 'credit assigment').
Ceci est réalisé en faisant apparaître une cascade de dérivées, c'est à dire pour une couche cachée comportant H neurones:

$$\frac{\partial E_t}{\partial I_j} = \sum_{h=0}^{H} \left(\frac{\partial E_t}{\partial I_h} \cdot \frac{\partial I_h}{\partial I_j} \right) = \sum_{h=0}^{H} \left(\frac{\partial E_t}{\partial I_h} \cdot \frac{\partial I_h}{\partial O_i} \cdot \frac{\partial O_i}{\partial I_j} \right)$$

$$= \sum_{h=0}^{H} \left(\frac{\partial E_t}{\partial I_h} \cdot \frac{\partial (W_{h0} \cdot O_0 + \ldots W_{hi} \cdot O_i + \ldots W_{hP} \cdot O_P)}{\partial O_i} \right) \cdot f'(I_i)$$

$$= \sum_{h=0}^{H} \left(\frac{\partial E_t}{\partial I_h} \cdot W_{hi} \right) \cdot f'(I_i)$$

d' où

$$\boxed{\frac{\partial E_t}{\partial W_{ij}} = O_j \cdot \Sigma_{h=0}^{H} \left(\frac{\partial E_t}{\partial I_h} \cdot W_{hi} \right) \cdot f'(I_i)} \quad (6)$$

La sommation h portant sur les neurones sur lesquels le neurone i envoie des connexions (voir figure 7).

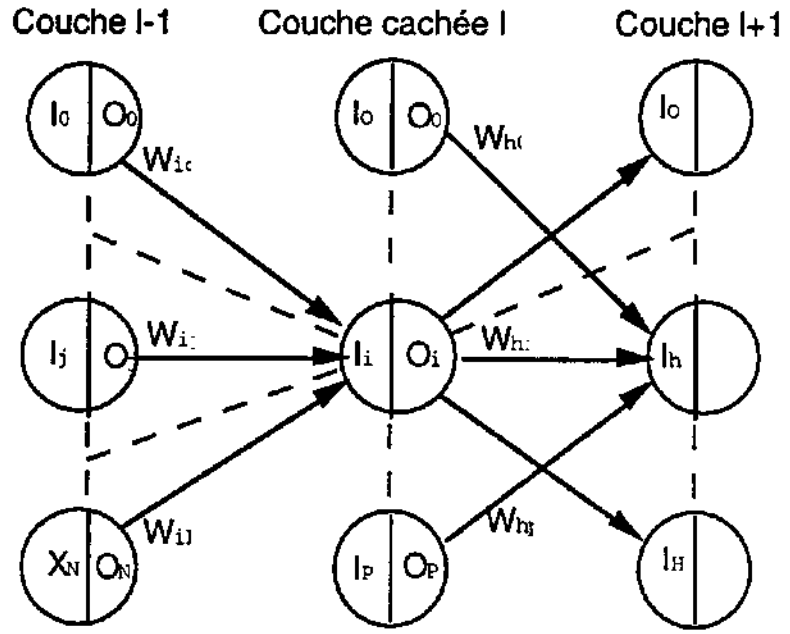

figure 7: Ajustement de Wij sur une couche cachée

$\frac{\partial E_t}{\partial I_h}$ ayant été calculé, l'erreur se propage ainsi de la couche de sortie vers la couche d'entrée ce qui provoque la rétropropagation du gradient (back-propagation).

<u>On peut résumer la phase d'apprentissage par cet algorithme:</u>

1) Initialiser tous les poids du réseau aléatoirement
2) Choisir et présenter un couple (vecteur entrée X vecteur sortie S) de la fonction à estimer
3) Calculer le vecteur sortie du réseau avec les formules (1) et (2)
4) Adapter les poids avec (4), (5) et (6)
5) Répéter à partir de l'étape 2

3.4) Comportement de l'erreur

Partant de poids aléatoires Wo, qui ne sont pas des minima de l'erreur E_t, les formules (4), (5) et (6), trouvées ci-dessus vont modifier ces poids de façon à ce que E_t décroisse et tende vers un minimum global.
Dans le cas où la fonction à estimer est linéaire, l'erreur atteint un seul minimum garantissant la convergence des coefficients.
Si cette fonction est non linéaire, il peut exister un nombre important de minima locaux, et il n'est pas garanti que E_t convergera vers un minimum global (voir figure 8).

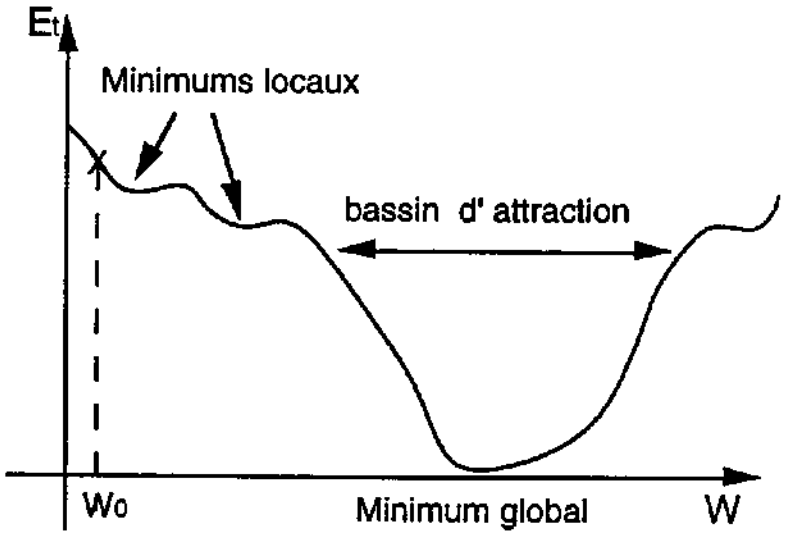

figure 8: Allure de l'erreur E_t en fonction d' un poids W

Cet inconvénient peut être contourné en augmentant le nombre de neurones de la couche cachée ou en effectuant une initialisation telle que la valeur de départ W_0 soit dans le bassin d'attraction du minimum global (voir figure 8).
De plus, le gradient calculé pour chaque couple (vecteur entrée X vecteur sortie S) ('pattern learning') ne représente pas nécessairement le gradient total.
Une technique, appelée batch learning, peut être utilisée. Elle consiste à calculer le gradient de façon à minimiser l'erreur E calculée sur un échantillon de couples (vecteur entrée X vecteur sortie S). Cela donne une estimation plus réaliste du gradient (pour plus de détails voir [8]).

$$E = \Sigma_{t=0}^{T} E_t$$

E représente l'erreur moyenne commise par le réseau lorsqu'on lui a présentée T vecteurs.

4) Résumé

En commande de processus, le réseau le plus utilisé est le réseau multicouche. Celui-ci, composé de neurones ayant pour fonction d'activation une sigmoide, est réglé par l'algorithme de rétropropagation du gradient lors de la phase d'apprentissage.

Le réseau est traversé par deux informations:

_ X présenté à l' entrée donne, par propagation avant, le vecteur Y qui est une estimation de S (vecteur de la fonction désirée)

_ S présenté à la sortie du réseau permet, par propagation arrière de l'erreur, d'ajuster les poids de la matrice de connexions par l'algorithme de rétropropagation du gradient.

On peut relever les inconvénients suivants:

_ Il n' y a pas de méthode connue pour déterminer le nombre de neurones et le nombre de couches pour une fonction donnée à estimer. Ceci est surement lié au fait que la connaissance a-priori est difficilement implantable dans le réseau

_ Comme le réseau est initialisé avec des poids pris aléatoirement, ceci nécessite une période d'entraînement longue pour assurer la convergence de l'erreur E vers un minimum.

Cette dernière consiste à présenter au réseau un vecteur entrée et le vecteur sortie correspondant désiré. L'algorithme de rétropropagation du gradient ajuste les poids pour que le réseau de neurones donne en sortie la réponse désirée.

PARTIE 3:

UN EXEMPLE DE COMMANDE PAR RÉSEAU DE NEURONES: LA COMMANDE ADAPTATIVE

1) Rappels sur la commande adaptative

Les commandes classiques, c'est à dire à retour d'état, critère quadratique, ... sont utilisées pour réduire l'effet des perturbations sur les variables à réguler: température, vitesse...
Elles se dégradent très vite dès lors que l'écart entre les paramètres réels du processus et les paramètres estimés de son modèle devient non négligeable. Cet écart peut être du, par exemple, à une approximation sur les équations différentielles, au vieillissement du processus, ..., de toute façon à un manque d'information.
L'intérêt de l'utilisation du réseau de neurones est qu'il va pouvoir apprendre de lui-même l'information non connue de l'utilisateur, et la réactualiser si les paramètres du système sont sensibles à des variations.
Les deux structures les plus utilisées en commande adaptative classique sont la Commande Auto-ajustable (Self Tuning Control _ S.T.C.) et la Commande Adaptative avec Modèle de Référence (Model Reference Adaptative Control _ M.R.A.C.).
Elles reposent sur l'hypothèse qu'il existe un régulateur de structure donnée qui puisse assurer la réalisation des performances désirées pour toutes les valeurs possibles des paramètres du processus. La boucle de régulation permet alors de trouver les bonnes valeurs des paramètres de ce régulateur. Le calcul du régulateur suppose d'avoir une bonne connaissance du modèle dynamique du processus et des performances désirées qui sont généralement représentées par un modèle de référence.
Ces deux techniques de commande adaptative sont relativement simples à mettre en œuvre sur des systèmes linéaires continus et discrets. En effet, des lois adaptatives stables peuvent être déterminées sous réserve de disposer d'informations sur la fonction de transfert du processus à commander.
Par contre, très peu de travaux ont été effectués sur la commande adaptative de processus décrits par des équations différentielles non linéaires. C'est dans la commande de tels processus que les propriétés de non linéarités, d' apprentissage et d' adaptation des réseaux de neurones peuvent être utilisées pour créer une commande neuronale.

2) Commande adaptative avec modèle de référence par réseaux de neurones

2.1) Principe

Les paramètres du régulateur peuvent être ajustés en une seule étape. Le mécanisme d'adaptation consiste, alors, à minimiser la différence entre la performance réelle et celle désirée définie par un modèle de référence.
On a alors une commande adaptative directe, la commande adaptative avec modèle de référence de la figure 9 en fait partie.

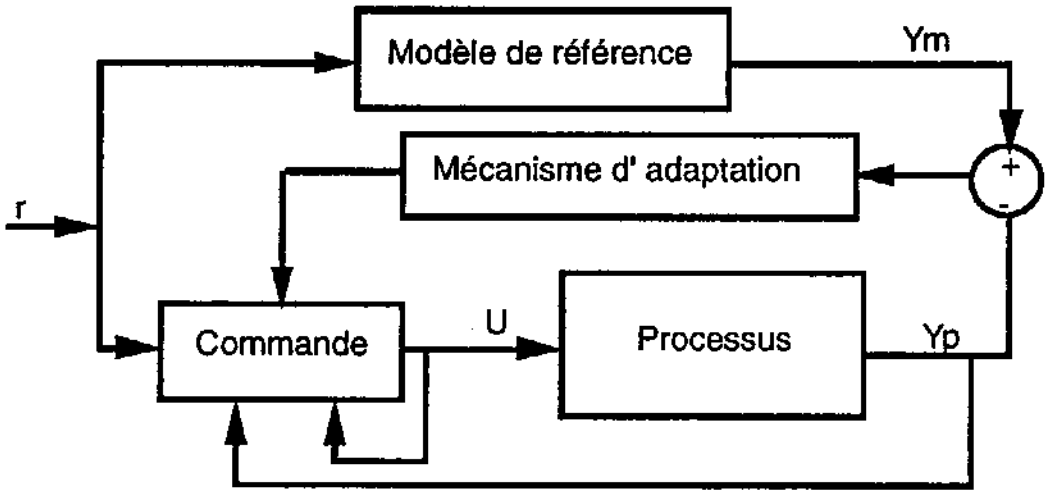

figure 9: Commande adaptative avec modèle de référence

Ici, les paramètres de la commande sont directement ajustés pour réduire la norme de l'erreur (Ym - Yp) entre la sortie du modèle de référence et la sortie du processus.
L'architecture de la commande par réseau de neurones s'en déduit très facilement (voir figure 10).
Le réseau de neurones Ni exécute l'identification du processus, mais n'entre pas en compte dans la commande du processus. Il permet, seulement, le bon fonctionnement de Nc (c'est à dire en fait de rétropropager l'erreur en sortie du processus sur le réseau Nc, voir partie 3: 2.2.3).

Remarque:
Les T.D.L. (Tapped Delay Line) sont des lignes à retard permettant de mémoriser les valeurs précédentes des variables.

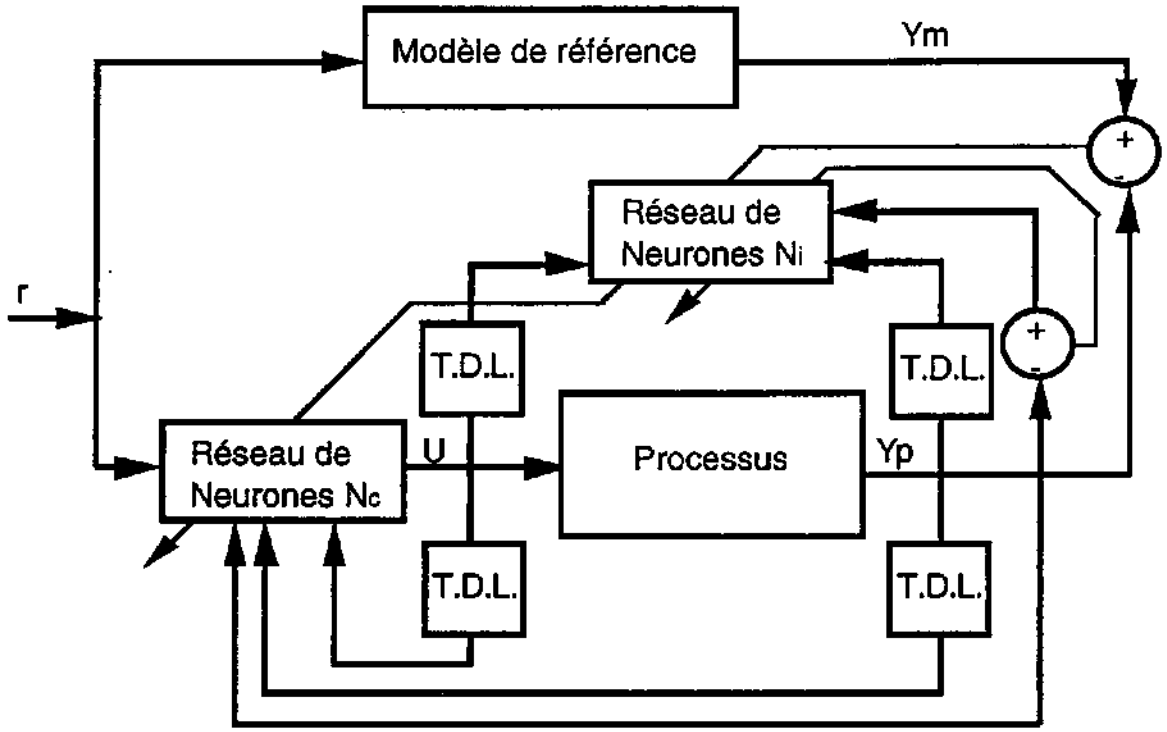

figure 10: commande directe par réseau de neurones

2.2) Apprentissage

2.2.1) Nécessité de l'apprentissage

Au départ, les poids du réseau sont initialisés aléatoirement. Si on commande le processus immédiatement, les deux réseaux vont passer un certain temps à apprendre, c'est à dire à ajuster leur poids de façon à ce que leurs sorties atteignent les valeurs optimales permettant une commande adaptative du processus.
Le temps de convergence de ces coefficients peut être très long et, durant cette période, le processus peut recevoir des commandes tout à fait incohérentes.
On ne peut prendre de tels risques, et, avant de contrôler directement le processus, on va initialiser les poids, lors d'une phase d'apprentissage, à des valeurs proches de celles à atteindre lors de la phase de commande (voir partie 2: 3.4).

2.2.2) Apprentissage du réseau appliqué à l'identification

Cette phase consiste à apprendre au réseau Ni les caractéristiques, initialement inconnues, du processus. Son utilisation va permettre d'apporter une estimation de la sortie du processus lors du calcul de la commande adaptative.

L'apprentissage consiste d'abord à présenter une entrée Uk simultanément au réseau Ni et au processus (voir figure 11).
Ensuite, l'erreur entre la sortie prédite par Ni et celle du processus est rétropropagée par l'algorithme du gradient pour ajuster les poids du réseau selon les formules trouvées dans la partie 2: 3.3):

$$W_{ij}(k+1) = W_{ij}(k) - \eta \cdot \frac{\partial E_t}{\partial W_{ij}} \quad (4) \qquad \frac{\partial E_t}{\partial W_{ij}} = -(S_i - Y_i) \cdot f'(I_i).O_j \quad (5)$$

$$\frac{\partial E_t}{\partial W_{ij}} = O_j \cdot \Sigma_{h=0}^{H} \left(\frac{\partial E_t}{\partial I_h} \cdot W_{hi} \right) \cdot f'(I_i) \quad (6)$$

avec $S_i = Y(k+1)$ et $Y_i = \hat{Y}(k+1)$

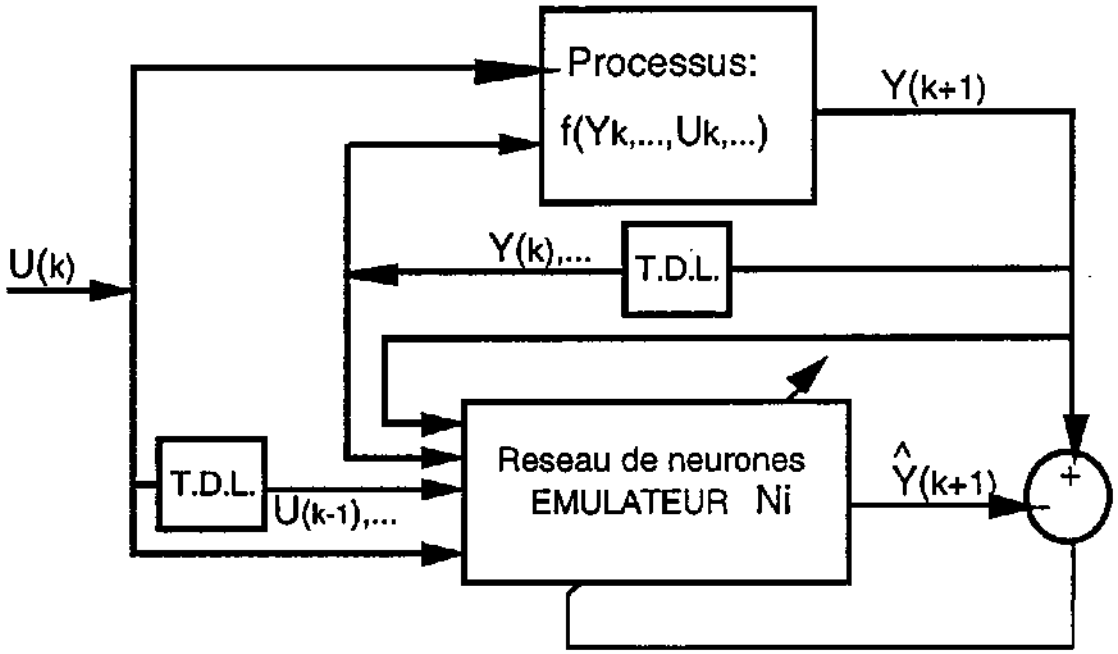

figure 11: Apprentissage pour l'identification

Le processus étant décrit par le modèle discret non linéaire suivant:

$Y(k+1) = f(Yk, Yk-1, ...,Yk-r ,Uk, Uk-1, ...,Uk-s)$

où Yk, Yk-1, ...,Yk-r ,Uk, Uk-1, ...,Uk-s sont, respectivement, les valeurs de la sortie et de l'entrée du processus, mémorisée dans les lignes à retard.

Le réseau Ni subit un entraînement en sortie et peut, ensuite, être utilisé comme émulateur. Le problème qui apparaît, lorsqu'on ne connait pas le processus, est la détermination de r et de s, ou, autrement dit, de l'ordre du système. Cela se traduit par la détermination du nombre de neurones de la couche d'entrée du réseau.
Deux procédures sont possibles:

- soit on utilise un nombre très grand de neurones sur la couche d'entrée et la procédure d'apprentissage affaiblira les poids des entrées (variables) non utilisées lors de l'identification du processus.
- soit on procède par essais successifs, en augmentant chaque fois le nombre de neurones de la couche d'entrée jusqu'à l'obtention d'un bon modèle.

On peut arrêter cette procédure d'apprentissage, pour passer à la phase de commande, quand l'erreur entre les sorties devient nulle ou inférieure à une limite acceptable.

Etant donné que nous avons une commande adaptative, cette phase d'identification, va être poursuivie durant la commande. Ceci permettra une adaptation du réseau à des conditions changeantes du système (processus + environnement) et une amélioration du modèle (affinement des réponses).

2.2.3) Apprentissage du réseau appliqué à la commande

Si Uk représente le vecteur des entrées du processus, Yk le vecteur des sorties du processus et Dk le vecteur des sorties désirées du processus, le problème général d'une commande est de trouver Uk pour avoir Yk = Dk, c'est à dire de résoudre le problème de la dynamique inverse du processus.
Ici, le vecteur des sorties désirées Dk provient de la sortie d'un modèle de référence qui rassemble les performances désirées du processus.
La commande Uk est réalisée (voir figure 12) en ajustant les poids du second réseau Nc sur le réseau Ni entraîné précédemment. L'utilisation d'un second réseau Ni, comme émulateur, permet d'éviter de détériorer le processus qui risquerait de recevoir des commandes aberrantes lors de l'apprentissage de Nc.

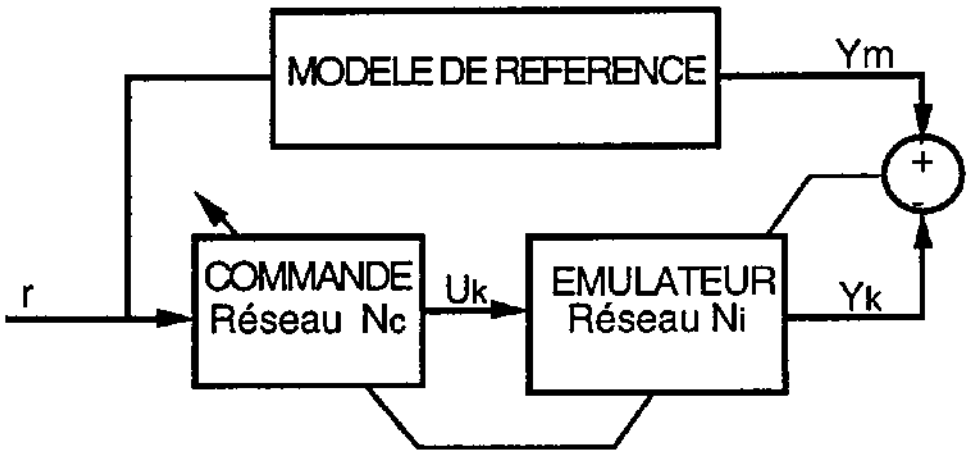

figure 12: apprentissage pour la commande

L'objectif de l'apprentissage de Nc est de fixer ses poids pour minimiser l'erreur:
(Ym - Yk).
Pour cela, nous avons besoin de connaître l'erreur commise sur la sortie de Nc: (Ud - Uk). Celle-ci n'étant pas disponible, on voit l'intérêt d'avoir effectué l'entraînement sur l'émulateur car l'erreur (Ym - Yk) est connue et va pouvoir être rétropropagée à travers ce réseau Ni, dont les poids fixés précédemment restent constants.
Ainsi, l'utilisation d'un réseau de neurones comme émulateur permet seulement de translater l'erreur sur la sortie (Ym - Yk) en erreur sur la commande (Ud - Uk) mais, n'entre pas en compte dans le calcul de la commande.

Cette dernière peut alors entraîner le réseau Nc par:

$$W_{ij}(k+1) = W_{ij}(k) - \eta \cdot \frac{\partial E_t}{\partial W_{ij}} \quad (4) \qquad \frac{\partial E_t}{\partial W_{ij}} = -(S_i - Y_i) \cdot f'(I_i).O_j \quad (5)$$

$$\frac{\partial E_t}{\partial W_{ij}} = O_j \cdot \sum_{h=0}^{H} \left(\frac{\partial E_t}{\partial I_h} \cdot W_{hi} \right) \cdot f'(I_i) \quad (6)$$

avec $S_i = U_d$ et $Y_i = U_k$

Au terme de cet apprentissage, le réseau Nc est parfaitement capable de commander le processus.

Remarque:
L'utilisation d'un réseau de neurones comme émulateur peut être évitée sous réserve qu'une étude variationnelle sur le processus puisse être réalisée (ce qui est, ici, difficilement envisageable étant donné que l'on a un processus non-linéaire).
En effet, on aurait

$$\frac{\partial E_t}{\partial W} = \frac{\partial Y_k}{\partial W} = \frac{\partial Y_k}{\partial U_k} \cdot \frac{\partial U_k}{\partial W}$$

On peut évaluer $\frac{\partial Y_k}{\partial U_k}$ par $\frac{\Delta Y_k}{\Delta U_k}$ sous réserve que ces variations soient mesurables.

2.3) Mise en œuvre de la commande

2.3.1) Description

Le processus non linéaire inconnu ayant été identifié par Ni, la commande à appliquer à ce processus ayant été apprise par Nc, ces deux réseaux, ainsi initialisés peuvent, alors, être insérés dans la structure globale de la figure 13 pour commander le processus.

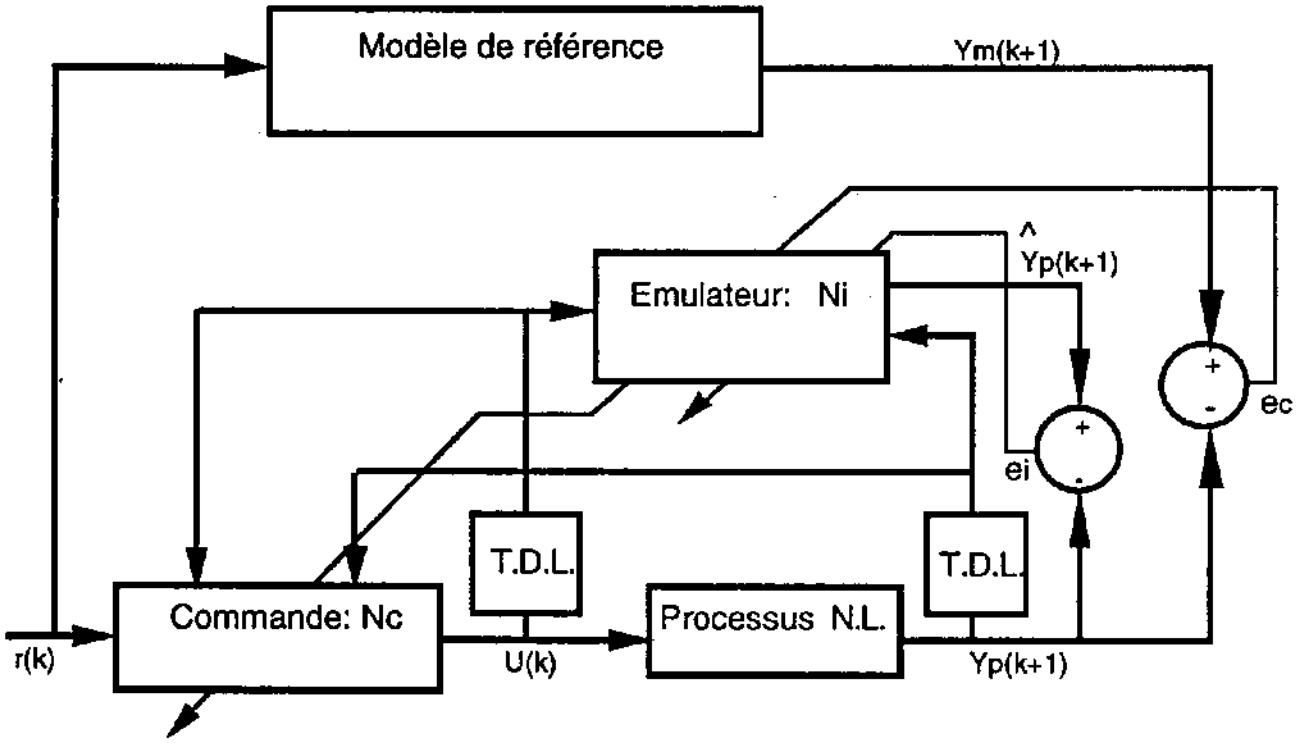

figure 13: commande adaptative avec modèle de référence par réseau de neurones

L'adaptation de la commande s'effectue selon la procédure suivante:

1) A l'instant k, Ym(k) et Yp(k) sont échantillonnés et leurs valeurs précédentes sont décalées d'une période d'échantillonnage
2) (Yp(k-1), ... , U(k-1), ...) sont pris en compte par Ni
3) Les poids de Ni sont actualisés pour minimiser ei
4) (Yp(k-1), ... , U(k-1), ...) sont pris en compte par Nc
5) Les poids de Nc sont actualisés pour minimiser ec translatée à travers Ni dont les poids demeurent constants
6) La sortie U(k) de Nc est appliquée à l'entrée de commande du processus

Les deux exemples suivant, tirés de [9], illustrent le comportement de cette commande.

2.3.2) Exemples d'application

2.3.2.1) Premier exemple

Le processus a été simulé par la fonction de transfert suivante:

$$Yp(k+1) = \frac{Yp(k).Yp(k-1).(Yp(k)+2.5)}{1+Yp(k)+Yp(k-1)} + U(k)$$

$$= f(Yp(k), Yp(k-1)) + U(k)$$

On souhaite que la sortie du processus se comporte comme celle du modèle de référence suivant :

$$Ym(k+1) = 0,6.Ym(k) + 0,2.Ym(k-1) + r(k)$$

avec pour entrée de référence: $r(k) = \sin(2\Pi.k/25)$

On peut essayer de trouver l'expression de la commande. L'erreur de sortie, définie par Ym(k+1) - Yp(k+1) (voir figure 13), doit tendre vers 0 pour k tendant vers l'infini. Dans ce cas, la commande s'écrit:

$$U(k) = - f(Yp(k), Yp(k-1)) + Yp(k+1)$$
$$= - f(Yp(k), Yp(k-1)) + 0,6.Yp(k) + 0,2.Yp(k-1) + r(k)$$

La fonction f(Yp(k), Yp(k-1)) étant inconnue, elle va être estimée par un réseau de neurones Ni, d'où l'expression de la commande établie par le réseau de neurones:

$$U(k) = - Ni(Yp(k), Yp(k-1)) + 0,6.Yp(k+1) + 0,2.Yp(k-1) + r(k)$$

L'objectif du réseau de neurones Nc est de simuler cette équation. Cette dernière peut permettre de construire une architecture possible du réseau (voir figure 14). On utilise ainsi la connaissance que l'on a-priori sur la commande pour améliorer les performances de la commande par réseau de neurones. Nc va devoir comporter un sous-réseau de neurones Ni (représenté par un cercle hachuré figure 14) qui va estimer le comportement des sorties du processus, ainsi que des neurones linéaires (représentés par des carrés).

Le schéma complet de la commande, illustré figure 15, est un peu plus compliqué que celui de la figure 13 car on a remplacé les neurones linéaires de la commande de la figure 14 par des sommateurs. Cette procédure est un moyen d'optimiser l'utilisation des réseaux de neurones.

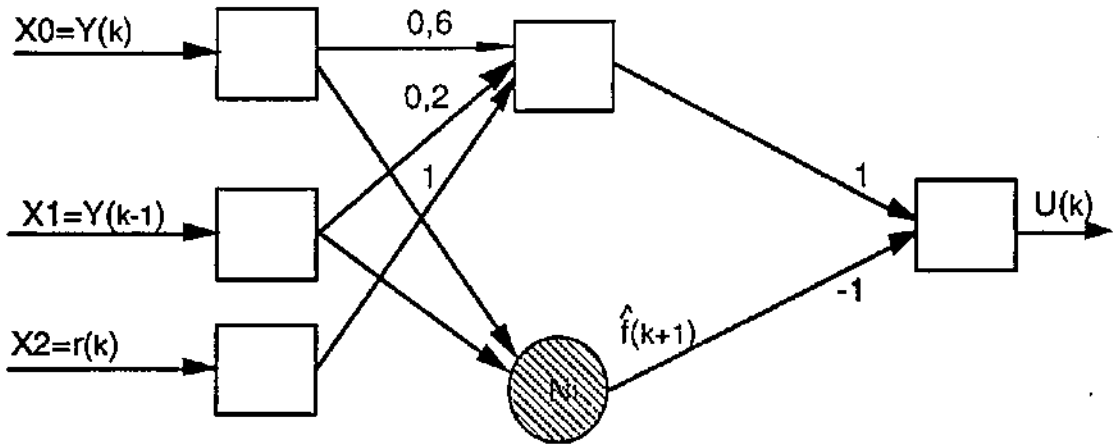

figure 14: Architecture du réseau Nc pour le premier exemple

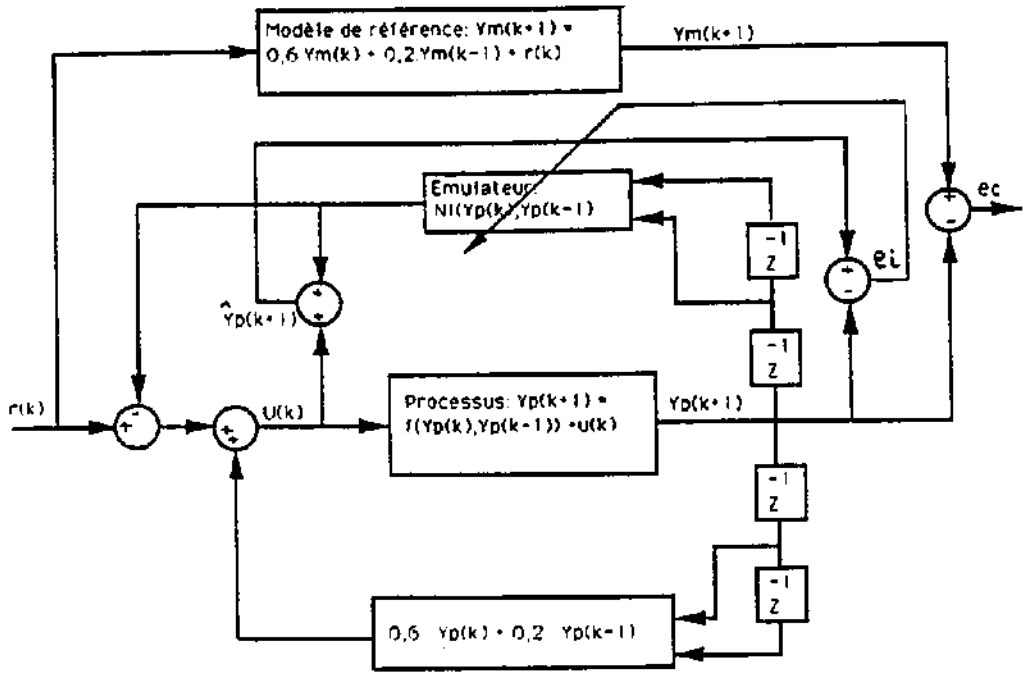

figure 15: Schéma de la commande avec modèle de référence par réseaux de neurones

Les sorties du modèle et du processus ont été relevées figure 16:
a) en n'appliquant pas la commande
b) en appliquant la commande

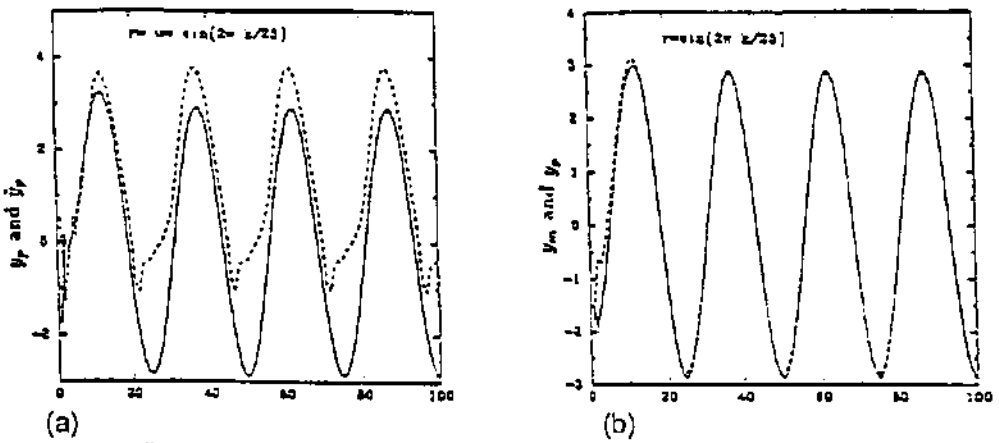

figure 16: Résultats

On observe que la sortie du processus suit bien la sortie du modèle de référence.

Remarque:
Pour cet exemple, la commande n'est pas strictement réalisée par un réseau de neurones. En effet, le processus étant linéaire par rapport à U, il n'était pas indispensable d'utiliser un réseau de neurones Nc pour le commander. Le deuxième exemple illustre le cas où la commande est directement fournie par le réseau de neurones étant donné que le processus est non linéaire par rapport à U.

2.3.2.2) Deuxième exemple

La fonction de transfert du second processus est décrite par une relation non linéaire par rapport à $U(k)$ et $Yp(k)$:

$$Yp(k+1) = \frac{Yp(k)}{1 + Yp(k)} + U(k)^3$$

En posant $f() = \frac{Yp(k)}{1 + Yp(k)}$ et $g() = U(k)^3$, on a plus simplement: $Yp(k+1) = f() + g()$

On a choisi le modèle de référence suivant: $Ym(k+1) = 0,6 \cdot Yp(k) + r(k)$

$U(k)$ doit annuler l'erreur de sortie $Yp(k) - Ym(k)$ pour k tendant vers l'infini.

$$U(k) = g^{-1}(Yp(k+1) - f(Yp(k))) = g^{-1}(Ym(k+1) - f(Yp(k)))$$

f et g sont estimées par le réseau de neurones Ni(Yp(k), U(k)), et g^{-1} par Nc de façon à ce que: $Nc(g(Nc(r(k)))) = r(k)$

On a alors $U(k) = Nc(0,6 \cdot Yp(k) + r(k) - f(Yp(k)))$

De cette équation, on peut déduire que Nc va devoir comporter des sous-réseaux, représentés par des cercles hachurés figure 17, qui estiment f, g et g^{-1}.

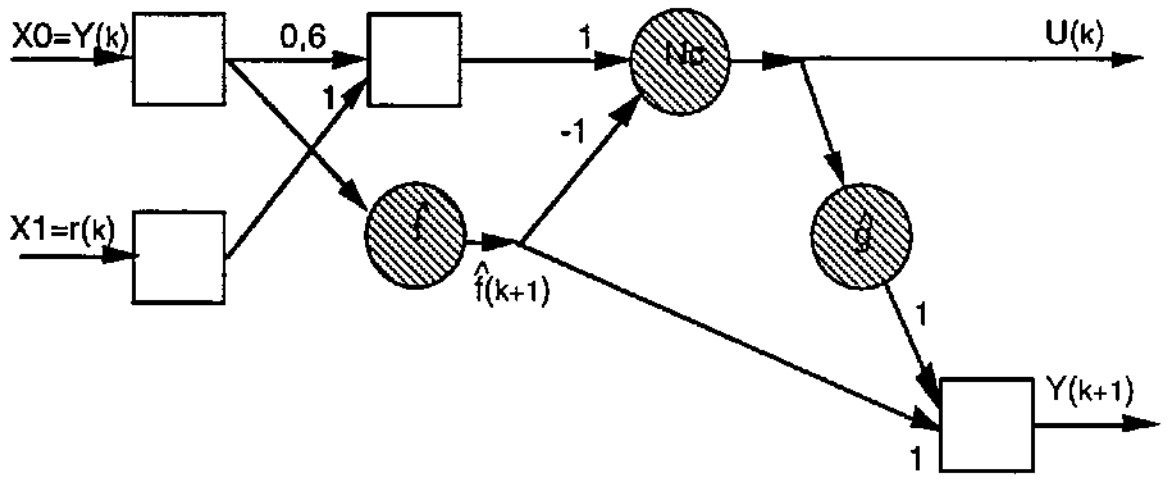

figure 17: Architecture du réseau utilisé pour la commande

Le schéma complet de cette commande est illustré figure 18

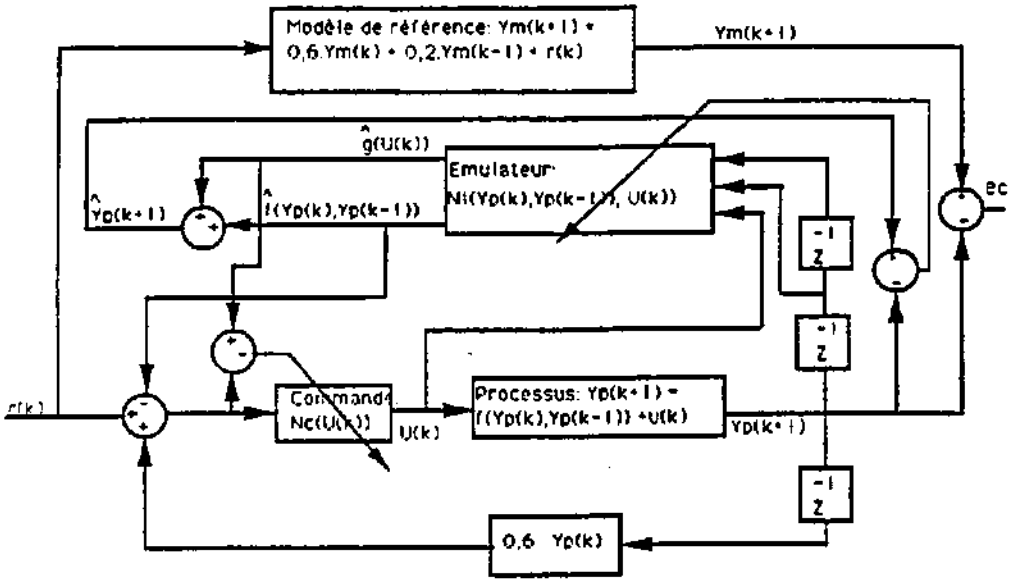

figure 18: Schéma de la commande avec modèle de référence par réseaux de neurones

Nc est constitué de deux couches cachées de 20 et 10 neurones.
L'identification a été réalisée en appliquant 25000 valeurs aléatoires en entrées comprise entre -4 et +4 avec η = 0,01.
L'apprentissage pour la commande a été réalisé en appliquant 100000 valeurs aléatoires en entrées comprises entre -2 et +2 avec η = 0,1.

Les sorties du modèle et du processus ont été relevées
a) en n'appliquant pas la commande
b) en appliquant la commande

Les courbes sont représentées ci-dessous figure 19.
On peut observer une bonne maîtrise de la sortie du processus malgré sa forte non linéarité.

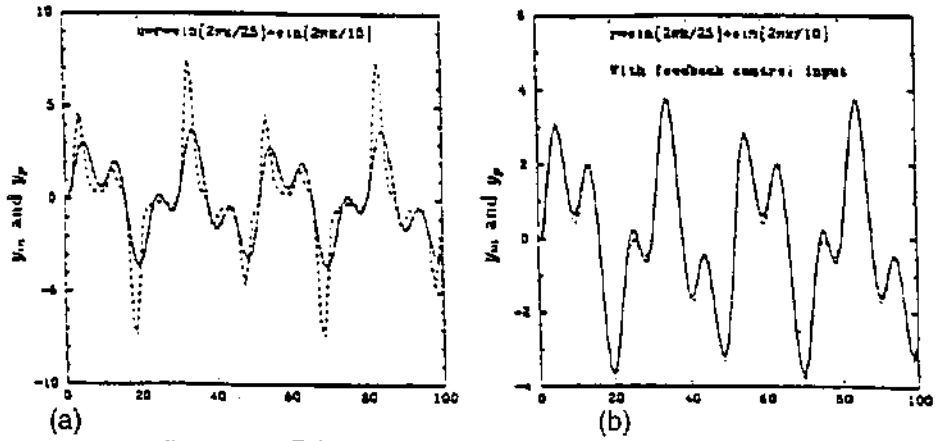

figure 19: Résultats

3) Commande Auto-Ajustable par réseau de neurones

3.1) Présentation

Les paramètres du régulateur peuvent être ajustés en deux étapes. Le modèle du processus est remplacé par un modèle estimé en temps réel et la commande consiste à effectuer

1) une identification des paramètres du processus à partir des mesures entrées-sorties
2) un calcul des paramètres du régulateur à partir des paramètres estimés

On obtient alors une commande indirecte, la commande auto-ajustable de la figure 20 en fait partie.

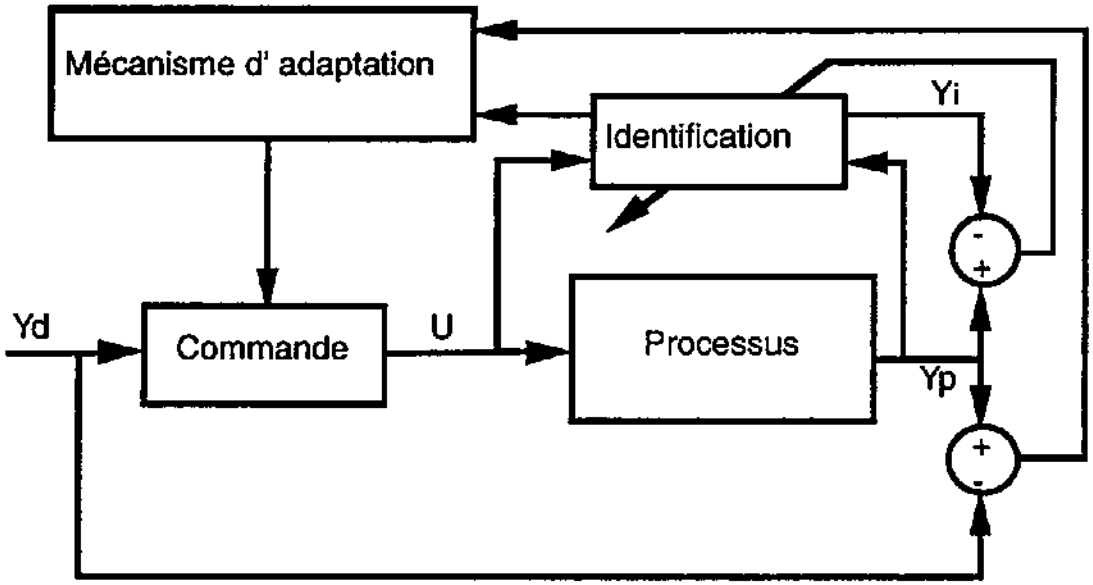

figure 20: Commande auto-ajustable

Les paramètres du processus sont estimés par un vecteur Yi à chaque instant k et le vecteur paramètre U(k) de la commande est choisi en supposant que Yi représente la valeur vraie Yp du vecteur paramètre du processus. Calquée sur cette philosophie, la commande par réseau de neurones est représentée figure 21.

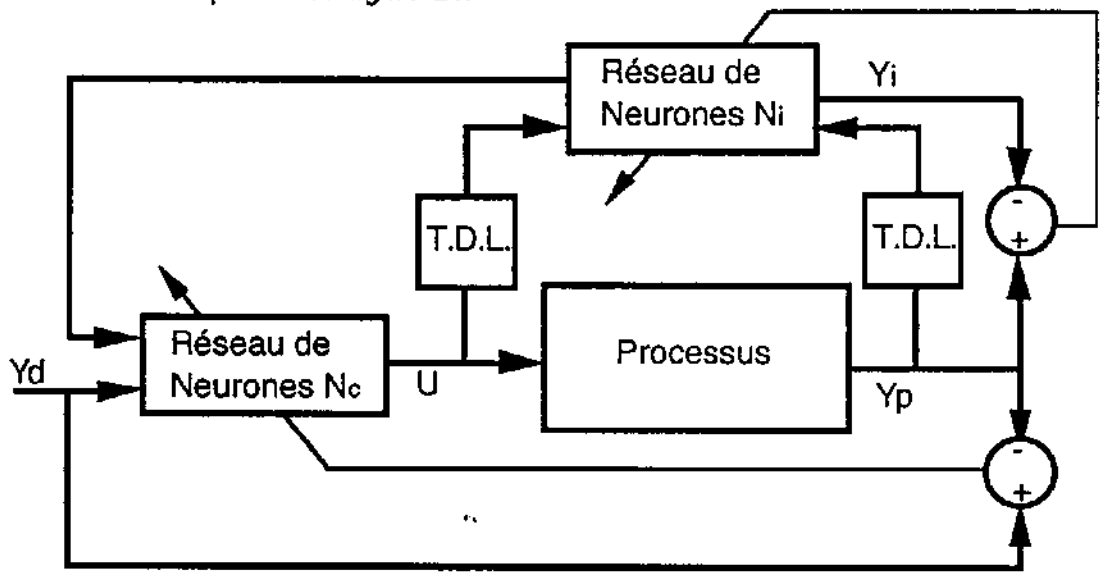

figure 21: Commande auto-ajustable par réseau neuronal

3.2) Apprentissage

3.2.1) Apprentissage du réseau appliqué à l'identification

La procédure d'identification est exactement identique à celle décrite dans la partie 3: 2.2.2) (voir figure 22).
Afin de simplifier les explications, nous allons traiter le cas particuliers où le processus peut être décrits par des équations non-linéaires du type:

$$Y_{k+1} = f(Y_k, Y_{k-1}, \ldots, Y_{k-r}, U_{k-1}, U_{k-2}, \ldots, U_{k-s}) + g(U_k) \quad (7)$$

où Y est la sortie du processus, U, sa commande.

Cette équation permet de déduire que le réseau Ni sera composé de 2 sous-réseaux tels que l'un estime g et l'autre f (voir figure 23).

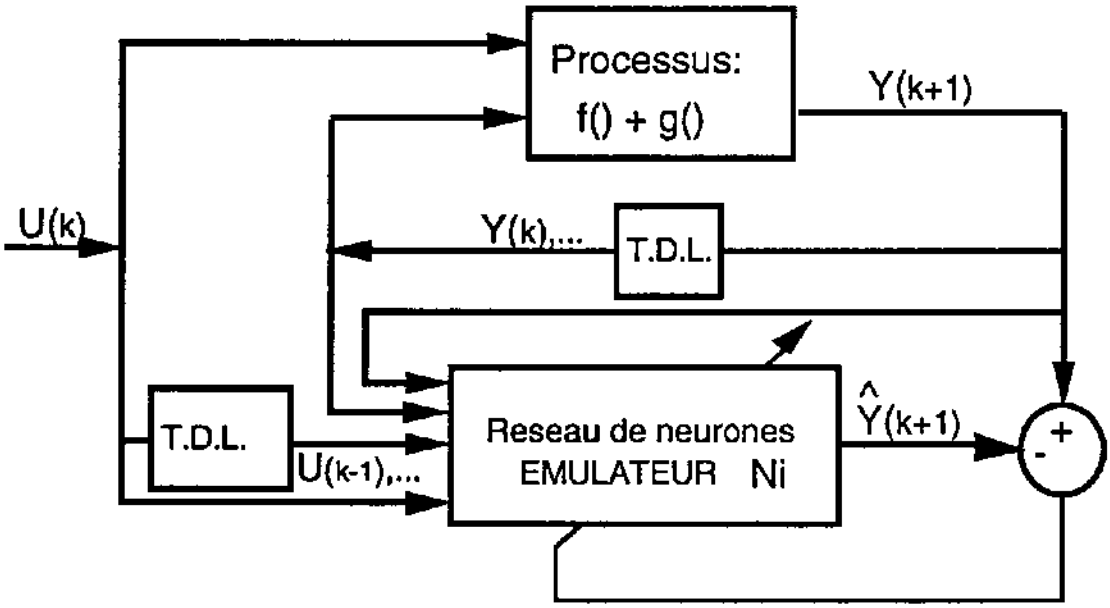

figure 22: Apprentissage pour l'identification

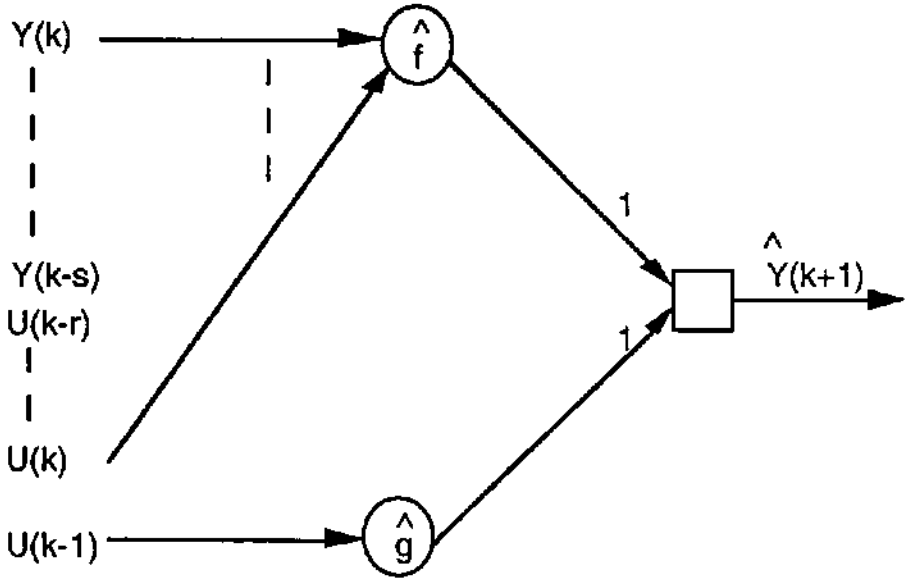

figure 23: Architecture du réseau de neurones Ni utilisé pour l'identification

3.2.2) Apprentissage du réseau appliqué à la commande

Si on connaît f et g (en fait, les caractéristiques du processus), alors la commande:
$U_k = g^{-1}\ [Y_d - f()]$
appliquée au processus:

$$Y_{k+1} = f(Y_k, Y_{k-1}, \ldots, Y_{k-r}, U_{k-1}, U_{k-2}, \ldots, U_{k-s}) + g(U_k) \quad (7)$$

va commander sa sortie Yk+1 à la valeur désirée Yd.

Cette phase va donc permettre au réseau Nc d'apprendre la commande:

$$U_{(k)} = N_c\ [Y_{d(k+1)} - f()] \quad (8)$$

L'architecture du réseau Nc s'en déduit facilement et est représenté figure 24.

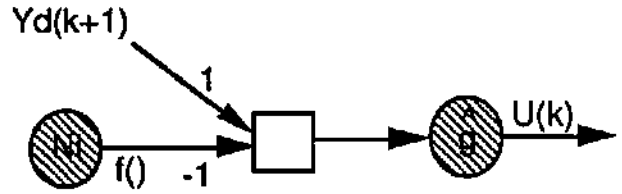

figure 24: Architecture du réseau de neurones Nc utilisé pour la commande

Le réseau Ni ayant appris la relation entre le vecteur entrée U du processus et son vecteur sortie Yp, on va entraîner le réseau Nc pour qu'il apprenne l' équation (8).
Tout comme dans la partie 3: 2.2.3), on va se servir du réseau Ni comme émulateur pour préserver le processus de détériorations possibles (figure 25).

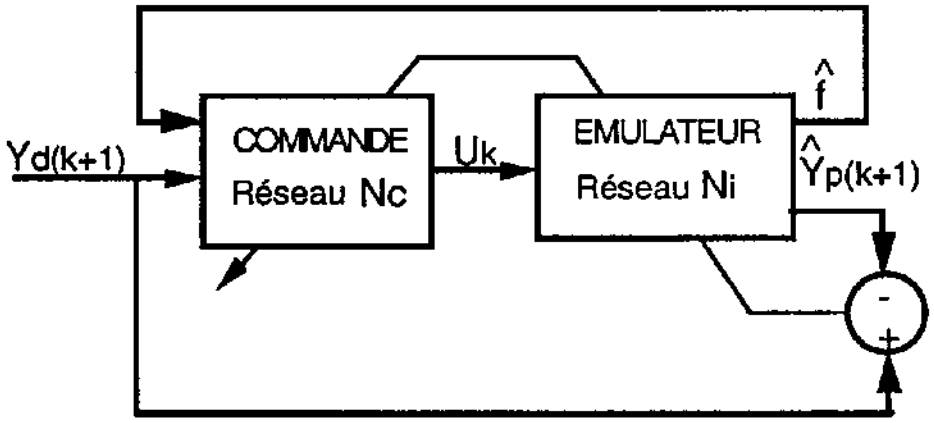

figure 25: Apprentissage de Nc pour la commande

L'erreur à minimiser est maintenant: (Yd - Yp), cette dernière est translatée à travers le réseau Ni en erreur sur la commande (Ud -Uk) afin de modifier les poids de Nc selon:

$$W_{ij}(k+1) = W_{ij}(k) - \eta\ .\ \frac{\partial E_t}{\partial W_{ij}} \quad (4) \qquad \frac{\partial E_t}{\partial W_{ij}} = -\ (S_i - Y_i)\ .\ f'(I_i).O_j \quad (5)$$

$$\frac{\partial E_t}{\partial W_{ij}} = O_j\ .\ \Sigma_{h=0}^{H}\ \left(\frac{\partial E_t}{\partial I_h}\ .\ W_{hi}\right)\ .\ f'(I_i) \quad (6)$$

avec $S_i = U_d$ et $Y_i = U_k$

3.3) Mise en œuvre de la commande

3.3.1) Description

En résumé, le rôle de la commande auto-ajustable va être:
_ d'estimer f et g en temps réel et de façon à modifier cette estimation au cas où le processus serait sensible à des variations de comportement
_ de construire la commande:

$$U_{(k)} = N_c\,[Y_{d(k+1)} - f()] \quad (8)$$

La commande par réseau de neurones est représentée figure 26.

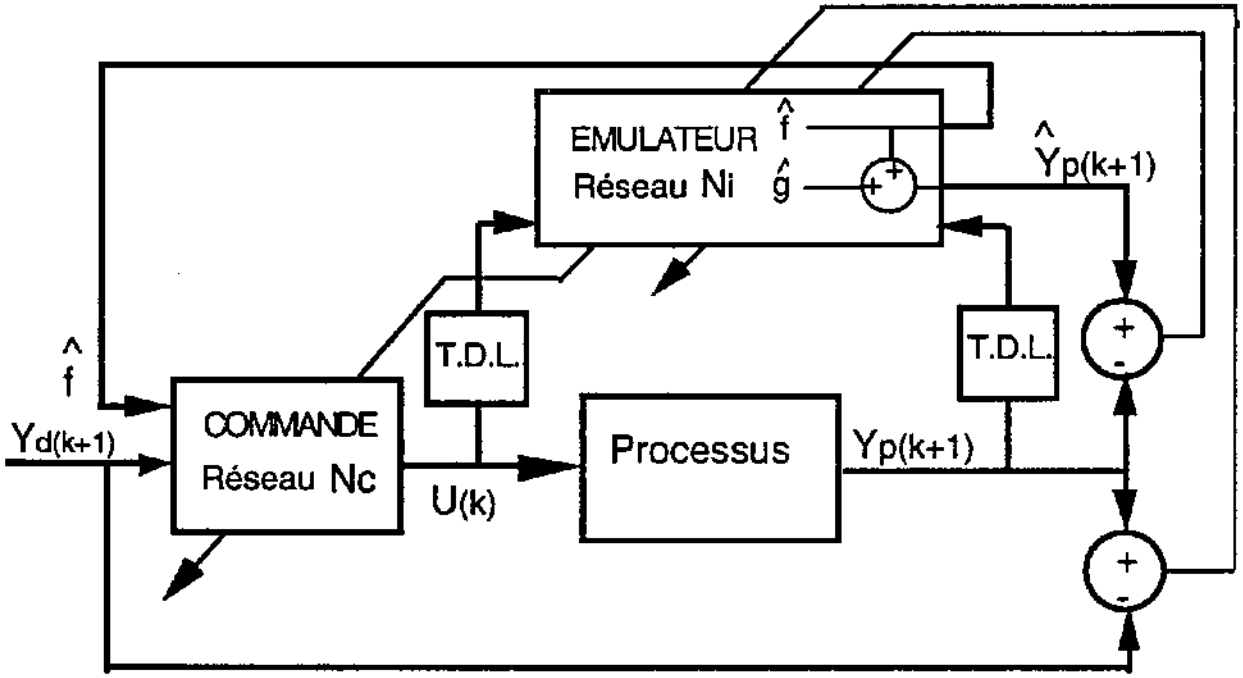

figure 26: Schéma de la commande auto-ajustable par réseau de neurones

3.3.2) Exemple

Cet exemple d'application, traité par [10] a été effectué sur un processus dont la fonction de transfert, supposée inconnue, est la suivante:

$$Yp(k+1) = 0.8\,.\,\sin(2.Yp(k)) + 1.2\,.\,U(k)$$

La commande se calcule facilement:

$$U(k) = \frac{Yp(k+1) - 0.8\,.\,\sin(2.Yp(k))}{1.2}$$

$$U(k) = g^{-1}[\,Yd(k+1) - f(\,Yp(k)\,)\,]$$

$f(\,Yp(k)\,)$ va être estimée par Ni et g^{-1} [...] par Nc.

Ni comporte 2 couches cachées de 10 neurones et Nc est constitué d'un seul neurone (g étant constante).

La sortie désirée est composée d'une sinusoïde et d'un échelon et est échantillonnée 400 fois (k=400) sur une période.
La sortie désirée et celle du processus ont été relevées pour:

a) 0< k<400
b) 25000<k<25400
c) 9000<k<90400

Les résultats sont représentés ci-dessous figure 27, et permettent d'évaluer la qualité d'adaptation des poids par rapport au nombre de données échantillonnées fournies. En effet,on observe qu'en prenant 400 échantillons sur la sortie désirée, la sortie du processus commence à suivre la sortie désirée. En répétant cette procédure, la commande devient parfaite.

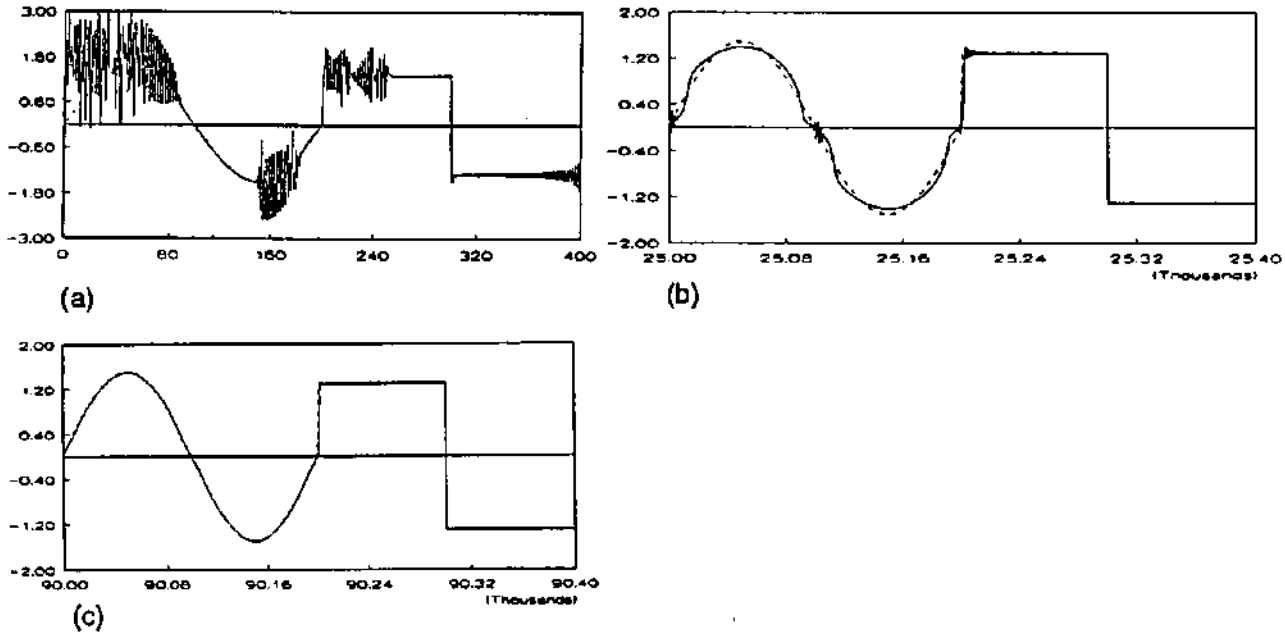

figure 27: Résultats obtenus par la commande auto-ajustable par réseau de neurones

PARTIE 4:

APPROXIMATION D'UNE FONCTION PAR UN RÉSEAU DE NEURONES

1) Introduction

L'intérêt de l'utilisation d'un réseau de neurones en automatique réside dans sa faculté d'approcher une fonction a priori inconnue en construisant une représentation interne de la fonction par modification de ses poids à partir de données entrées-sortie. En identification, la fonction à déterminer est la relation entrées-sorties du processus. En commande, le réseau doit déterminer la loi de commande qu'il faut appliquer au processus pour que ses sorties correspondent à celles désirées (voir partie 3: 2.2.3)).
Le problème posé par l'utilisation d'un réseau de neurones est la détermination de sa structure. C'est à dire:
_ la valeur des poids des connexions
_ la fonction d'activation de chaque neurone
_ le nombre de neurones dans la couche cachée

Pratiquement:
_ on fixe les poids aléatoirement, et, on les ajuste au cours d'une phase d'apprentissage, par l'algorithme du gradient.
_ on utilise des sigmoïdes comme fonctions d'activation car elles sont facilement dérivables.
_ on effectue plusieurs essais en augmentant le nombre de neurones à chaque fois jusqu'à obtenir une approximation convenable pour l'application envisagée.

Cette démarche nécessite énormément de temps, tant dans la phase de conception du réseau que dans la phase d'utilisation. Et, lorsque certaines informations sont disponibles sur le problème à résoudre, il serait souhaitable de les utiliser dans la détermination du réseau. Ceci a été partiellement fait dans les exemples cités partie 3: 2.3.2.1), 2.3.2.2) et 3.3.2.), on réduit, ainsi, la difficulté en obtenant un réseau de sous réseaux.
Cependant, il n'existe toujours pas de règles permettant de déterminer complètement le réseau qui est utilisé pour approcher une fonction.

Le paragraphe suivant vise à présenter les travaux effectués quant à la détermination de ces règles.

2) Travaux réalisés

En résolvant le 13ème problème de HILBERT, KOLMOGOROV [4] a démontré que n'importe quelle fonction f:

$$[0,1]^n \rightarrow R^n$$

pouvait être approchée par des superpositions d'un nombre fini de fonctions (g1, g2, ...) d'une variable:

$$\hat{f} = \Sigma_i \alpha_i . g_1\left[\Sigma_j \beta_j . g_2\left[\Sigma \ldots \left[\Sigma_k \beta_k . g_l(x)\right]\ldots\right]\right]$$

CYBENKO [5], HORNIK and al. [6] ont restreint ce théorème en prouvant sa validité au cas de la somme d'une même fonction d'activation (sigmoïde):

$$\hat{f} = \Sigma_{i=1}^{l} \alpha_i \cdot g(\beta_i . X)$$

Ils en ont déduit que les réseaux neuronaux comportant 3 couches pouvaient approcher n'importe quelle fonction continue.
Ce réseau possède trois couches:

- une couche d'entrée ayant autant de neurones linéaires que de variables d'entrée, ces neurones permettant de mémoriser ces variables.
- une couche cachée.
- une couche de sortie ayant autant de neurones linéaires que de variables de sortie, ces neurones effectuant la somme de ces fonctions sigmoïdes.

L'architecture d'un réseau de M neurones approchant une fonction à 2 entées-1 sortie, déduit de ces résultats est représenté figure 27.

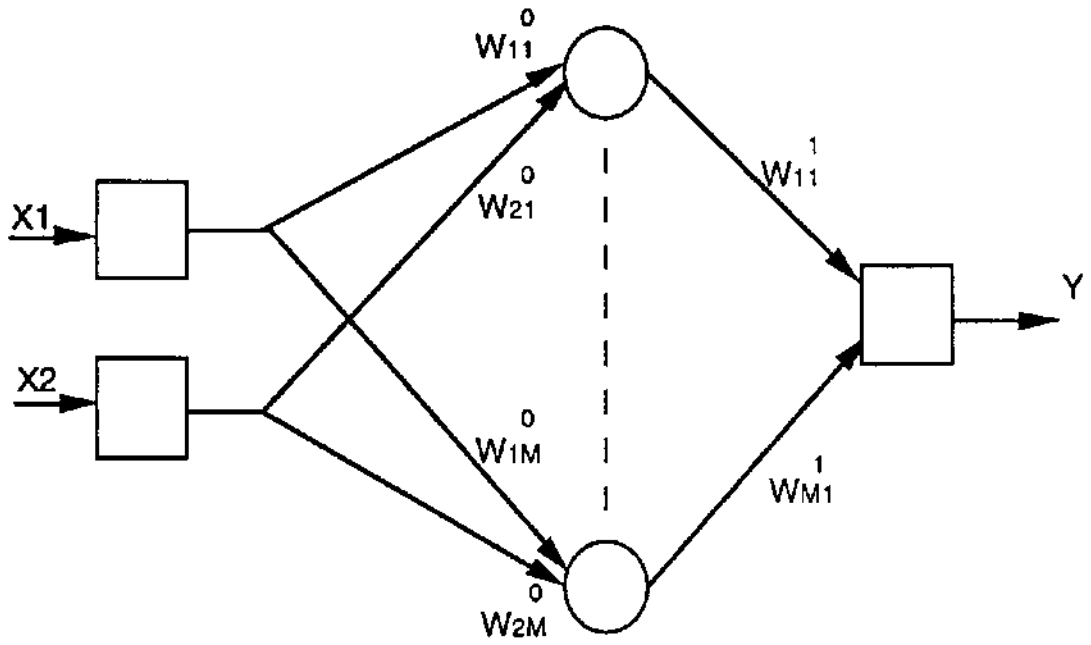

figure 27: Réseau de neurones à 2 entrées-1 sortie

Les formules (1) et (2) (partie 2 : 2.3)) permettent d'écrire l'expression de la sortie du réseau:

$$Y = \begin{bmatrix} W^1_{11} & \dots & W^1_{M1} \end{bmatrix} \cdot f\left(\begin{bmatrix} W^0_{11} & W^0_{21} \\ \vdots & \vdots \\ W^0_{1M} & W^0_{2M} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \right)$$

Et, si l'on écrit cette expression sous la forme de sommes, on a:

$$Y = \sum_{m=1}^{M} W^1_{m1} \cdot f\left(\sum_{n=1}^{2} W^0_{nm} . X_n \right)$$

Néanmoins, les théorèmes trouvés jusqu'à présent ne sont que des théorèmes d'existence. En aucun cas, ils ne déterminent la structure du réseau de neurones. C'est à dire:
_ le nombre de sommations à effectuer pour approcher correctement une fonction donnée (= nombre de neurones de la couche cachée)
_ la valeur des coefficients (= poids)
_ l'expression de l'erreur entre la fonction réelle et la fonction à approcher.

Ils permettent juste d'assurer qu'il existe un réseau de neurones réalisant l'approximation de n'importe quelle fonction.

3) Approximation par les polynômes de BERNSTEIN

3.1) Définitions

Théorème de WEIERSTRASS:
Soit f une fonction numérique continue définie sur un intervalle fermé et borné [a,b]. Pour tout $\varepsilon>0$ il existe un polynôme P qui est une approximation à ε près de f.

Définition des polynômes de BERNSTEIN:
Soit f une fonction, continue définie sur l'intervalle [0,1]., le n_ième polynôme de BERNSTEIN de f s'écrit:

$$B_n(f)(x) = \sum_{k=0}^{n} C^n_k . f\left(\frac{k}{n}\right) . x^k . (1-x)^{n-k} \qquad (9)$$

Théorème d'approximation:
Soit f une fonction continue définie sur l'intervalle [0,1], on considère M>0 tel que:
$|f(x)| \leq M$ et pour tout $x \in [0,1]$, $\varepsilon>0$ et $\delta>0$ tels que:
$|f(u)-f(v)| \leq \varepsilon$ si $u,v \in [0,1]$ satisfont à $|u-v|<\delta$.

Alors pour tout $x \in [0,1]$, on a: $$|f(x)-B_n(f)(x)| \leq \varepsilon + \frac{M}{n.\delta^2} \qquad (10)$$

Remarque:
Pour appliquer les résultats précédents sur tout intervalle [a,b], il suffit d'effectuer la transformation $x = \frac{x-a}{b-a}$ dans la formule (9).

3.2) Démonstration

La démonstration du théorème prouvant que les polynômes de BERNSTEIN peuvent approcher n'importe quelle fonction nécessite plusieurs formules qui sont démontrées ci-après.

a)
Soit f une fonction continue sur un intervalle borné [a,b], alors si f est bornée et uniformément continue (Théorème de HEINE): il existe un M>0 tel que $|f(x)| \leq M$
Pour tout x; et pour tout $\varepsilon>0$ on peut trouver un $\delta>0$ tel que $|f(s)-f(t)| < \varepsilon$
pour s, $t \in [a,b]$ vérifiant $|s-t| < \delta$.

Soient u, $v \in [a,b]$, si $|u-v| < \delta$, on a $|f(u) - f(v)| < \varepsilon$ et a fortiori

$$|f(u) - f(v)| < \varepsilon + 2.M.\left(\frac{u-v}{\delta}\right)^2$$

L'ajout de ce terme donne une indication dans le nombre n de sommation à effectuer (voir remarque).

Si $|u-v| \geq \delta$, alors $\left(\frac{u-v}{\delta}\right)^2 \geq 1$ et

$$|f(u) - f(v)| \leq |f(u)| + |f(v)| \leq 2.M \leq 2.M.\left(\frac{u-v}{\delta}\right)^2 < \varepsilon + 2.M.\left(\frac{u-v}{\delta}\right)^2$$

Soit,

$$\boxed{|f(u) - f(v)| < \varepsilon + 2.M.\left(\frac{u-v}{\delta}\right)^2 \quad (A)}$$

b)
La formule du binôme de NEWTON s'écrit

$$(a+b)^n = \sum_{k=0}^{n} C_k^n . a^k . b^{n-k}$$

Par le changement de variable a -> x et b -> 1-x, on en déduit que:

$$\boxed{\sum_{k=0}^{n} C_k^n . x^k . (1-x)^{n-k} = (x+1-x)^n = 1 \quad (B)}$$

c)

$$\sum_{k=0}^{n} C_k^n .k. x^k . (1-x)^{n-k} = \sum_{k=1}^{n} C_k^n .k. x^k . (1-x)^{n-k} = \sum_{k=1}^{n} \frac{n(n-1)...(n-k+1)}{(k-1)...2.1} .x^k . (1-x)^{n-k}$$

$$= n.x. \sum_{k=1}^{n} \frac{(n-1)...(n-k+1)}{(k-1)...2.1} .x^{k-1} . (1-x)^{(n-1)-(k-1)}$$

$$= n.x. \sum_{k=1}^{n} C_{k-1}^{n-1} . x^{k-1} . (1-x)^{(n-1)-(k-1)}$$

$$= n.x. \sum_{k=0}^{n-1} C_{k-1}^{n-1} . x^k . (1-x)^{(n-1-k)}$$

Si l'on remplace dans cette identité (B) à l'ordre n-1, on obtient

$$\boxed{\sum_{k=0}^{n} C_k^n \, k \, . \, x^k \, . \, (1-x)^{n-k} = n.x \quad (C)}$$

d)
De la même façon

$$\sum_{k=0}^{n} C_k^n \, .k^2 . \, x^k \, . \, (1-x)^{n-k} = \sum_{k=0}^{n} C_k^n \, .k.(k-1). \; x^k \, . \, (1-x)^{n-k} + \sum_{k=0}^{n} C_k^n \, .k. \, x^k \, . \, (1-x)^{n-k}$$

$$= \sum_{k=2}^{n} \frac{n(n-1)\ldots(n-k+1)}{k(k-1)\ldots 2.1} \, .k.(k-1).x^k \, . \, (1-x)^{n-k} + n.x$$

$$= n.(n-1).x^2 + n.x$$

D'où

$$\boxed{\sum_{k=0}^{n} C_k^n \, .k^2 . \, x^k \, . \, (1-x)^{n-k} = n.(n-1).x^2 + n.x \quad (D)}$$

On rappelle les conditions du théorème (9):
f est une fonction continue définie sur l' intervalle [0,1] et on considère M>0 tel que:
$|f(x)| \leq M$ et pour tout $x \in [0,1]$, $\varepsilon>0$ et $\delta>0$ tels que:
$|f(u)-f(v)| \leq \varepsilon$ si $u,v \in [0,1]$ satisfont à $|u-v|<\delta$.

Quels que soient u,v appartenant à [0,1], on a d'après (A):

$$|f(u) - f(v)| < \varepsilon + 2.M.\left(\frac{u-v}{\delta}\right)^2$$

A partir de (B), on déduit:

$$f(x) - B_n(f)(x) = \sum_{k=0}^{n} C_k^n \, . \left[f(x) - f\left(\frac{k}{n}\right)\right] . \, x^k \, . \, (1-x)^{n-k}$$

Par l'inégalité triangulaire et en remarquant que $x^k \, . \, (1-x)^{n-k} > 0$ pour $x \in [0,1]$, on obtient:

$$|f(x) - B_n(f)(x)| \leq \sum_{k=0}^{n} C_k^n \, . \left|f(x) - f\left(\frac{k}{n}\right)\right| . \, x^k \, . \, (1-x)^{n-k}$$

De (A), on déduit $\left|f(x) - f\left(\frac{k}{n}\right)\right| < \varepsilon + \frac{2.M}{\delta^2}.\left(x-\frac{k}{n}\right)^2$, et, alors

$$|f(x) - B_n(f)(x)| < \sum_{k=0}^{n} \varepsilon \, . \, C_k^n \, . \, x^k \, . \, (1-x)^{n-k} + \sum_{k=0}^{n} C_k^n \, . \, x^k \, . \, (1-x)^{n-k} \, . \, \frac{2.M}{\delta^2}.\left(x-\frac{k}{n}\right)^2$$

Par l'identité (B)

$$|f(x) - B_n(f)(x)| < \varepsilon + \frac{2.M}{\delta^2} \, . \sum_{k=0}^{n} C_k^n \, . \left[x^2 - \frac{2.k.x}{n} + \frac{k^2}{n^2}\right] . \, x^k \, . \, (1-x)^{n-k}$$

Puis, par les identités (B), (C) et (D)

$$|f(x) - B_n(g)(x)| < \varepsilon + \frac{2.M}{\delta^2} \cdot \left[x^2 - 2.x^2 + \frac{n.(n-1)}{n^2}.x^2 + \frac{x}{n}\right] = \varepsilon + \frac{2.M}{\delta^2} \cdot \left[\frac{x}{n} - \frac{x^2}{n}\right]$$

Or, on a $x.(1-x) \leq \frac{1}{2}$ quelque soit $x \in [a,b]$, donc

$$\boxed{|f(x)-B_n(f)(x)| \leq \varepsilon + \frac{M}{n.\delta^2}} \quad (10)$$

Remarque:

Si l' on prend en particulier $n \geq \frac{M}{\varepsilon.\delta^2}$, on aura $|f(x)-B_n(f)(x)| \leq 2.\varepsilon$

Autrement dit, toute fonction continue peut être approcher pour tout x appartenant à [0,1] peut être approchée à la précision $2.\varepsilon$ par un polynôme, où ε est un nombre quelconque; ce qui revient à dire qu'une fonction continue sur [0,1] peut être approchée d'aussi près qu'on le veut par un polynôme.

4) Application aux réseaux de neurones

4.1) Détermination de l'architecture

Le réseau doit simuler:

$$\hat{f}(x) = B_n(f)(x) = \sum_{k=0}^{n} C_k^n.f(\frac{k}{n}).x^k.(1-x)^{n-k}$$

Ceci peut être réalisé (voir figure 28) en prenant:

- un neurone linéaire dans la couche de sortie qui va effectuer la somme
- n neurones dans la couche cachée; le neurone k ayant pour la fonction d'activation : $x^k.(1-x)^{n-k}$
- un neurone linéaire dans la couche d'entrée, permettant de mémoriser la valeur d'entrée

La formule (9) fournit en plus une initialisation du réseau.
Les poids entre la couche d'entrée et la couche cachée à 1 et ceux entre la couche cachée et la couche de sortie à :

$C_k^n.f(\frac{k}{n})$ pour la connexion k du réseau comportant n neurones (11)

La seule connaissance nécessaire à la construction de ce réseau de neurones est donc un vecteur des valeurs des sorties de la fonction à approcher {f(a), ..., f(b)} dont le nombre de composantes permet de déterminer :

- le nombre de neurones de la couche cachée.
- les poids des connexions.

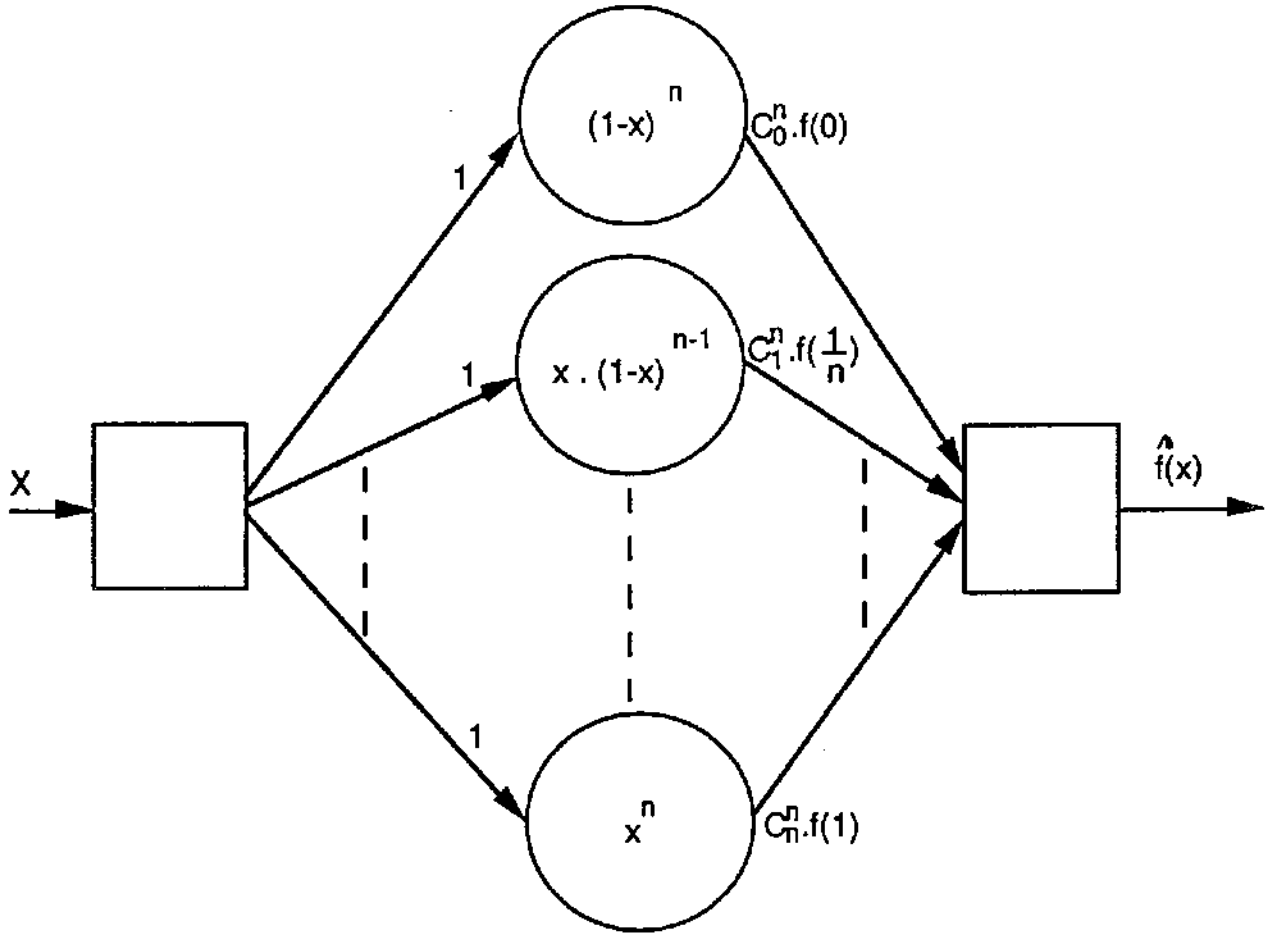

figure 28: Architecture du réseau de neurones approchant une fonction f par les polynômes de BERNSTEIN

4.2) Mise en œuvre de la simulation

Les simulations ont été effectuées sur le logiciel MATHLAB fonctionnant sur un PC 80386.
Le lancement de la simulation s'effectue en appelant la fonction k(n) où n représente le nombre de neurones du réseau qui approchera la fonction f(), préalablement entrée.
L'initialisation des poids est réalisée en identifiant un vecteur V contenant les poids des connexions entre la couche d'entrée et la couche cachée à 1, et chaque composante k du vecteur U contenant les poids des connexions entre la couche cachée et la couche de sortie à $C_k^n.f(\frac{k}{n})$
La procédure de calcul de la fonction et de son approximation s'effectue en faisant balayer l'index i dans l'intervalle [0,1] et en calculant:
_ la sortie de la fonction f(i)
_ la sortie I du réseau de neurones
_ l'erreur d'approximation
que l'on range respectivement dans les vecteurs: Z, Y, e

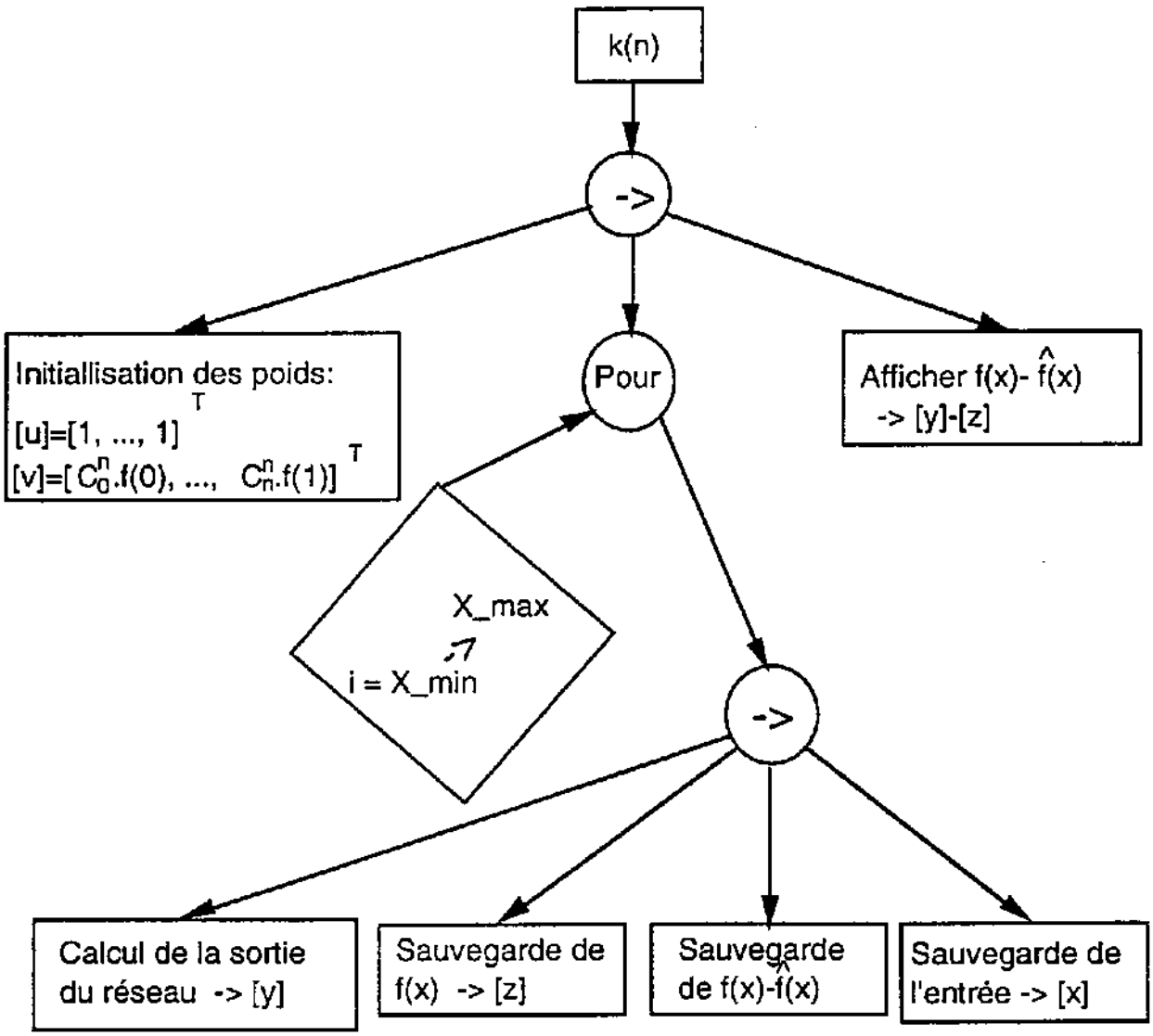

figure 29: Arbre du programme simulant le réseau de neurones

```
%*****************************************************************
function [y]=h(n,i)
%*****************************************************************

y=f(i/n);



%*****************************************************************
% calcul de n!/(k!(n-k)!)
%*****************************************************************


function [r]=cnp(n,k)

r=fact(n)/(fact(k)*fact(n-k));
```

```
%*****************************************************************
%Simulation d'un réseau de n neurones approximant par
%les polynômes de BERNSTEIN une fonction f()
%*****************************************************************
function [y]=k(n)

%intervalle d'approximation
x_min=0;
x_max=1;

%*****************************************************************
%initiallisation des poids
%*****************************************************************
%                              de la couche cachée
[u]=cnp(n,0)*h(n,0);
for k=1:n,
    [u]=[u cnp(n,k)*h(n,k)];
    end;
%                              de la couche de sortie
[v]=1;
for k=1:n,
    [v]=[v 1];
    end;

%*****************************************************************
%procédure de calcul
%*****************************************************************
%_ de la sortie du réseau de neurones
%_ de la sortie de la fonction à approximer pour des valeurs
%  de x espacées de pas=0.05 et comprise dans [0,1]

%sauvegarde de l'entrée
[x]=[x i];

pas=0.05;
for i=x_min:pas:x_max,

    %calcul du vecteur des sorties de la couche cachée
    [w]=(1-i*v(1))^n;
    for k=1:n,
        [w]=[w (i*v(k+1))^k*(1-i*v(k+1))^(n-k)];
        end;

     %calcul de la sortie du neurone de la couche de sortie
     l=0;
     for k=0:n,
         l=l+u(k+1)*w(k+1);
         end;

     % sauvegarde de la sortie du réseau
     [y]=[y l];

     %sauvegarde de la sortie de la fonction
     [z]=[z f(i)];

     %sauvegarde de l'erreur d'approximation
     [e]=[e f(i)-l];
     end;
plot(x,y,'r',x,z,'b',x,e,'g');
end;
```

5) Essais et résultats

5.1) Approximation d'une fonction linéaire

La 1ère approximation a été faite sur la fonction f(x)= x + 0,2 pour 0<x<1 sur un réseau comportant 10 neurones (n=9), initialisés selon (11) avec {f(0), f(0.1), ..., f(0.9)}.
Le résultat obtenu est excellent et est représenté figure 30.
Il est à noter que l'erreur n'est peut être pas due à l'approximation, mais à la précision de calcul de l'ordinateur même.

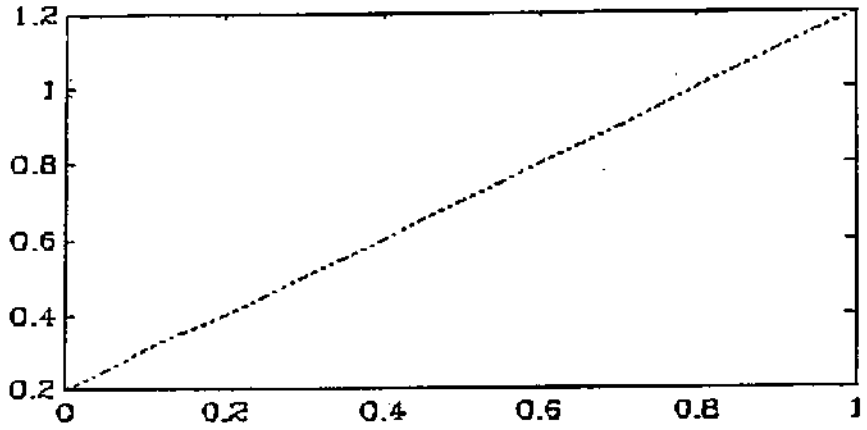

figure 30 a): f(x)=x + 0,2 et son approximation par un réseau de 10 neurones

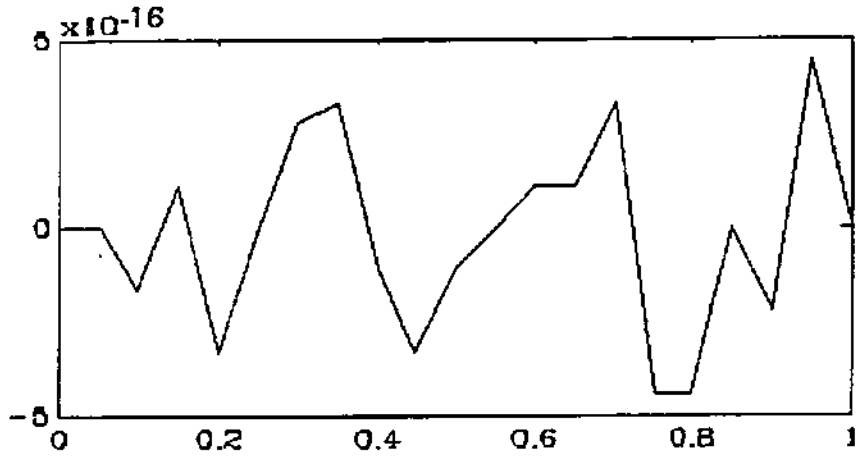

figure 30 b): erreur d'approximation

Le réseau ayant été calculé pour approcher f() dans l'intervalle [0,1], il est intéressant d'étudier la validité de cette approximation en dehors de cet intervalle. Les résultats de la figure 31 permettent de conclure que ce réseau approche la fonction f(x)= x + 0,2 avec une erreur d'approximation maximale inférieure à + ou - 10% pour -15<x<15.
Cette augmentation de l'intervalle est en accord avec la propriété de généralisation bien connue des réseaux de neurones.
Ce réseau est capable de fournir une bonne approximation de la fonction pour des valeurs de x sortant de l'intervalle prévu pour lequel il a été calculé.

D'autres approximations de fonctions élémentaires ont été simulées. La fonction exponentielle approchée par un réseau de 10 neurones et la fonction sin(pi.x) par un réseau de 12 neurones sont représentées figures 32, 33, 34 et 35.

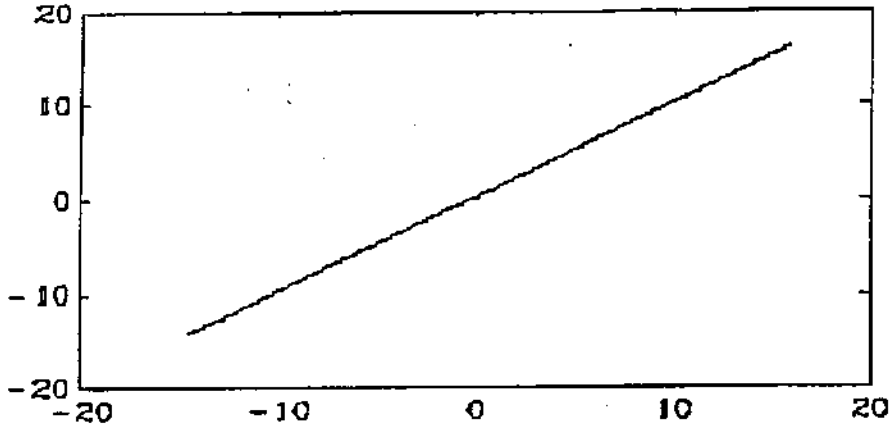

figure 31 a): f(x)=x + 0,2 et son approximation par un réseau de 10 neurones

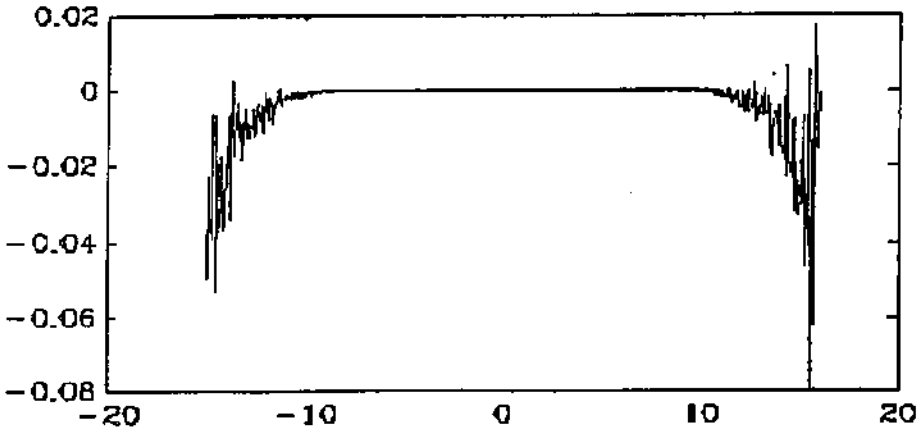

figure 31 b): erreur d'approximation

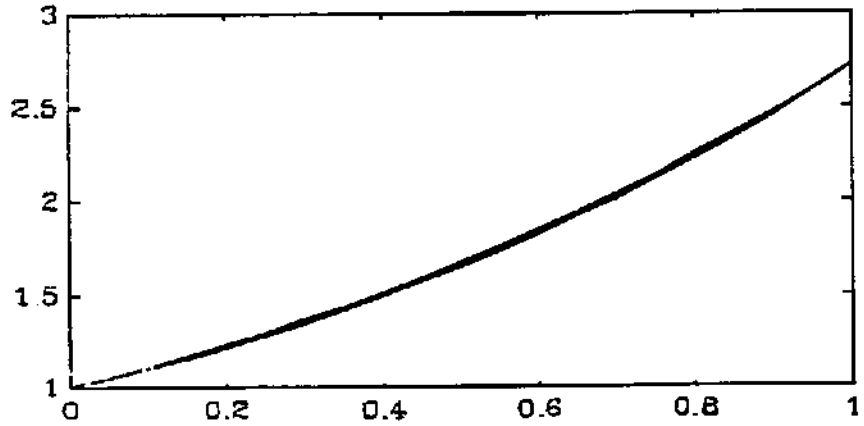

figure 32 a): f(x)=exp(x) et son approximation par un réseau de 10 neurones

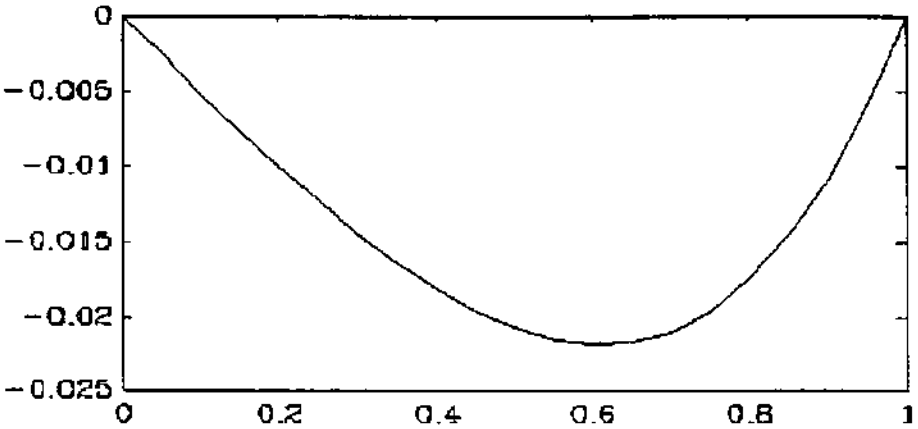

figure 32 b): erreur d'approximation

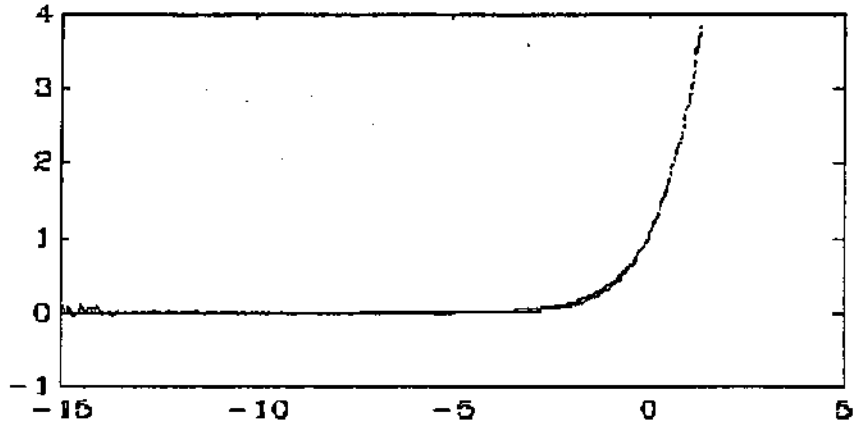

figure 33 a): f(x)=exp(x) et son approximation par un réseau de 10 neurones

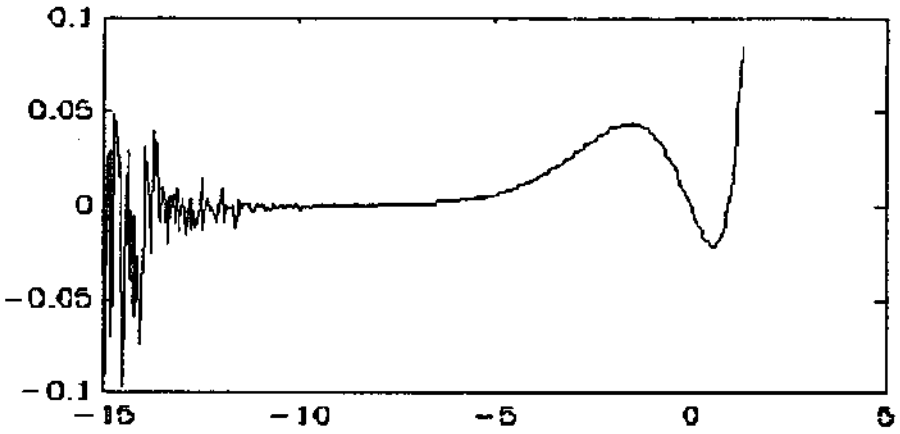

figure 33 b): erreur d'approximation

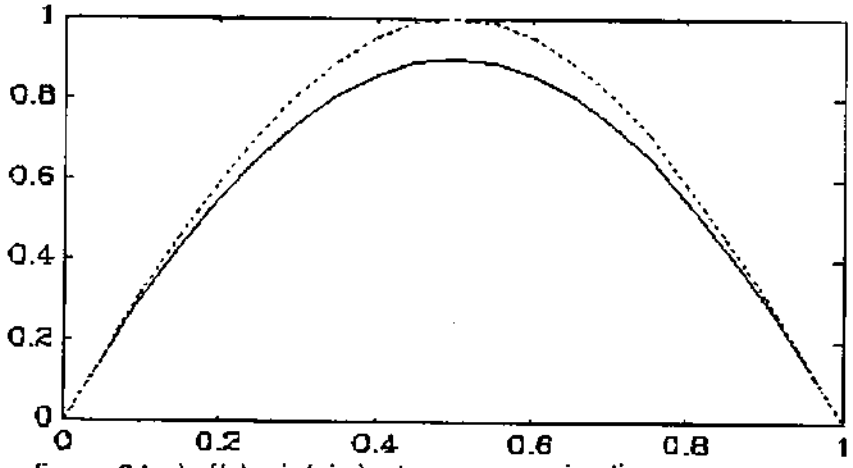

figure 34 a): f(x)=sin(pi.x) et son approximation par un reseau de12 neurones

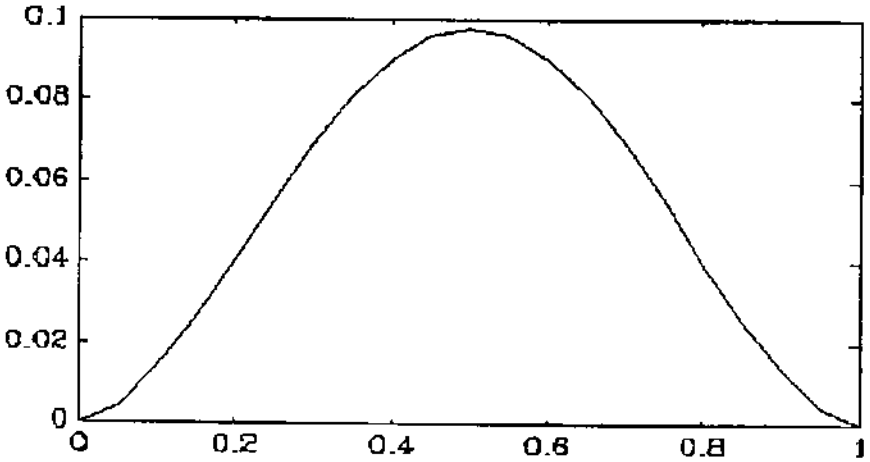

figure 34 b): erreur d'approximation

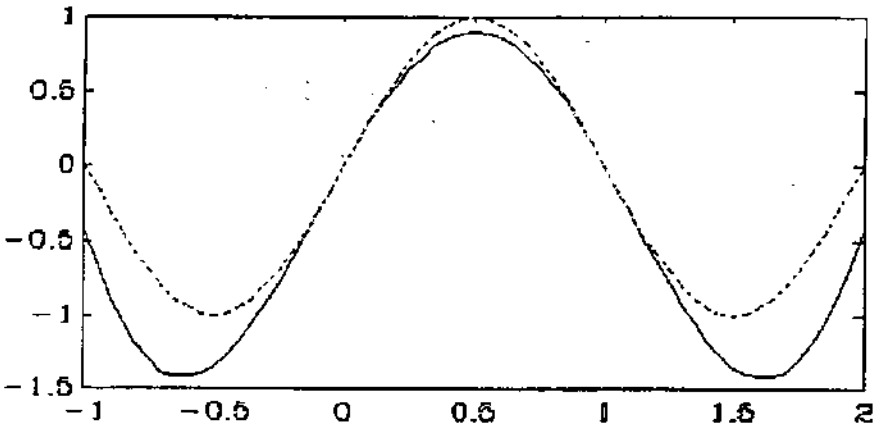

figure 35 a): f(x)=sin(pi.x) et son approximation par un réseau de 12 neurones

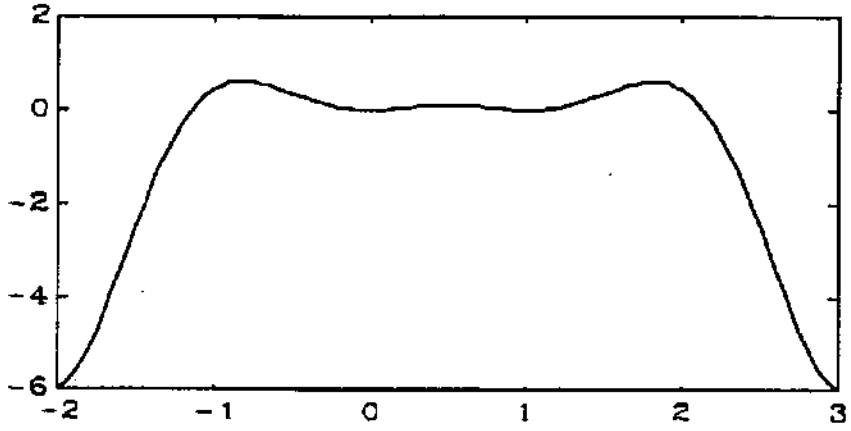

figure 35 b): erreur d'approximation

5.2) Etude du comportement de l'erreur

5.2.1) Minimisation de l'erreur par augmentation du nombre de neurones

Le réseau déduit en 3.2.1) va fournir une approximation de f:

$$f(x) = \hat{f}(x) + \varepsilon$$

On peut envisager diverses solutions pour réduire l'erreur d'approximation.

La formule (10): $|f(x)-B_n(f)(x)| \leq \varepsilon + \frac{M}{n.\delta^2}$ implique que l'erreur d'approximation diminue si le nombre de neurones n augmente.

Ceci peut être réalisé à condition de connaître autant de valeurs de f() afin d'initialiser les poids des connexions par: $C_k^n.f(\frac{k}{n})$
(Poids du Kième neurone de la couche cachée comportant au total n neurones)

Partant d'un réseau comportant n1 neurones qui conduit à une erreur d'approximation: ε_1 trop importante pour être acceptable, on va augmenter progressivement la taille de ce réseau jusqu'à obtenir une erreur: ε_2 acceptable qui correspond, alors, à un nombre de neurones n2 (>n1).
L'approximation de sin (2.pi.x) par un réseau de 10 neurones (n1=9), initialisé par les valeurs de f(): {f(0), f(0.1), ..., f(0.9)} conduit à une erreur d'approximation maximale de + ou - 30% (voir figure 36).

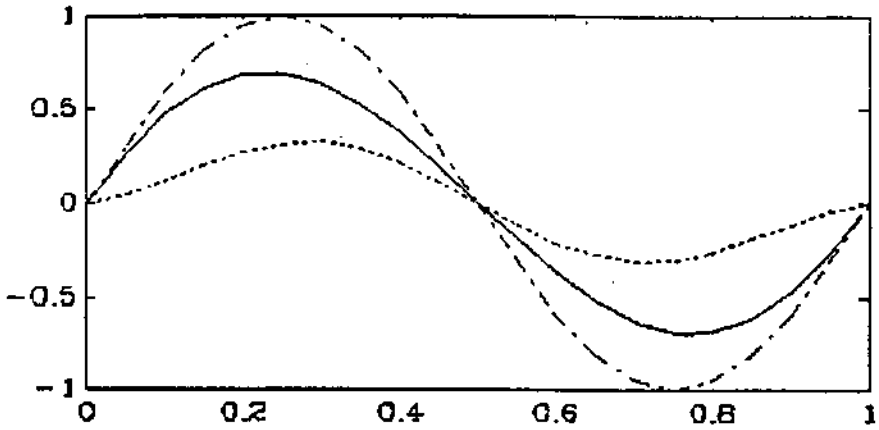

figure 36 a): f(x)= sin(2.pi.x) et son approximation par un réseau de 10 neurones

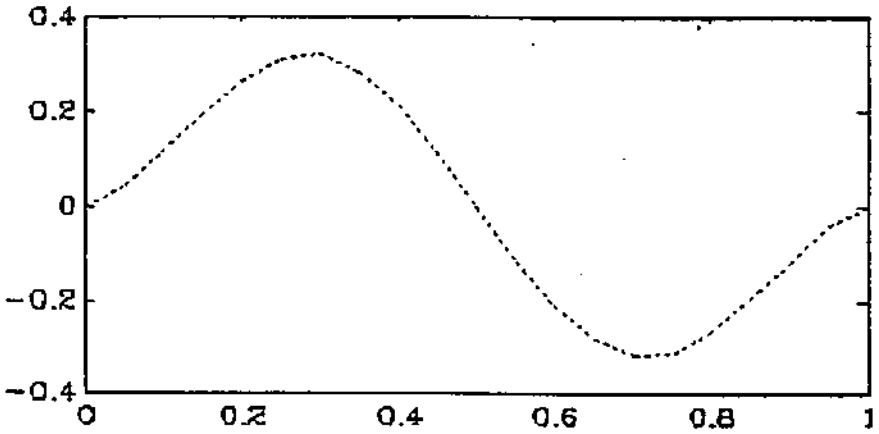

figure 36 b): erreur d'approximation

On est parvenu à obtenir une erreur d'approximation de + ou - 10% en portant ce réseau à 38 neurones (n2=37), initialisé par {f(0), f(1/38), ..., f(37/38)} (Voir figure 37).

Plusieurs essais n'ont pas aboutis à l'obtention d' une relation linéaire du type: $\varepsilon = k \cdot n$

Ceci peut conduire à l'hypothèse suivante:
L'erreur dépend de la fonction à approcher; c'est à dire de ses variations (minima, maxima) dans l'intervalle où elle est approchée.

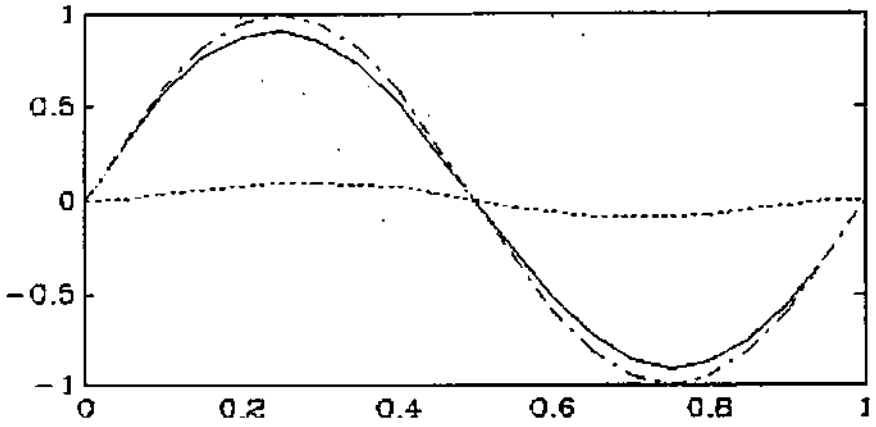

figure 37 a): f(x)= sin(2.pi.x) et son approximation par un réseau de 38 neurones

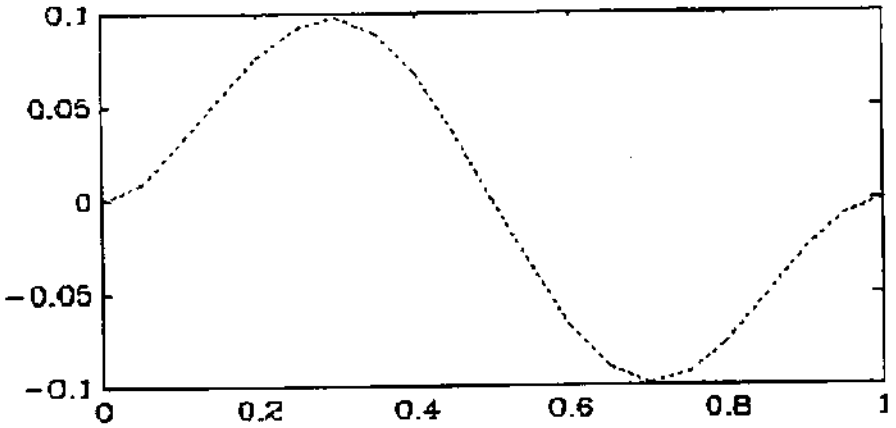

figure 37 b): erreur d'approximation

5.2.2) Minimisation de l'erreur par apprentissage

Précédemment, on a minimiser l'erreur d'approximation en augmentant la taille du réseau. On peut supposer que les poids obtenus par initialisation ne sont pas optimaux et, partant de leurs valeurs, on peut les ajuster par apprentissage (voir partie 2: 3)) suivant l'algorithme du gradient selon le schéma de la figure 38.

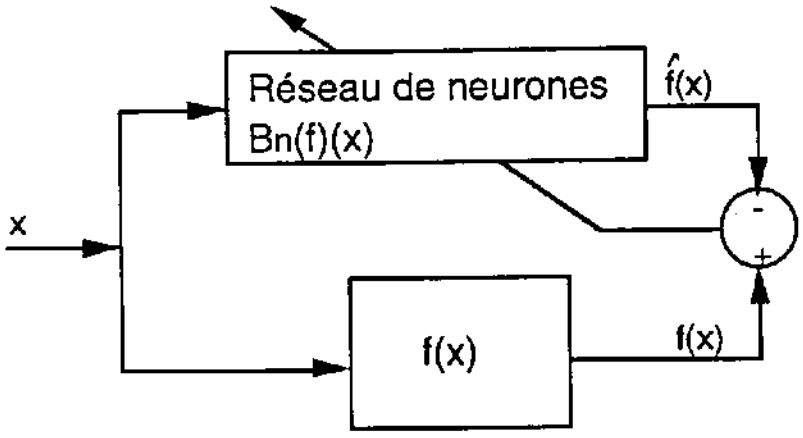

figure 38: Minimisation de l'erreur d'approximation par apprentissage du réseau de neurones

L'expérience a été réalisée sur le même réseau de 10 neurones (n=9) approchant sin(2.pi.x). L'apprentissage consiste à présenter un échantillon de 20 valeurs prises dans l'intervalle d'approximation de la fonction [0,1]: {0, 0.05, 0.1, ..., 0.9}, et, à ajuster les poids selon l'algorithme du gradient (voir partie 2: 3.3)) pour chaque valeur et de façon à minimiser l'erreur d'approximation.

Les résultats (figure 39) sont obtenus au bout de 7 présentations de cet échantillon (c'est à dire 7.20=140 ajustements).

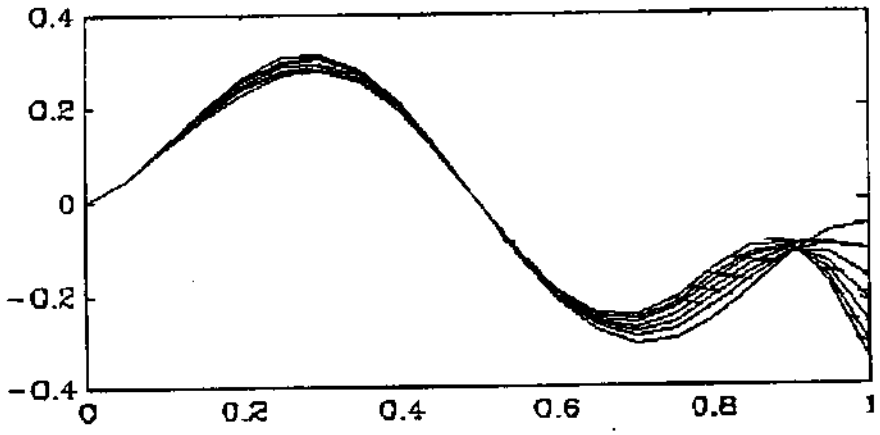

figure 39 : Erreur d'approximation après apprentissage du réseau de 10 neurones avec 7 échantillons de 20 valeurs prises dans l'intervalle [0,1]

On observe bien une diminution de l'erreur sur la majorité de l'intervalle [0,1], par contre elle augmente au voisinage de 1. On peut donc affirmer que l'apprentissage, s'il est utilisé avec un test d'arrêt permet de diminuer l'erreur d'approximation pour ce type de réseau.
On peut noter que certaines valeurs de l'échantillon utilisées lors de l'apprentissage, correspondaient à une erreur nulle. Un autre test a été effectué en prenant toujours 20 valeurs mais dans l'intervalle [0.1,1]: {0.1, 0.145, ...}; ce qui permet d'exclure une partie des valeurs pour lesquelles l'erreur est nulle.
Les résultats obtenus au bout de 7 présentations de ce nouvel échantillon (figure 40), comparés à ceux obtenus précédemment (figure 39), montrent une très nette amélioration du comportement de l'erreur.

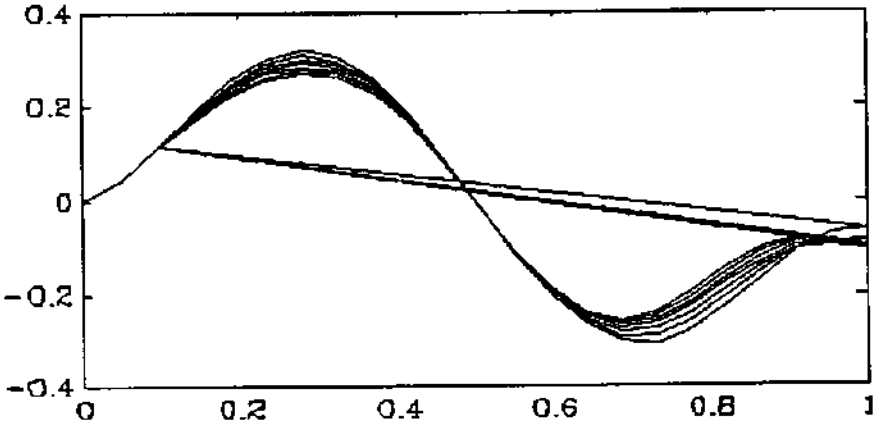

figure 40 : Erreur d'approximation après apprentissage du réseau de 10 neurones avec 7 échantillons de 20 valeurs prises dans l'intervalle [0.1,1]

Finalement, le résultat optimal, est obtenu en présentant 38 échantillons et est représenté figure 41.
Ces résultats montrent que l'apprentissage spécialisé apparaît indispensable pour ce type de minimisation.
La valeur de chaque poids avant et après apprentissage a été relevée figure 42.

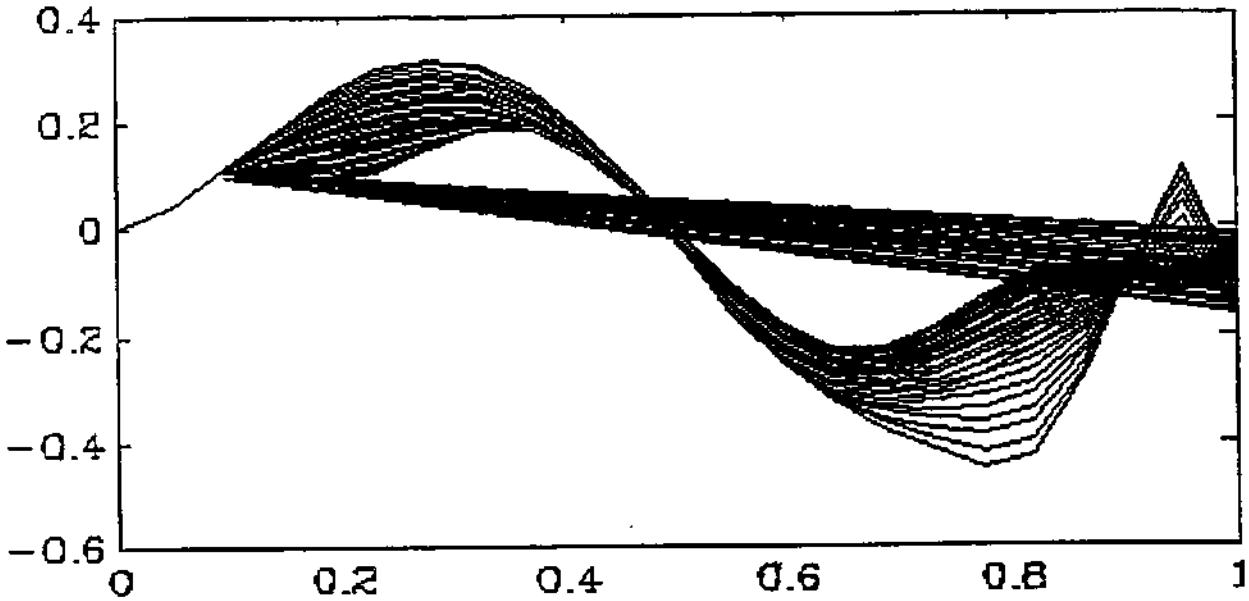

figure 41: Erreur d'approximation après apprentissage du réseau de 10 neurones avec 38 échantillons de 20 valeurs

Valeurs des poids:

a) initialisés		b) après apprentissage	
1.000000e+000	0.000000e+000	9.829490e-001	5.339479e-002
1.000000e+000	5.877853e+000	6.118337e-001	5.892313e+000
1.000000e+000	4.279754e+001	8.054844e-001	4.280076e+001
1.000000e+000	1.141268e+002	1.341749e+000	1.141279e+002
1.000000e+000	1.234349e+002	1.497010e+000	1.234348e+002
1.000000e+000	3.086008e-014	9.999999e-001	-8.015873e-005
1.000000e+000	-1.234349e+002	5.161066e-001	-1.234356e+002
1.000000e+000	-1.141268e+002	7.140138e-001	-1.141281e+002
1.000000e+000	-4.279754e+001	1.232109e+000	-4.280350e+001
1.000000e+000	-5.877853e+000	1.159510e+000	-5.884939e+000

figure 42 : Evolution des poids

On peut noter que les poids entre la couche d'entrée et la couche de sortie (initialisés à 1) ont été les plus affectés (mise à part le 1er et le 6ème) par cet apprentissage. Quand aux connections entre la couche cachée et la couche de sortie, ce sont surtout les premiers et les derniers (c'est à dire ceux situés aux extrémités du réseau) qui ont le plus variés.
Le programme de la procédure d'apprentissage est donné ci dessous, ainsi que l'arbre programmatique respectif à l'ajustement des poids figure 43.

```
%*************************************************************
% Procédure d'apprentissage
%*************************************************************

function [y]=appr(x_min,x_max,o,n,d)

%pas de convergence
b=0.01;

for i=x_min:1/20:x_max,

%ajustement des poids des connexions de la couche de sortie
[c]=u(1)+b*(f(i)-1)*w(1);%ajustement du 1er poids
for k=1:n,
    a=b*(f(i)-1)*w(k+1);
    [c]=[c u(k+1)+a];
    end;

%recopie du vecteur des poids ajustés dans u
[u]=c(1);
for k=1:n,
    [u]=[u c(k+1)];
    end;

%ajustement des poids des connexions de la couche d'entrée
[c]=v(1)+b*(f(i)-1)*u(1)*n*(1-i)^(n-1);%ajustement du 1er poids
for k=1:n,
   a=((k*i^(k-1))*((1-i)^(n-k))-
                  (i^k)*(n-k)*(1-i)^(n-k-1))*(f(i)-1)*u(k+1)*b;
   [c]=[c v(k+1)+a];
   end;

%recopie du vecteur des poids ajustés dans u
[v]=c(1);
for k=1:n,
    [v]=[v c(k+1)];
    end;
```

En comparant ces deux méthodes de minimisation, on peut affirmer que la méthode qui consiste à augmenter le nombre de neurones est plus rapide et moins problématique que celle par apprentissage qui nécessite un test d'arrêt sur l'erreur.

L'ajustement des poids s'effectuant sur chaque valeur de l'échantillon, on peut qualifier cette minimisation de locale. Pour obtenir une minimisation globale de l'erreur, il suffirait d'additionner la valeur absolue de l'erreur commise pour chaque valeur de l'échantillon, et d'ajuster les poids après passage de l'échantillon complet afin de minimiser cette somme. On aurait, alors une minimisation globale de l'erreur sur l'échantillon (c'est à dire un ensemble de valeurs de l'intervalle d'approximation) plutôt que sur une valeur de la fonction (voir partie 2: 3.4)).

Un autre moyen de diminuer l'erreur d'approximation serait d'estimer l'erreur par un second réseau dont les poids seraient ajustés pour minimiser l'erreur de l'ensemble:
(sortie du réseau approchant f() - sortie du réseau approchant l'erreur) par rapport à la fonction f selon le schéma de la figure 44.

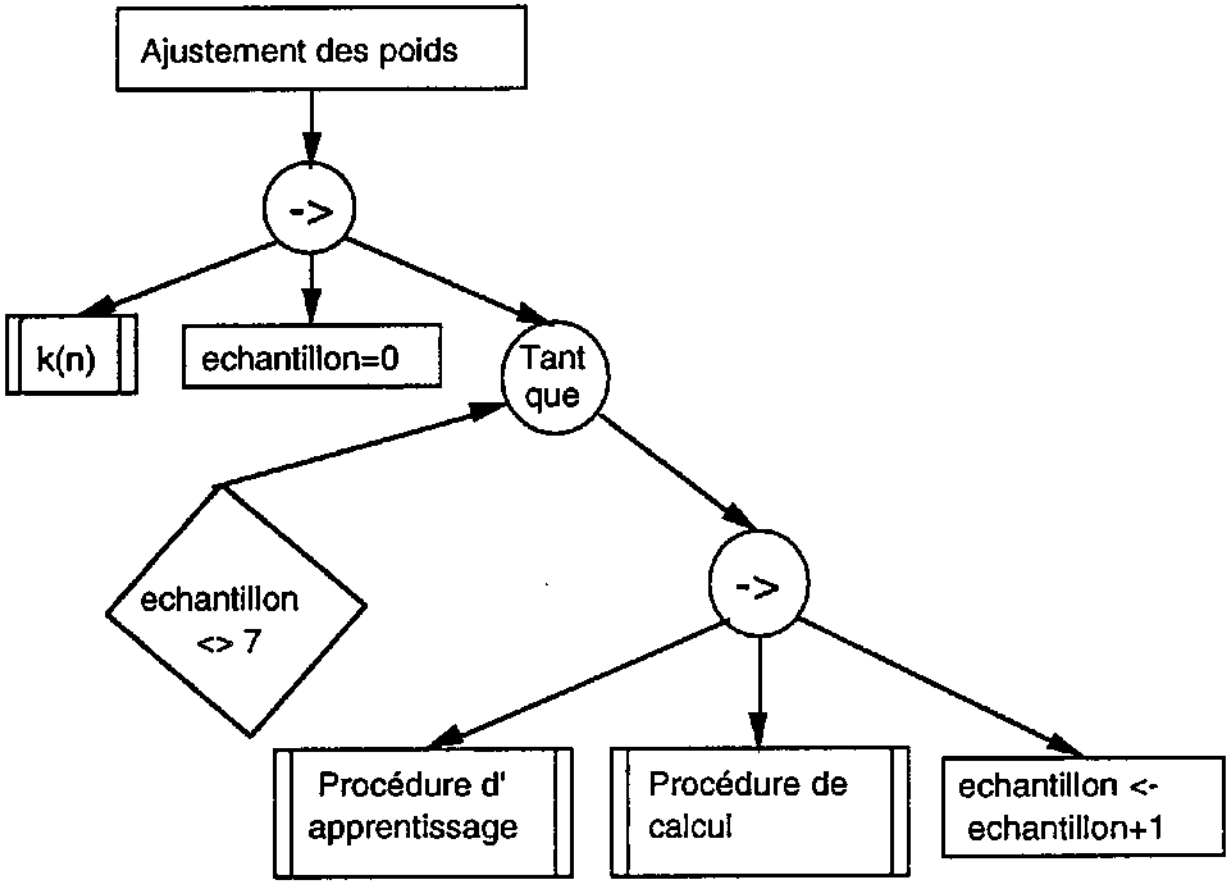

figure 43: Arbre programmatique de réalisant l'ajustement des poids

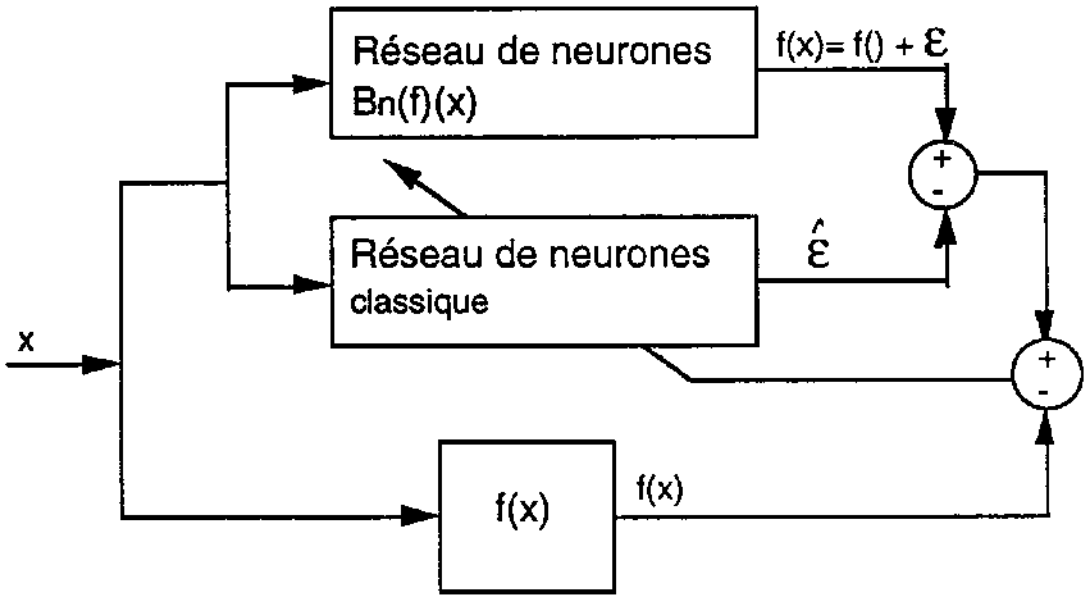

figure 44: Minimisation de l'erreur d'approximation par apprentissage d'un second réseau de neurones

5.3) Influence des poids dans l'erreur d'approximation

Pour pouvoir évaluer l'importance des poids dans l'erreur d'approximation, on a coupé tour à tour chaque connexion et l'on a relevé la nouvelle approximation.
Cette expérience a été effectuée sur le réseau de 39 neurones approchant sin(2.pi.x). Quelques résultats sont représentés figures 45, 46 et 47, et sont à comparer avec la figure 37.
Cette expérience permet d'affirmer que les poids de notre réseau ont une influence locale dans l'approximation d'une fonction.
Par exemple, le 1er poids provoque une diminution de l'erreur pour 0.05<x<0.15, par contre le 19ème, 20ème et 21ème ne provoque aucune modification de l'erreur si ils sont supprimés.

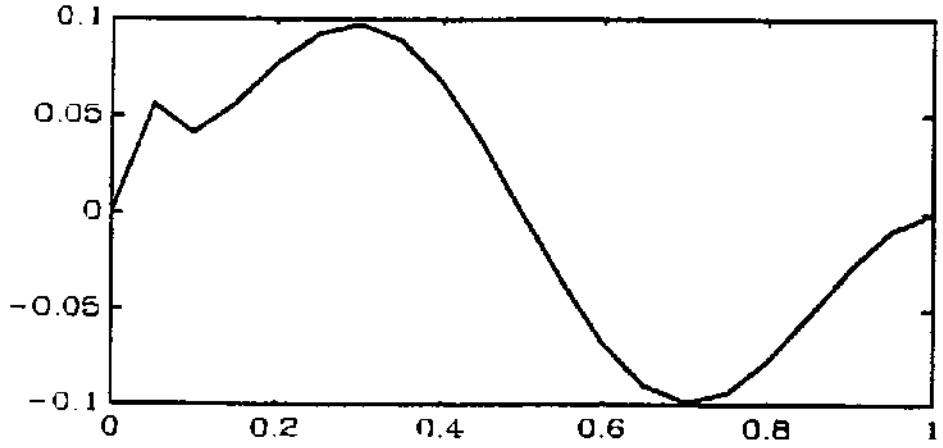

figure 45: Coupure de la 1ère connexion du réseau de 39 neurones

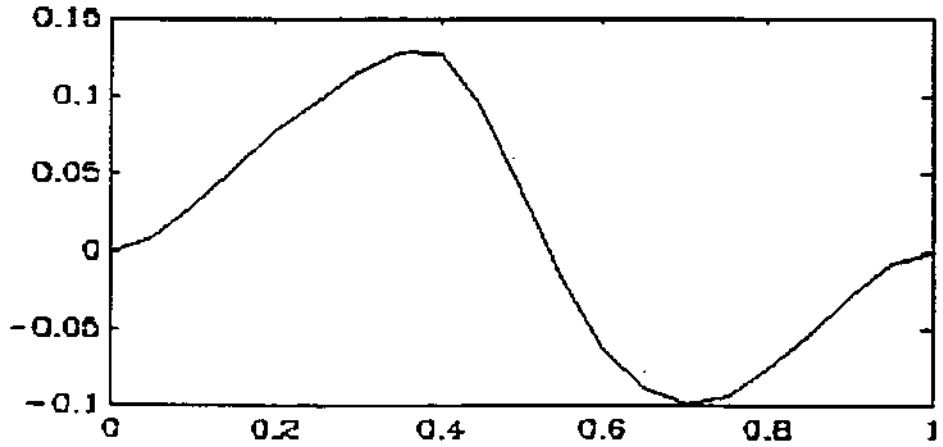

figure 46: Coupure de la 17ème connexion du réseau de 39 neurones

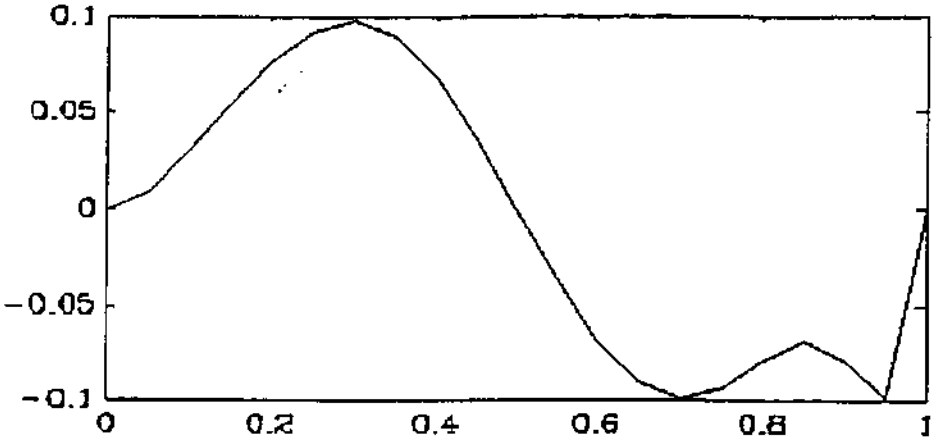

figure 47: Coupure de la 27ème connexion du réseau de 39 neurones

5.4) Généralisation de l'approximation

On a vu que l'on pouvait utiliser la formule (9) sur n'importe quel intervalle [a,b], en effectuant la transformation: $x \rightarrow \frac{x-a}{b-a}$

On se propose, dans cet essai, de vérifier cette propriété en entraînant un réseau de 39 neurones à approcher sin(2.pi.x) sur l'intervalle [-1,0]. L'erreur, représentée figure 48, comparable à celle obtenue précédemment (figure 36), confirme bien cette possibilité de translation.

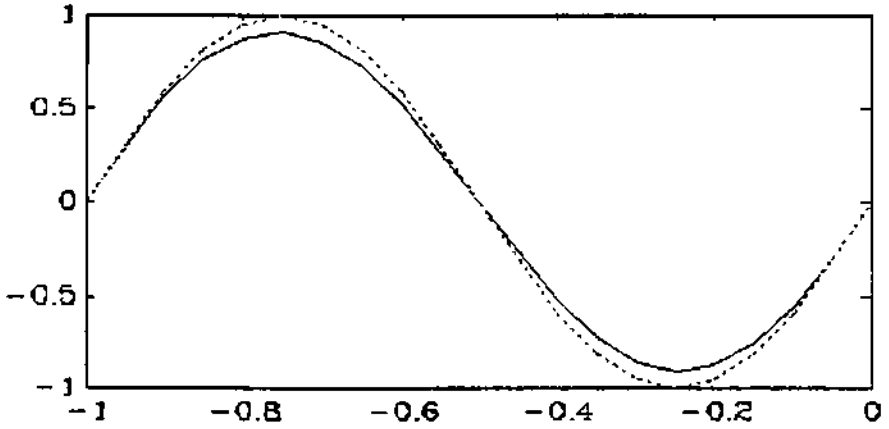

figure 48 a) : f(x) = sin(2.pi.x) et son approximation sur [-1,0]

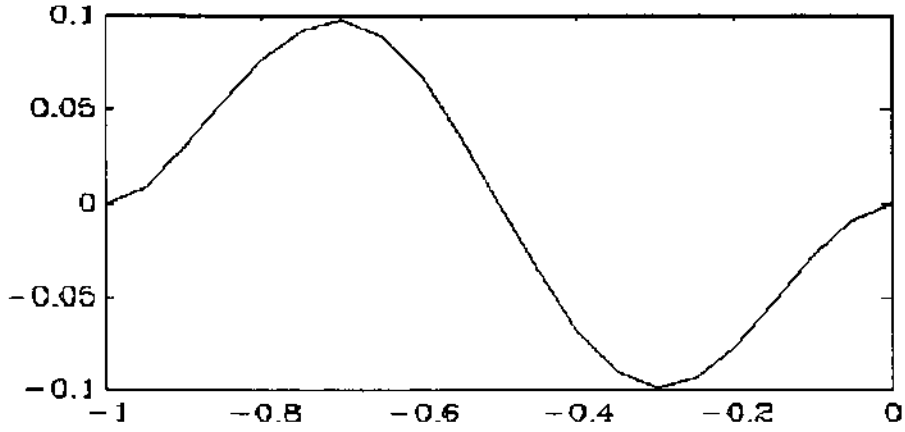

figure 48 b): erreur d'approximation

6) Extension aux systèmes à une entrée, plusieurs sorties (SIMO)

La formule (9): $B_n(f)(x) = \sum_{k=0}^{n} C_k^n . f(\frac{k}{n}) . x^k . (1-x)^{n-k}$ est limitée à l'approximation d'une fonction mono-variable (x).
En pratique, on a à faire à de très gros systèmes régis par des équations multi-variables. On va étendre (9) à des systèmes une entrée, plusieurs sorties.

Ces systèmes sont régis par des équations du type:

$Y1 = f1(x)$
.
.
.
$Yk = fk(x)$

Il suffit, alors d'approcher chaque fonction (f1(x), ..., fk(x)) par un polynôme de BERNSTEIN et de construire l'approximation du système en mettant en parallèle les architectures déduites en 3.2.1) (figure 49)

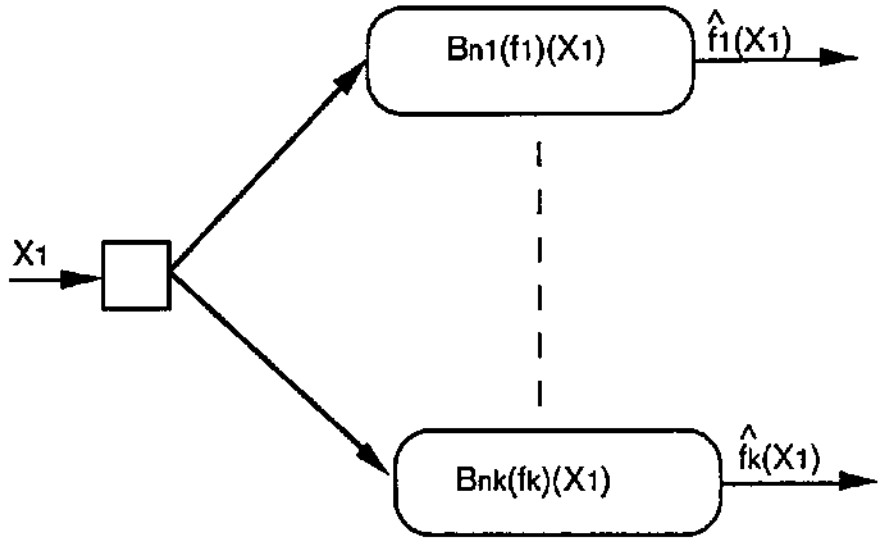

figure 49: Architecture d'un réseau simulant un système SIMO

7) Détermination pratique du réseau

L'architecture du réseau de neurones approchant une fonction f() par les polynômes de BERNSTEIN a été donnée figure 25
La seule connaissance nécessaire à la construction de ce réseau est un vecteur de certaines des valeurs [f(a), ..., f(b)] ainsi que les bornes: a,b entre lesquelles son entrée a varié.
Si l'on a à faire à une identification, [f(a), ..., f(b)] représente la sortie du processus à identifier, son entrée variant dans [a,b].
En commande, [f(a), ..., f(b)] représente les valeurs connues qui, appliquées au processus, conduisent ses sorties aux valeurs souhaitées.

Pour déterminer complètement le réseau, il suffit de suivre la procédure suivante:
1) Détermination des valeurs de f(), rangées dans un vecteur, pour une entrée croissante dans l'intervalle [a,b]
2) Le nombre de neurones de la couche cachée est alors égal au nombre de valeur connue de f()
3) La fonction d'activation de chaque neurone est donnée par $x^k.(1-x)^{n-k}$ dans laquelle on doit effectuer la transformation $x \rightarrow \frac{x-a}{b-a}$
4) L'initialisation des poids est réalisée en identifiant les poids des connexions d'entrée à 1 et les poids des connexions de sortie à $C_k^n.f_{\frac{k}{n}}()$ pour la connexion k du réseau comportant n neurones et la $\frac{k}{n}$ valeur de f() rangée dans le vecteur lors de 1).

8) Autres approximations implantables sur réseaux de neurones

8.1) Formule de TAYLOR

Soit une fonction continue définie et n fois dérivable dans l'intervalle]a,b] qui contient Xo.

Pour tout x appartenant à [a,b], on a:

$$\hat{f}(x+x_0) = f(x_0) + x \,.\, f'(x_0) + \frac{x^2}{2!} \,.\, f''(x_0) + \ldots + \frac{x^{n-1}}{(n-1)!} \,.\, f^{n-1}(x_0) + \frac{x^n}{n!} \,.\, f^n(x_0+\delta.x)$$

avec δ, un nombre réel compris entre 0 et 1

Pour Xo = 0, on a:

$$\hat{f}(x) = f(0) + x \,.\, f'(0) + \frac{x^2}{2!} \,.\, f''(0) + \ldots + \frac{x^{n-1}}{(n-1)!} \,.\, f^{n-1}(0) + \frac{x^n}{n!} \,.\, f^n(\delta.x)$$

Le réseau de neurones simulant cette formule comporte:
_ 1 neurone linéaire dans la couche de sortie éffectuant la somme
_ n neurones ayant pour fonction d'activation: x^n
_ ses n poids des connexions couche d'entrée-couche cachée égaux à 1
_ ses n poids des connexions couche cachée-couche de sortie égaux à : $\frac{f^n(0)}{n!}$

Le réseau de neurones correspondant s'en déduit facilement (figure 51), le biais étant égal à f(0).

Pratiquement, ce réseau est beaucoup plus contraignant à implanter car il nécessite la connaissance des n dérivées successives de f() qui sont:
_ difficilement mesurable sur un processus
_ difficilement calculable sur ordinateur à cause des phénomènes de bruits

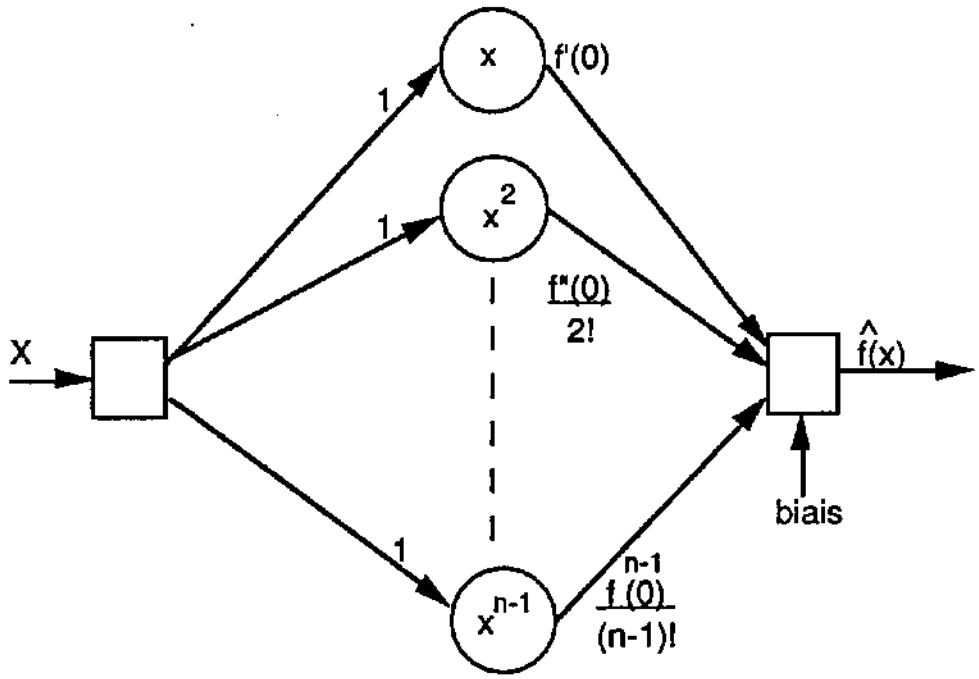

figure 51: Architecture du réseau de neurones approchant une fonction f par la formule de TAYLOR

8.2) Développement limité en série de fonctions sigmoïdes

8.2.1) Principe

Pour pouvoir déduire un réseau dont les neurones possèdent la même fonction d'activation, l'idéal serait de trouver un développement limité du type:

$$\widehat{f}(x) = \alpha_1^2.g(\alpha_1^1.x) + \alpha_2^3.g(\alpha_2^2.g(\alpha_2^1.x)) + \ldots$$

Si l'on utilise les résultats de [2] et [3], cités dans la partie 4: 2), on peut se limiter à un développement de la forme:

$$\widehat{f}(x) = \alpha_1^2.g(\alpha_1^1.x) + \alpha_2^2.g(\alpha_2^1.x) + \alpha_3^2.g(\alpha_3^1.x) + \ldots + \alpha_k^2.g(\alpha_k^1.x)$$

$$\widehat{f}(x) = \Sigma_{l=1}^{k} \alpha_l^2 .g(\alpha_l^1.x)$$ avec pour fonction g(), une fonction sigmoïde

Si on peut déterminer l'ordre du développement (k) et les valeurs des coefficients alors l'architecture du réseau de neurones correspondant est immédiate (voir figure 52).

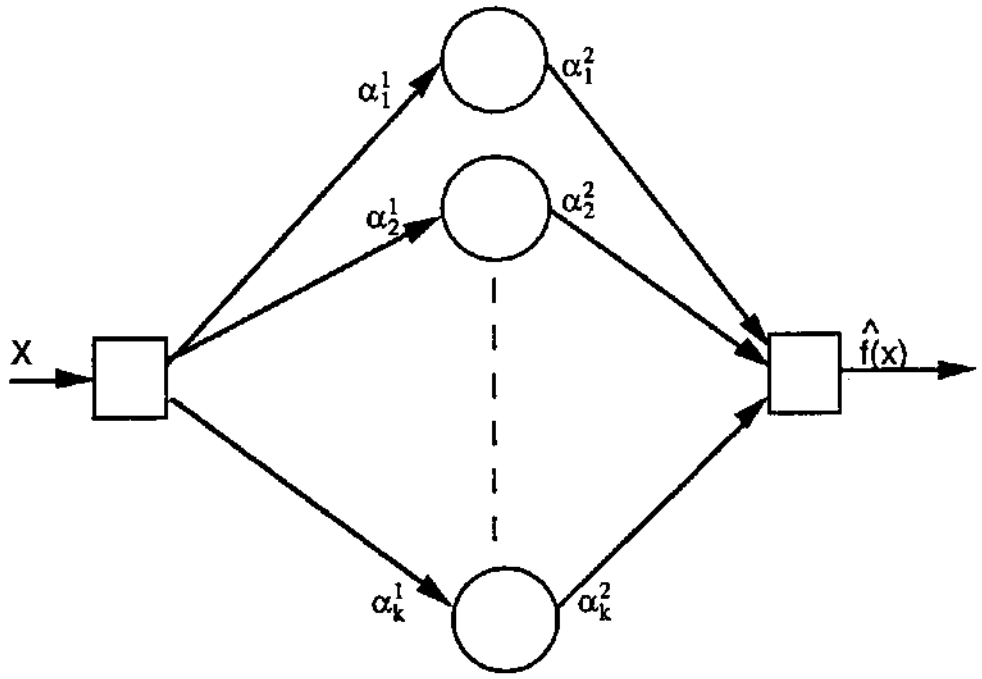

figure 52: Architecture du réseau de k neurones approchant une fonction f par un développement en série de fonctions sigmoïdes

8.2.2) Résolution par limitation du développement

La formule de TAYLOR donne une approximation de

$$th(\beta_i x) = \beta_i x - \frac{2}{3!}.\beta_i^3.x^3 + \frac{16}{5!}.\beta_i^5.x^5 - \frac{272}{7!}.\beta_i^7.x^7 + ... + \frac{m_p}{p!}.\beta_i^p.x^p$$

De même, toute fonction f() peut être approchée par son développement limité à l'ordre n:

$$\hat{f}(x+h) = f(h) + f'(h) \cdot x + \frac{f''(h)}{2!} \cdot x^2 + ... + \frac{f^{n-1}(h)}{(n-1)!} \cdot x^{n-1} + \varepsilon(h)$$

La démarche consiste à trouver une relation entre les coefficients respectifs de
$x, x^2, x^3, x^4, x^5, x^6, x^7, ...$
de façon à égaler le développement de th(x) et celui de f().

Etant donné que l'approximation de th() par la formule de TAYLOR ne fournit que des puissance de x impaires, on est obligé de recourir à l'utilisation de l'approximation de th'(x):

$$th'(\beta_i x) = \beta_i - \frac{2.3}{3!}.\beta_i^2.x^2 + \frac{16.5}{5!}.\beta_i^4.x^4 - \frac{272.7}{7!}.\beta_i^6.x^6 + ... + \frac{m_p.p}{p!}.\beta_i^{p-1}.x^{p-1}$$

Si l'on égalise les sommes des deux développements de th(x) et th'(x) au développement limité à l'ordre n de f(x) en x=0, on obtient:

$$f(0) + f'(0) \cdot x + \frac{f''(0)}{2!} \cdot x^2 + \ldots + \frac{f^{n-1}(0)}{(n-1)!} \cdot x^{n-1} + \varepsilon(0)$$

$$= \sum_{i=0}^{k} \alpha_i.th(\beta_i.x) + \sum_{i=k}^{l} \alpha_i.th'(\beta_i.x)$$

$$= \sum_{i=0}^{k} \alpha_i \left[\beta_i.x - \frac{2}{3!}.\beta_i^3.x^3 + \ldots \right] + \sum_{i=k}^{l} \alpha_i \left[1 - \beta_i^2.x^2 + \frac{16}{4!}.\beta_i^4.x^4 + \ldots \right]$$

Si on limite le développement de f() à l'ordre p (supposé impair), cette équation est vérifiée si les coefficients sont égaux:

$$\sum_{i=1}^{k} \alpha_i.\beta_i = f'(0) \qquad \sum_{i=k+1}^{l} \frac{2.3}{3!}.\alpha_i.\beta_i^2 = \frac{f''(0)}{2!}$$

$$\sum_{i=1}^{k} -\frac{2}{3!}.\alpha_i.\beta_i^3 = \frac{f^{(3)}(0)}{3!} \qquad \sum_{i=k+1}^{l} \frac{16.5}{5!}.\alpha_i.\beta_i^4 = \frac{f^{(4)}(0)}{4!}$$

$$\sum_{i=1}^{k} \frac{16}{5!}.\alpha_i.\beta_i^5 = \frac{f^{(5)}(0)}{5!} \qquad \sum_{i=k+1}^{l} -\frac{272.7}{7!}.\alpha_i.\beta_i^6 = \frac{f^{(6)}(0)}{6!}$$

$$\vdots \qquad\qquad \vdots$$

$$\sum_{i=1}^{k} \frac{m_p}{p!}.\alpha_i.\beta_i^p = \frac{f^{(p)}(0)}{p!} \qquad \sum_{i=1}^{k} \frac{m_p.p}{p!}.\alpha_i.\beta_i^{p-1} = \frac{f^{(p-1)}(0)}{(p-1)!}$$

On a alors le système d'équations à résoudre suivant:

_ pour les puissances impaires de x

$$\alpha_1.\beta_1 + \alpha_2.\beta_2 + \ldots + \alpha_k.\beta_k = f'(0)$$

$$\alpha_1.\beta_1^3 + \alpha_2.\beta_2^3 + \ldots + \alpha_k.\beta_k^3 = \frac{f^3(0)}{-2}$$

$$\vdots \qquad\qquad \vdots$$

$$\alpha_1.\beta_1^p + \alpha_2.\beta_2^p + \ldots + \alpha_k.\beta_k^p = \frac{f^{(p)}(0)}{m_p}$$

_ pour les puissances paires de x

$$\alpha_{k+1}.\beta_{k+1}^2 + \alpha_{k+2}.\beta_{k+2}^2 + \ldots + \alpha_l.\beta_l^2 = \frac{f''(0)}{2}$$

$$\alpha_{k+1}.\beta_{k+1}^4 + \alpha_{k+2}.\beta_{k+2}^4 + \ldots + \alpha_l.\beta_l^4 = \frac{f^{(4)}(0)}{16}$$

$$\vdots \qquad\qquad \vdots$$

$$\alpha_{k+1}.\beta_{k+1}^{p-1} + \alpha_{k+2}.\beta_{k+2}^{p-1} + \ldots + \alpha_l.\beta_l^{p-1} = \frac{p}{m_p}.\frac{f^{(p-1)}(0)}{(p-1)!}$$

On a donc au total 2.I coefficients à trouver, le nombre d'équation (p) doit donc être supérieur ou égal à 2.I .
Pour approcher le développement limité à l'ordre p d'une fonction f() , il faudra $\frac{p}{2}$ neurones ayant comme fonction d'activation th(x) et $\frac{p}{2}$ neurones ayant comme fonction d'activation th'(x).
De plus, si les équations sont résolubles, une initialisation du réseau est possible. On obtient , alors, les valeurs des poids α et β à remplacer dans le réseau de la figure 53.

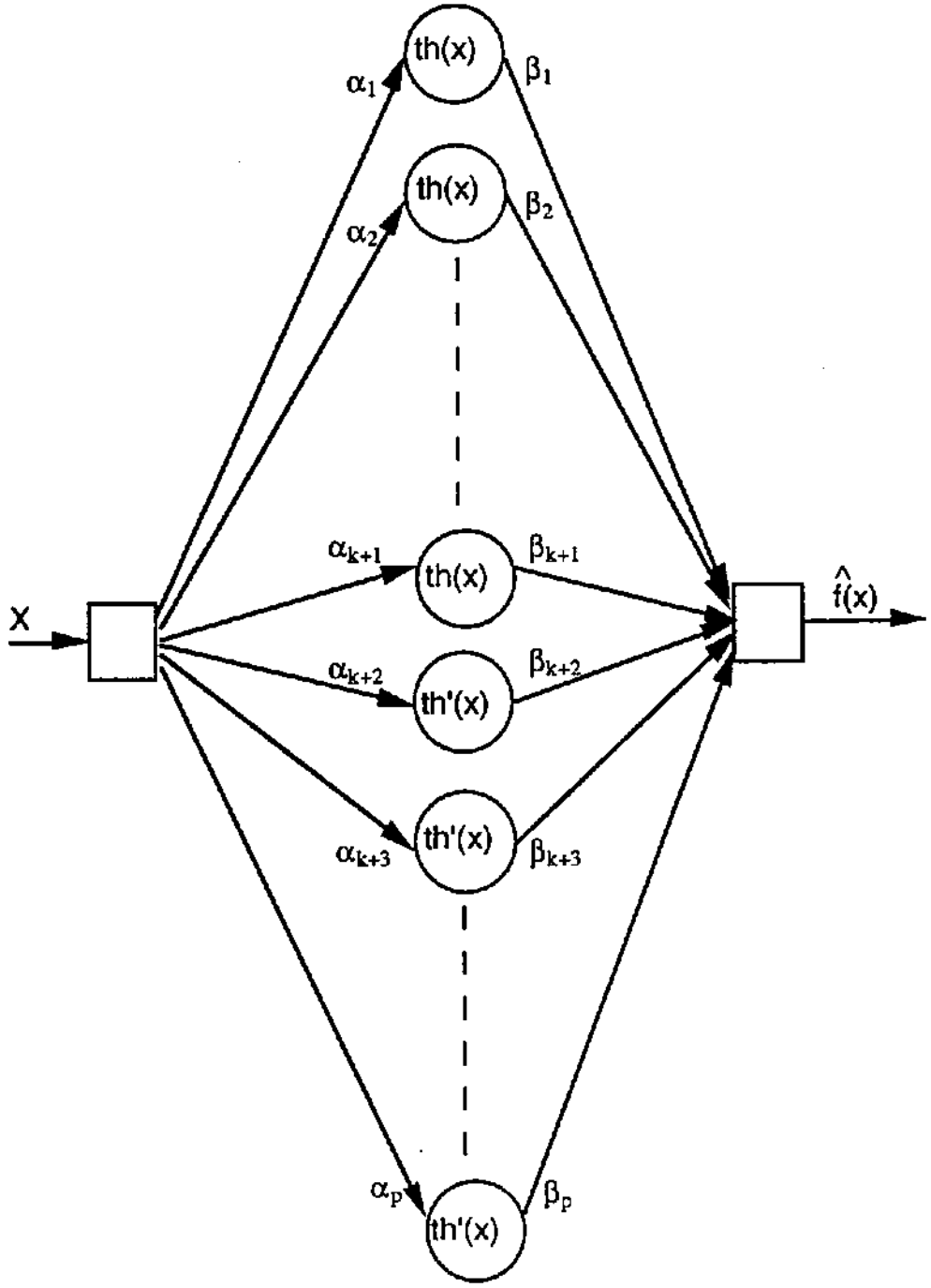

figure 53: Architecture du réseau de P neurones approchant le développement limité à l'ordre P de f(x)

CONCLUSION

En résumé, l'utilisation d'un réseau de neurones est à recommander lorsque l'on a un manque d'information sur l'expression de la commande ou sur le processus lui-même. Il va contrôler un processus tout en s'adaptant aux situations extérieures et en y répondant de manière appropriée conformément à une certaine expérience acquise par le passé.
Ceci est réalisé par une modification des paramètres internes du réseau (poids des connexions) de manière à ce que ses sorties approchent celles de la commande qui n'a pu être déterminée.
Toutefois, une certaine connaissance de cette commande étant toujours disponible, on a vu une façon de d'utiliser cette information pour construire et initialiser un réseau en utilisant une approximation de la fonction par les polynômes de BERNSTEIN.
Cependant, bien d'autres approximations existent (par les polynômes de LEGENDRE, de TCHEBYTCHEV), et, il serait intéressant de pouvoir les implanter sur des réseaux de neurones et de comparer les résultats obtenus afin de trouver la meilleure ou à défaut la plus adaptée pour un type de fonction donnée à approcher.
De toute façon, il est évident qu'un développement en série de fonctions sigmoïdes serait l'approximation la plus facile à implanter sur un réseau à 3 couches étant donné qu'il correspond à la relation entrée-sortie de ce réseau.
La détermination de ce développement sur le développement limité de la fonction à approcher aboutit à l'obtention d'équations non-linéaires fournissant le nombre de neurones de la couche cachée et leurs fonctions d'activation.
Cependant, ces équations, étant difficilement résolubles, donnent rarement la valeur des poids des connexions et laissent, donc, l'opportunité d'effectuer des recherches future dans cette direction.

BIBLIOGRAPHIE

[1] Behnam BAVARIAN
'Introduction to neural networks for intelligent control'
IEEE Control System Magagazine, April 1988, p. 3-7

[2] K. J. ASTRÖM
'Intelligent control'
Europ. Contr. Conf. 91, GRENOBLE, FRANCE, Hermes, 1991, p. 2328-2339

[3] Eric DAVALO, Patrick NAÏM
'Des réseaux de neurones'
Eyrolles, 1989, Chap. 6

[4] HECHT NIELSEN R.
'Neurocomputing'
Addison-Wesley, 1990, chap. 5, p. 122-137

[5] G. CYBENKO
'Approximation by Superposition of a Sigmoïdal Function'
Mathematics of controls, Signals, and Systems, 1989, vol. 2, p. 303-314

[6] K. HORNIK, M. STINCHCOMBE and H. WHITE
'Multilayer Feedforward Networks are Universal Approximators'
Neural Networks, vol. 2, 1989, p. 359-366

[7] D. RUMELHART, G. HINTON and R. WILLIAMS
' PARALLEL DISTRIBUTED PROCESSING'
CAMBRIDGE, MA: MIT Press, 1986, vol. 1, chap.8

[8] SI-ZHAO QIN, HONG-TE SU, and Thomas J. McAVOY
'Comparison of Four Neural Net Learning Methods for Dynamic System Identification'
IEEE Transaction on Neural Networks, vol.3, No 1, January 92, p. 122-130

[9] KUMPATI S. NARENDRA, KANNAN PARTHASARATHY
'Identification and Control of Dynamical Systems Using Neural Netwoks'
IEEE Transaction on Neural Networks, March 1990, vol. 1, No 1, p. 4-26

[10] FU-CHUANG CHEN
'Back-Propagation Neural Networks for Nonlinear Self-Tuning Adaptive Control'
IEEE Control System Magazine, April 1990, p. 44-47