11/02

République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

Ecole Nationale Polytechnique

D.E.R :Génie Electrique et Informatique

Automatique

Projet de Fin d'Etudes

Thème :

Identification et Commande par Réseaux de Neurones Application: bioréacteur

Présenté par :
BOUSSAOUD Brahim.
CHAIB Salim

Dirigé par :
M.ILLOUL.
M.SELATNIA.

Promotion : 2001/2002

ENP : 10 avenue Hassen Badi El Harrach Alger

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — مكتبة
Ecole Nationale Polytechnique

Remerciement :

Dédicace :

Brahim :

Je dédie ce modeste travail à :

- *ma mère, mon père, mes frères et sœurs et toute ma famille.*
- *Mes amis ,H, A, N, S, M(J), B et tous ceux que je connais sans exception.*
- *A tous les martyrs d'une Algérie libre et démocratique.*
- *A ceux qui travaillent pour la science.*

Salim :

- *A mes parents.*
- *Mes frères, surtout Yacine, mes sœurs, mes amis : Sid Ali, Amine, Mehdi, Messaoud.*

Sommaire

INTRODUCTION GENERALE

Les *réseaux de neurones artificiels* sont une alternative prometteuse pour la modélisation de phénomènes complexes présentant des dynamiques complexes, variant avec le temps, non linéaires et/ou à paramètres distribués. Or, une bonne modélisation est généralement la clef du contrôle de tels systèmes, nombreuses étant les théories de l'automatique qui reposent sur la notion de modèle. Le domaine des bio-procédés fournit de tels problèmes et constitue aujourd'hui un domaine de recherche d'intérêt majeur pour l'industrie.

Plusieurs approches neuronales pour la modélisation (de paramètres de croissance ou des variables d'état) des procédés biotechnologiques ont été abordées. Les modèles neuronaux ont été utilisés pour la prédiction de variables d'état : concentration en biomasse, substrat, produit. Ils ont été inclus dans des stratégies de commande prédictive, puis de supervision. Les estimations, par exemple, de la biomasse et de la vitesse de croissance de la biomasse pour une fermentation, à partir de mesures de gaz et temps ont été réalisées avec des modèles neuronaux.

Le manque de capteurs en ligne se faisant cruellement ressentir dans l'industrie agroalimentaire, l'écriture de modèles complexes peut parfois permettre de disposer en ligne d'informations permettant de palier ce manque. Dans cette optique, les travaux réalisés sur la conduite d'une croissance bactérienne ont permis de mettre en évidence l'importance d'une analyse fine des phénomènes pour, à la fois, avoir une structure de modèle adéquate et pouvoir effectuer des simplifications pertinentes.

Dans ce présent travail, nous avons à commander un bioréacteur, qui présente de fortes nonlinéarités, on a fait la modélisation de ce système, ensuite on a abordé la théorie des réseaux de neurones artificiels, et les différents algorithmes sur lesquels repose cette technique, ensuite, on a illustré les différentes structures d'identification, pour différents systèmes, ainsi on a exposé les différentes structure de commande, enfin des simulations qui permettent de voire les résultats donnés par cette technique.

I.1 Introduction :

Les micro-organismes sont capables d'effectuer une grande diversité de réactions chimiques qui se traduisent soit par la production de biomasse c'est à dire de corps cellulaires, soit par la production ou la transformation de substances organiques et parfois minérales. En biotechnologie, il est courant d'employer le terme **"fermentation"** pour définir la plupart des cultures et réactions microbiennes, bien qu'il ne s'agisse pas toujours d'une fermentation au sens biochimique.

C'est le Hollandais Antonie Van Leeuwenhoek (1632-1723) qui révéla à l'homme l'existence du monde microbien, en utilisant des microscopes simples (en fait des loupes) d'une qualité nettement supérieure à celle de microscopes composés existant depuis la fin du XVIéme siècle.

La microbiologie étudie sous leurs multiples aspects les micro-organismes, c'est à dire les êtres vivants si petits qu'ils ne peuvent être observés qu'au microscope. L'essor de la microbiologie au cours du XVIéme siècle est marqué par la découverte du rôle des micro-organismes dans la transformation de matière organique et la genèse des maladies. L'étude des fermentations amena la découverte de leur nature biologique : les fermentations sont produites par des micro-organismes (levures ou bactéries) qui se multiplient ; à des fermentations différentes correspondent des micro-organismes morphologiquement et physiologiquement différents.

Lié d'abord aux améliorations techniques des instruments d'observation, puis préparé par les progrès de la chimie organique et de la physiologie, Louis Pasteur (1822-1895) révolutionne par ses travaux la médecine, la chirurgie, l'industrie, l'agriculture. Parallèlement Pasteur mit en évidence la présence de micro-organismes dans l'atmosphère. Il montra que l'introduction et le développement de ces micro-organismes dans certains liquides organiques sont la cause de leurs altérations et mit fin ainsi à une controverse entre scientifiques de l'époque sur la croyance à la génération spontanée d'êtres vivants microscopiques à partir de matières organiques inertes.

I.2 Définition de la fermentation :

Lorsqu'un micro-organisme est introduit dans un milieu de culture lui convenant bien, il y développe une activité en relation avec sa composition et les conditions environnantes. Cette activité peut présenter deux aspects liés l'un à l'autre :

- Le microbe se reproduit ; il en résulte une augmentation de la concentration en biomasse microbienne, la biosynthèse des constituants cellulaires se faisant à partir des composés du milieu de culture ;

- Parallèlement, il arrive, et c'est ce qui intéresse le biotechnologue, que le microbe excrète, au travers de ses enveloppes cellulaires, tel ou tel composé intéressant, qui sera concentré, purifié et récupéré en fin de culture.

Mis à part les cas où la biomasse elle-même est le but ultime d'une fermentation (levurerie, production de protéines d'organismes unicellulaires), c'est en général la production d'un **métabolite bactérien** qui est visée par le bio-industriel.

On a coutume d'en distinguer deux types :

- Les métabolites **primaires** indispensables à la vie des micro-organismes et dont certains font l'objet d'une production industrielle : aminoacides, nucléotides, vitamines, acides organiques.

- Les métabolites **secondaires**, synthétisés généralement tardivement dans le processus de croissance ; on distingue alors :

- Les antibiotiques,
- Les mycotoxines,
- Les pigments.

Les premiers sont, de très loin, les plus importants industriellement puisque la totalité des antibiotiques et des motifs chimiques ultérieurement modifiables par hémisynthése sont obtenus par voie fermentaire.

I.3 Description du problème de commande d'un bioréacteur :

I.3.1 Modélisation :

Le bioréacteur peut être considéré comme relativement simple, car il possède peu de variables, mais encore difficile à commander à cause de ses fortes nonlinéarités qu'on ne peut modéliser d'une manière précise. Sous sa forme la plus simple, le bioréacteur est un réservoir contenant de l'eau et des cellules (bactéries ou levures) qui consomment des éléments nutritifs (substrat) et produisent certains produits et plus de cellules. Le bioréacteur est d'autre part compliqué, car les cellules peuvent changer leur taux de croissance et de production radicalement, suivant la température et la concentration des déchets (alcool). Pour un début, un modèle relativement simple du bioréacteur constitue le meilleur choix.

La version la plus simple du problème est un réservoir à flux continu dans lequel la croissance des bactéries dépend uniquement des éléments nutritifs donnés au système. La grandeur à contrôler est la concentration des bactéries dans le réservoir. Un tel système est régi par le système d'équations suivant :

$$\begin{aligned}\frac{dC_1}{dt} &= -C_1 w + C_1(1-C_2)\exp\left(\frac{C_2}{\gamma}\right)\\ \frac{dC_2}{dt} &= -C_2 w + C_1(1-C_2)\exp\left(\frac{C_2}{\gamma}\right)\frac{1+\beta}{1+\beta-C_2}\end{aligned} \qquad (I.1)$$

Où :
C_1 : la concentration relative des bactéries.
C_2 : la concentration relative convertie du substrat avec :
w : le taux du flux à travers le réacteur.
γ : le taux de consommation des éléments nutritifs.
β : le taux de croissance des cellules.

$$C_2 = \frac{S_f - S}{S_f} \qquad (I.2)$$

S_f : la concentration relative du substrat dans le flux.
S : la concentration relative du substrat dans le réacteur.

La première équation montre que la variation de la quantité des bactéries dans le réservoir est égale à la quantité de substrat qui sort de celui-ci ($C_1 w$) plus la quantité qui s'ajoute due à la croissance des cellules ($C_1(1-C_2)\exp(C_2/\gamma)$).

La deuxième équation montre que la variation de la quantité de substrat (C_2) dans le réservoir est égale à la quantité qui quitte celui-ci plus la quantité de substrat métabolisé par les bactéries.

Ce système est difficile à commander pour différentes raisons :

- Les équations qui régissent ce système sont fortement nonlinéaires.
- Un comportement optimal du système est proche de la région d'instabilité.
- Deux valeurs de l'action de commande (w) peuvent mener le système au même état permanent (quantité des cellules).

La discrétisation en utilisant l'algorithme d'Euler donné du modèle continu par (I.1) :

$$C_1(k+1)=C_1(k)+h(-C_1(k)w(k)+C_1(k)(1-C_2(k))\exp(\frac{C_2(k)}{\gamma})$$

$$C_2(k+1)=C_2(k)+h(-C_2(k)w(k)+C_1(k)(1-C_2(k))\exp(\frac{C_2(k)}{\gamma})\frac{1+\beta}{1+\beta-C_2(k)}$$

Où : h : la constante d'échantillonnage.

Pour la simulation, nous prendrons pour paramètres les valeurs suivantes : β=0.02, γ=0.48, h=0.001. $C_1(0)$, $C_2(0)$ et w(0) sont aléatoires telle que $C_1(0)\geq 0$, $C_2\leq 1$.

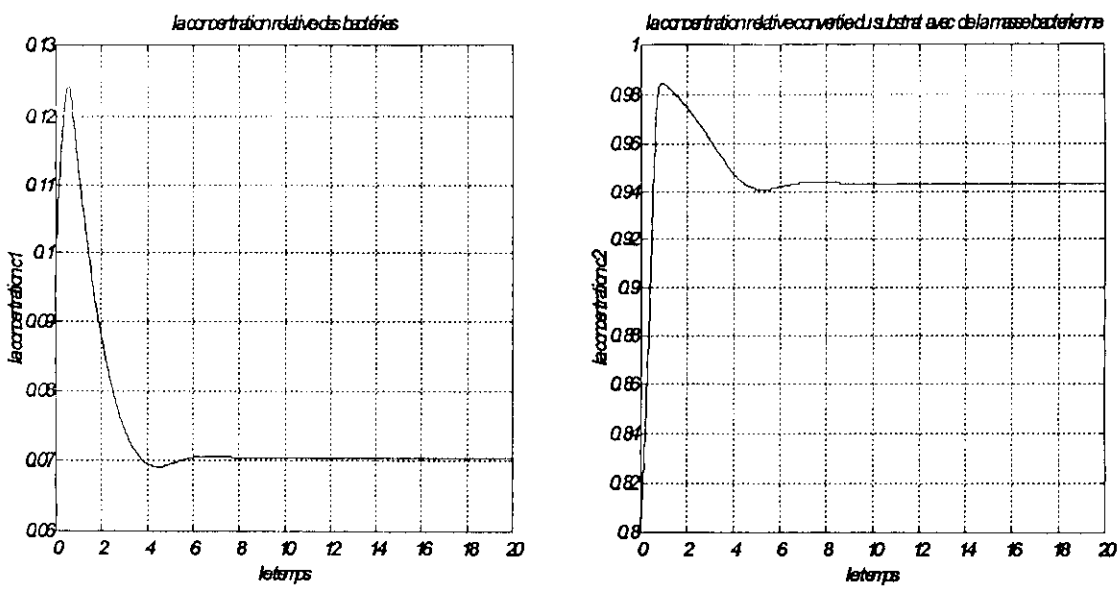
Ainsi, on obtiendra le système suivant :

$$C_1(k+1)=C_1(k)+0.001(-C_1(k)w(k)+C_1(k)(1-C_2(k))\exp(2.0833C_2(k))$$

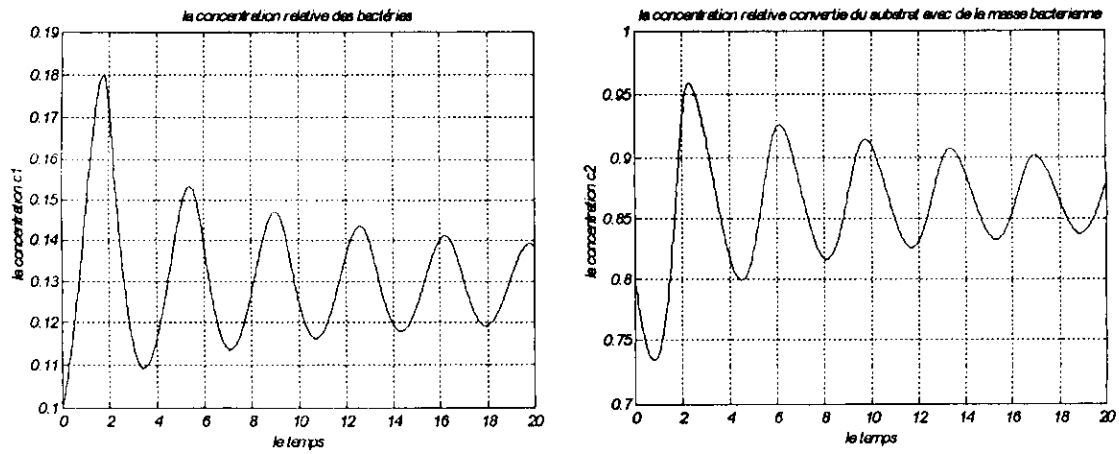
$$C_2(k+1)=C_2(k)+0.001(-C_2(k)w(k)+C_1(k)(1-C_2(k))\frac{1.02}{1.02-C_2(k)}\exp(2.0833C_2(k))$$

I.3.2 Simulation du système en boucle ouverte :

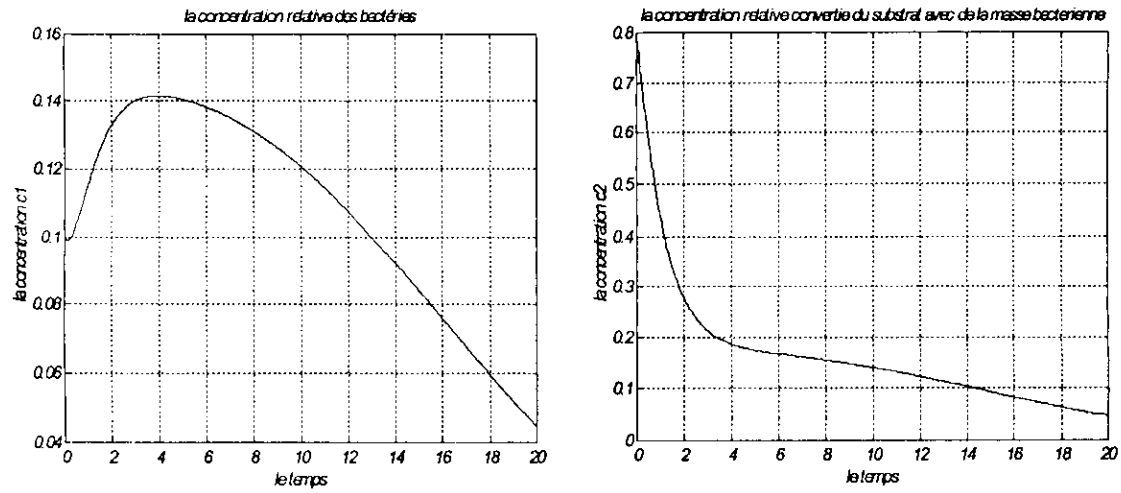
Pour des valeurs différentes de flux (w), on verra le comportement du système :



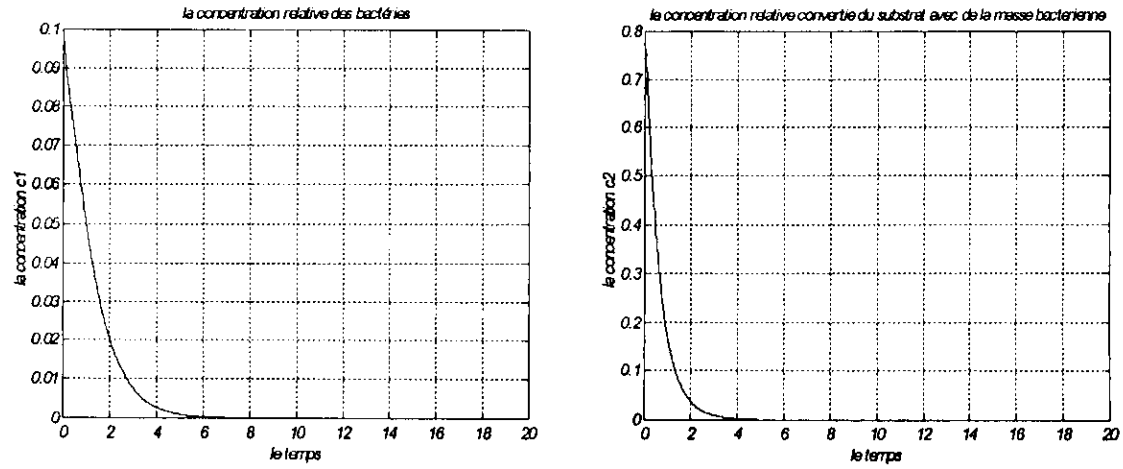
Fig(I.1): la réponse du système pour $C_1(0)$=0.1 $C_2(0)$=0.8 w=0.4



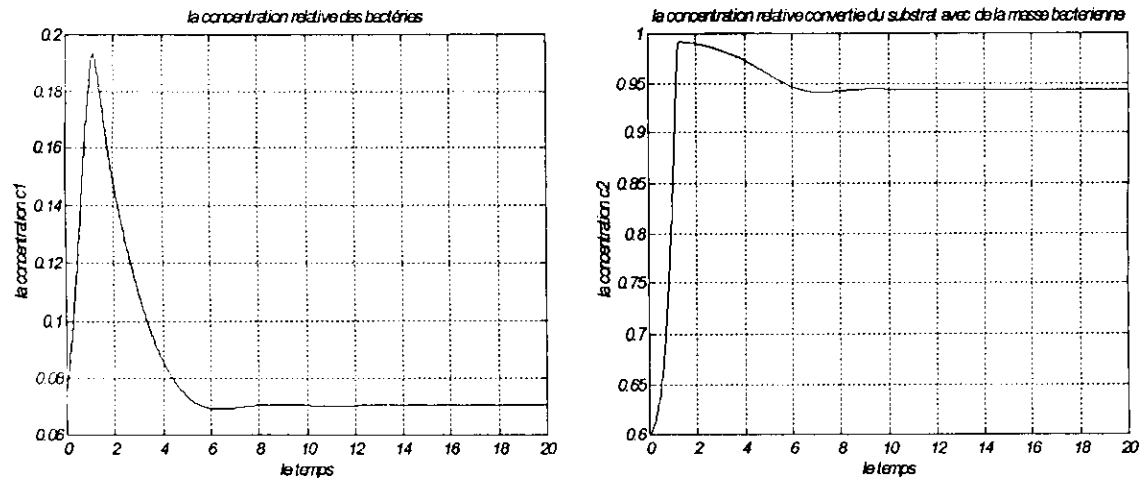
Fig(I2): la réponse du système pour $C_1(0)=0.1$ $C_2(0)=0.8$ $w=0.8$



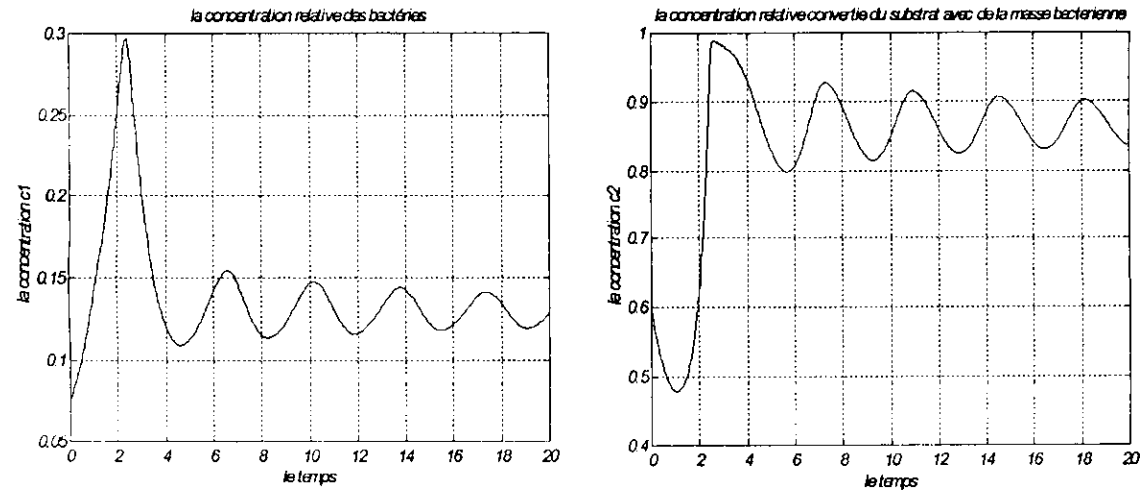
Fig(I3): la réponse du système pour $C_1(0)=0.1$ $C_2(0)=0.8$ $w=1.2$



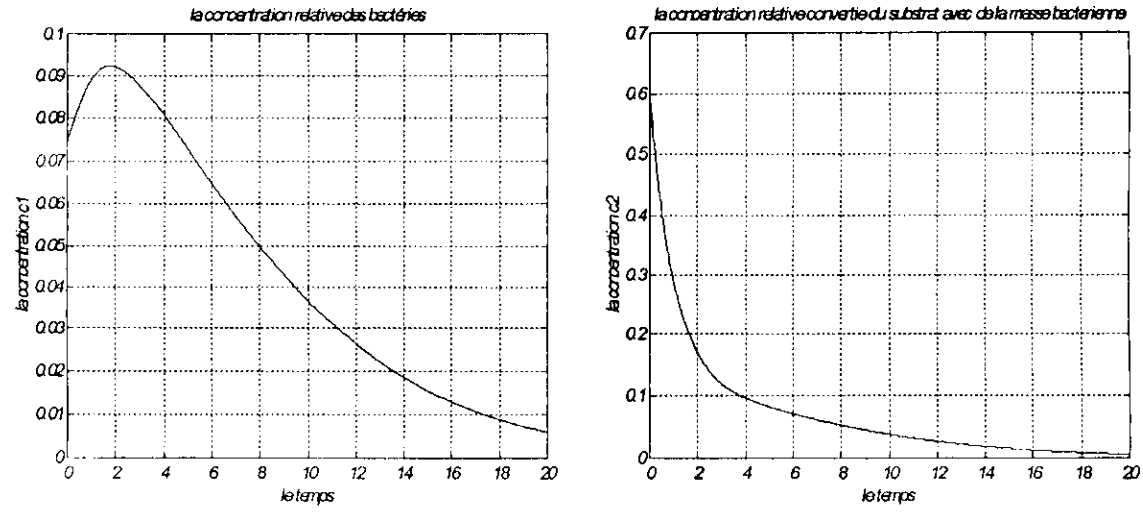
Fig(I4): la réponse du système pour $C_1(0)=0.1$ $C_2(0)=0.8$ $w=2$



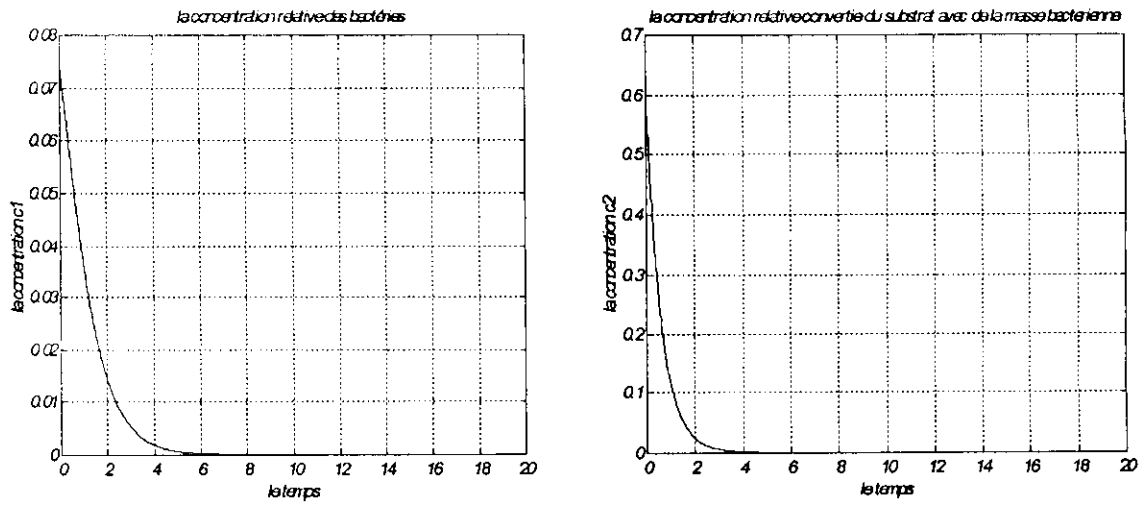
Fig(I.5): la réponse du système pour $C_1(0)=0.075$ $C_2(0)=0.6$ w=0.4



Fig(I.6): la réponse du système pour $C_1(0)=0.075$ $C_2(0)=0.6$ w=0.8



Fig(I.7): la réponse du système pour $C_1(0)=0.075$ $C_2(0)=0.6$ w=1.2



Fig(I.8): la réponse du système pour $C_1(0)=0.075$ $C_2(0)=0.6$ $w=2$

On remarque des simulations précédentes que le comportement du système en boucle ouverte dépend non seulement de la commande mais aussi des conditions initiales, on voit que dans les figures (I.2) et (I.6) que pour la même commande $w=0.8$, dans la figure (I.2) la concentration C_1 atteint une valeur de 0.18 mais parcontre elle atteint 0.3 dans la figure (I.6).

Comme, on remarque également que la plage de commande est répartie en trois partie, deux plages de stabilité allant de $w=0$ à $w=0.7$ et $w=1.2$ à $w=2$ et une plage d'instabilité allant de $w=0.7$ à $w=1.2$.

II.1 Historique

Classiquement, l'histoire commence ainsi : en 1943, Mc Cullogh et Pitts inventèrent le premier neurone formel (analogie directe avec le neurone biologique), un produit scalaire entre vecteur d'entrée et un vecteur poids, suivi d'un élément à seuil répondant 0 ou 1 selon que le résultat du produit scalaire est supérieur ou non au seuil. Elle se poursuit ainsi : puis dans les années 1950, à l'époque où Hebb proposait le principe de la plasticité synaptique comme modèle de mémoire. Rosenblatt inventa le « Perceptron », identique au neurone formel, mais doué de la faculté d'apprentissage; c'est un séparateur linéaire qui permet d'apprendre à fournir la bonne décision dans des opérations de classification. Vinrent ensuite Minsky et Papert en 1960, qui démontrèrent qu'avec un perceptron à une couche de neurones, on ne pouvait séparer que des données linéairement séparables et que pour des opérations plus générales de classification, il fallait mettre plusieurs couches. Seulement là, on ne savait pas comment faire apprendre les bons vecteurs poids. Il y eu ensuite un trou dans les années 70 où rien ne passa, jusqu'à ce que le renouveau apparaisse en 1982 avec Hopfield et les neurones sur le modèle des verres de spin.

Parmi les premières approches, il est intéressant de citer celle de Sigmund Freud, à une époque où les notions de physiologie étaient rudimentaires et où les sciences des systèmes étaient inconnues. Voici comment il imaginait, à partir de ses réflexions sur la psychanalyse, les prémices de la plasticité neuronale :

« Il y a des neurones perméables (n'opposant aucune résistance et ne retenant rien), qui servent à la perception, et des neurones imperméables (affectés de résistance et retenant Qn), qui sont porteurs de mémoire, vraisemblablement donc des processus psychiques en général ».

Revenons à ce trou des années 70 : ce fut au contraire de ce que disent certains auteurs, une période extrêmement riche où des chercheurs comme Anderson, Kohonen et bien d'autres ont travaillé ces idées, apprenant progressivement à les formaliser en combinant les méthodes de l'algèbre linéaire et celles des techniques non linéaires. Il est vrai cependant que le brio des théories de Hopfield, épaulé par un engouement des physiciens à partir de 1982, a été le moteur d'un renouveau mondial sur les neurones artificiels. Comme avec toutes les belles théories, on y avait un trop cru, et quelques chercheurs pas trop optimistes ont failli y perdre leur âme. Il n'en reste pas moins que les "adeptes" des réseaux neuromimétiques, du connexionisme, sont de plus en plus nombreux.

II.2 Pourquoi des neurones artificiels ?

Capter une image, la numériser, la segmenter en élément de contours, détecter un objet mobile, le reconnaître quelle que soit sa position et estimer sa profondeur. Capter le son d'une voix au milieu d'un brouhaha et du bruit ambiant et reconnaître les mots qui sont prononcés. Capter et analyser les deux cents signaux issus d'un processus industriel et déduire si tout est conforme ou si une avarie se prépare : voici quelques problèmes pourtant courants dans les sciences de l'ingénieur mais dont les solutions, encore incomplètes, impliquent de multiples efforts de recherche dans la communauté scientifique.

Malgré la constante augmentation de puissance des calculateurs, malgré les approches théoriques de plus en plus sophistiquées, un certains nombre de tâches résistent encore aux algorithmes et aux méthodes classiques de traitement des signaux et des données. Ces tâches

relèvent typiquement du traitement, en temps réel, de très grands flots de données souvent multidimensionnelles et arrivant à des cadences élevées. Le grand nombre des données, leur validité, le fait qu'elles ne répondent pas à des modèles physiques connus nous laissent souvent démunis devant des tâches de caractérisation, de reconnaissance et de prise de décision.

Or, toutes ces opérations se trouvent réalisées de manière naturelle chez les êtres vivant : un visage est reconnu en quelques dixièmes de seconde, une voix est reconnue au milieu du bruit ambiant et le discours est perçu, le contrôle simultané d'une centaine de muscles en fonction de la position et de la survenue d'un obstacle dans le champ visuel est réalisé en quelques dixièmes de secondes. Pour une machine, réaliser de telles tâches serait une véritable prouesse, et nous, nous les réalisons sans même y penser ! C'est là que réside l'attrait des neurones artificiels. Copier le cerveau restera pour longtemps encore une ambition exagérée, mais vouloir s'inspirer des architectures et des fonctions du système nerveux n'est pas un rêve inaccessible. Le développement continuel des connaissances en biologie, l'apparition de nouvelles méthodes théoriques et l'incessante montée en puissance des outils de simulation nous autorisent les meilleurs espoirs.

Neuromimétique ou connexionniste ? Le terme neuromimétrique implique "ressemblance avec les neurones". En fait, la ressemblance est souvent très vague et ce terme peut se prêter à diverses interprétations. Dans le cas présent, les caractéristiques essentielles des réseaux de neurones réels que nous conserverons dans les modèles mathématiques étudiés, concernent le grand nombre de connexions, la non-linéarité des relations entrée-sortie et la faculté de plasticité ou d'adaptabilité. Ces caractéristiques, même simplifiées, leur confèrent déjà de multiples possibilités en traitement des signaux et des informations ainsi que la faculté d'apprendre à classer, à reconnaître des formes ou à réaliser des tâches complexes. Le terme "connexionniste"est aussi souvent employé pour qualifier cette approche, afin de conserver une certaine distance par rapport au modèle biologique.

II.3 Le Neurone biologique

Le neurones est l'élément de base du système nerveux. Ce dernier possède à peu prés 100 à 1000 milliards de neurones ,le neurones est cellule constituée de trois parties comme l'indique la figure suivante :

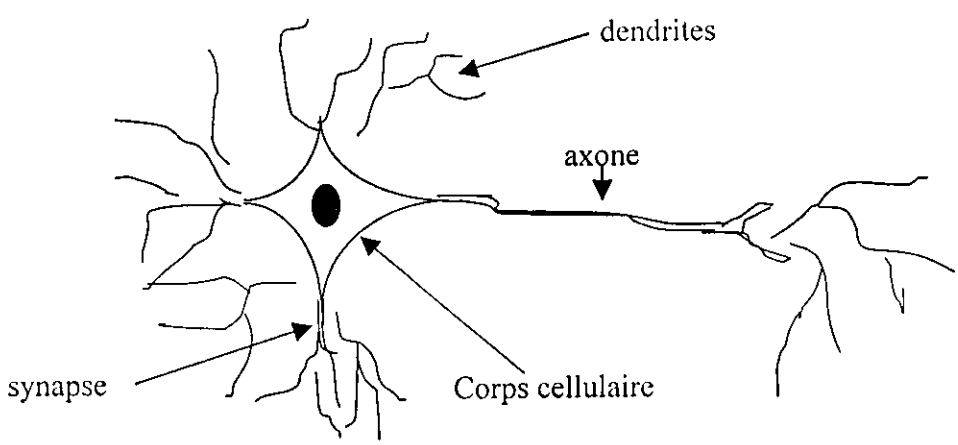


Figure II.1
Le neurone biologique

- **Le corps cellulaire :**

  -Réception des influx nerveux.
  -Traitements des influx nerveux.
  - Activation du neurone.

- **Axone :**

  - Transmission du potentiel d'action.

- **Synapses :**

  - Transmission du signal provenant de l'axone par transformation électrique-chimique grâce aux neurotransmetteurs.
  - Réception du signal par les dendrites par transformation électrique-chimique grâce aux neuro-récepteurs.

- **Dendrites :**

  - Acheminement des signaux reçus des autres neurones vers le corps cellulaire.

Le neurones reçoit des signaux de neurones voisins, il intègre ces signaux et engendre un influx nerveux, le conduit et le transmet à un autre neurone qui peut le recevoir. L'influx nerveux est une impulsion électrique, et les signaux dendritiques et somatique sont des variations de potentiel électrique. La transmission entre deux neurones se fait à l'aide d'un médiateur chimique appelé ***neurotransmetteur*** **.**

Le Soma recueille l'ensemble des information reçues par les dendrites et fait leur somme ,si le potentiel somatique dépasse un certain seuil, il y'aura émission d'un potentiel d'action. Ce signal très bref (1ms) est transmis le long de l'axone sans atténuation, et réparti sur les synapses des neurones cibles grâce à l'arborisation terminale.

En général ,le système nerveux est formé par des neurones qui sont connectés les uns aux autres suivant des répartitions spatiales complexes. Les connexions entre deux neurones se font en des endroits appelés ***synapses*** ou ils sont séparés par un petit espace synaptique de l'ordre d'un centième de microns.

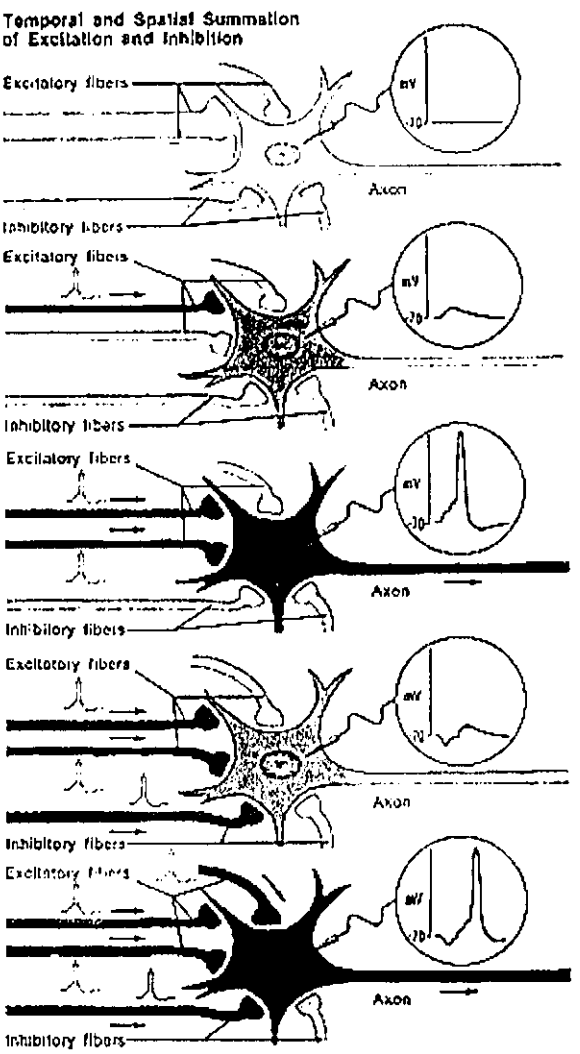


Figure III.2
Fonctionnement du neurone biologique

II.4 Les réseaux de neurones artificiels

II.4.1 Modélisation simplifiée du neurone biologique

Le réseau de neurones est graphe orienté et pondéré. Les nœuds de ce graphe sont des automates simples nommés neurones, et dotés d'un état interne (l'activation) qui se propage dans le graphe le long d'arcs pondérés appelés lins synaptiques.

Le modèle théorique du neurone trouve ces origine dans la neurobiologie. C'est en 1943 que McCulloch et Pitts ont émis l'idée simplificatrice du neurone formel, c'est -à-dire un opérateur binaire interconnecté à ses semblables par des synapses excitatrices ou inhibitrices. Une semblable de tels opérateurs a des propriétés collectives, capable de faire certains calculs que chacun d'eux est incapable de l'exécuter séparément.

Le neurone formel est modélisé par certains opérateurs qui sont :

1. **Opérateur de sommation :**

$$P=\sum_{i=1}^{n} w_i x_i \qquad \text{(II.1)}$$

Ou :

P : le potentiel somatique.
x_i : entrée du neurone i.
w_i : poids synaptiques.

2. **Fonction d'activation :**

$$y=f(P) \qquad \text{(II.2)}$$

D'où on aura la relation entre la sortie et les entrées :

$$y=f\left(\sum_{i=1}^{n} w_i x_i\right) \qquad \text{(II.3)}$$

on représente le neurones par le modèle simplifié suivant :

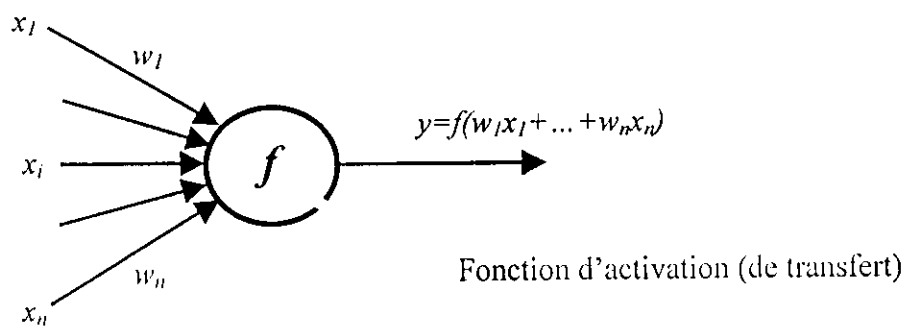


Figure II.3
Modèle formel d'un neurone

II.4.2 La règle de Hebb

Hebb à proposé en 1949 une règle qui montre que le renforcement synaptique intervient lorsqu'il y'a activité du neurone pré-synaptique (cause) et activité du neurone post-synaptique
(effet).Cette règle est la suivante :

$$w_{ij}(t+\delta t)=w_{ij}(t)+\eta.a_i.a_j \qquad \eta>0 \qquad \text{(II.4)}$$

Avec :

$w_{ij}(t)$: poids de la connexion entre le neurone j et le neurone i .

$a_i.a_j$: Les activation des neurones i et j .

η : paramètre reflétant l'intensité de l'apprentissage .

La plus part des modèles de neurones sont capables de modifier leurs poids synaptiques automatiquement, ils sont dotés de **règle d'apprentissage.**

II.4.3 Modèle mathématique générale du neurone

Le neurone formel est représenté par le modèle suivant :

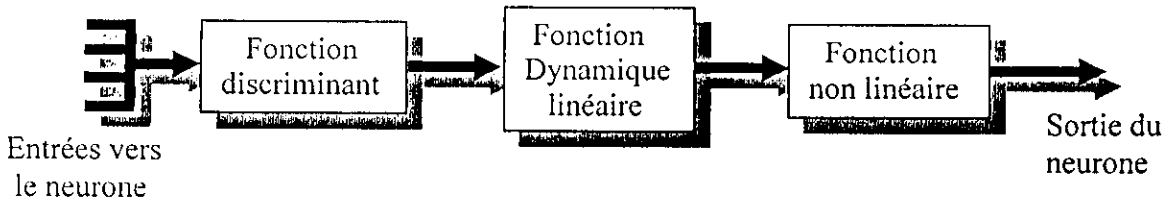


Figure II.4
Schéma des différentes fonctions mathématiques
à l'intérieur d'un neurone formel

1. Fonction discriminant

Cette fonction, appelée aussi fonction de base (Basis fonction) ,définit l'activité du neurone. dans le cas du perceptron , c'est la fonction linéaire qui est utilisée .C'est le cas pour les réseaux multicouches et d'une manière générale les réseaux LBF (Linear Basis Fonction). Les fonction d'activation de base utilisées se résument comme suit :

a. la fonction de base linéaire LBF (Linear Basis Fonction)

Cette fonction est une sommation pondérée des entrées vers le neurone, elle est très efficace en plus de sa elle est la plus simple à implémenter. Sa forme est en générale définie par :

$$f(X,W)=X^T.W \qquad \text{(II.5)}$$

W : matrices des poids .

X :vecteur des entrées du neurone .

b. La fonction de base radiale RBF (Radial Basis Function)

La forme de cette fonction est inspirée des réseaux de neurones utilisés pour la classification notamment les ART.

$$f(X,W)=-\frac{\|X-W\|^2}{2} \qquad \text{(II.6)}$$

En utilisant cette fonction de base, les poids W sont présentés comme étant les centroïdes de chaque classe de l'espace d'entrée. Le *discriminant* calcul le rayon entre les centres de chaque classe. Cette fonction sert à effectuer un échantillonnage de l'espace des entrées, ou chaque groupe de poids synaptique représente uns concentration de données.

c. La fonction de base elliptique EBF (Elliptic basis Function)

La forme de ce discriminant est une généralisation de la fonction de base radiale :

$$f(X,W)=\sum_{k=1}^{n}\alpha_k(X-W)^2+\theta \qquad \text{(II.7)}$$

Cette fonction est modulée par des paramètres α_k et dotée d'un biais θ. Elle conserve les mêmes caractéristiques que la précédente, mais elle est très dépendante de ces paramètres qui déterminent les formes de zones de regroupement de données dans l'espace d'entrées .

2. Fonction dynamique linéaire

Cette fonction est régie par l'équation différentielle de premier ordre suivante :

$$\alpha_0\dot{u}_i(t)+\alpha_1 u_i(t)=v_i(t) \qquad \text{(II.8)}$$

Ou :

u_i : représente l'activité du neurone i .
v_i :représente l'entrée du système dynamique décrit par l'équation précédente.

II.4.4 Réseau de neurones

L'assemblage de plusieurs neurones (éléments linéaires adaptatifs), où chaque élément est relié à d'autre par le biais de ses entrées et sorties, est appelé réseau de neurones. Un réseau neuronal n'est autre qu'un système connexionniste complexe, formé de nombreuses interactions entres les éléments, leur permettant ainsi de travailler ensemble pour résoudre les problèmes. Ces éléments ont un comportement individuel simple, leurs influences mutuelles décident du comportement global du réseau.

Différents modèles et types de réseaux de neurones ont été établis, dont les plus importants sont résumés comme suit :

- 1943 : **Le neurone formel** (McCulloch & Pitts)
- 1948 : **Les réseaux d'automates** (Von Neuman)

- 1949 : **Première règle d'apprentissage (Hebb)**
- 1958 : **Premier modèle des réseaux neuronaux**, utilisé pour la reconnaissance des formes. Il comprenait : une rétine (couche d'entrée), une couche de cellules d'association (couche cachée), et une couche de cellules de décision (couche de sortie).
- 1960 : **Le madaline de widrow**
  Assemblage de plusieurs éléments adaptatifs sous forme de couche. Addition d'un terme biais à la somme pondérée et d'une fonction binaire à la sortie du neurone.
- 1969 : **Perceptrons (Minsky & Papert)**
  les limites du Perceptron.
  besoin d'architectures plus complexes.
  Comment effectuer l'apprentissage ? On ne sait pas !
- 1982 : **Le réseau de Hopfield**
  Modélisation d'une mémoire associative. Le réseau est entièrement connecté, les neurones n'étant pas reliés à eux-mêmes.
- 1983 : **La machine de Boltzmann**
  Amélioration de l'algorithme d'apprentissage d'un réseau de Hopfield.
- 1984 : **Le modèle de Kohonen**
  Le réseau se distingue par une auto-adaptation, une projection lui permettant de compresser les données et une bonne résistance aux bruits.
- 1986 : **Rétropropagation (Rumelhart & McClelland)**
  nouvelles architectures de Réseaux de Neurones.
  applications :
  - reconnaissance de l'écriture.
  - reconnaissance/synthèse de la parole.
  - vision (traitement d'images).
- 1990 : **« Société de l'Information »**
  nouvelles applications :
  - recherche/filtrage, extraction d'information, multimédia, besoin de combiner différents modèles

II.4.5 Différentes architectures des réseaux de neurones

Les réseaux neuronaux peuvent être classés suivant leur architecture en trois classes essentielles :

- réseaux feed-forward.
- réseaux récurrents.
- cartes topologiques.

II.4.5.1 Réseaux feed-forward (Réseaux multicouches)

Le réseau est constitué de plusieurs couches, comportant chacune un nombre donné d'éléments adaptatifs. Les éléments d'une seule couche ne sont pas reliés entre eux. Le neurone de chaque couche est relié à tous les neurones de la couche précédente par le biais de ses entrées et à tous les neurones de la suivante par le biais de sa sortie. Le réseau possède

une couche d'entrée liée directement au vecteur d'entrée X, et une couche de sortie donnant les signaux accessibles du réseau. Les couches dont les sorties des éléments ne sont pas accessibles sont dites couches cachées. On les introduit dans le réseau pour augmenter sa capacité à simuler des fonctions de plus en plus complexes.

Dans ce type de réseaux, la propagation des signaux se fait vers l'avant, de la couche d'entrée vers la couche de sortie. On les nomme alors réseaux à propagation avant ou feedforward networks.

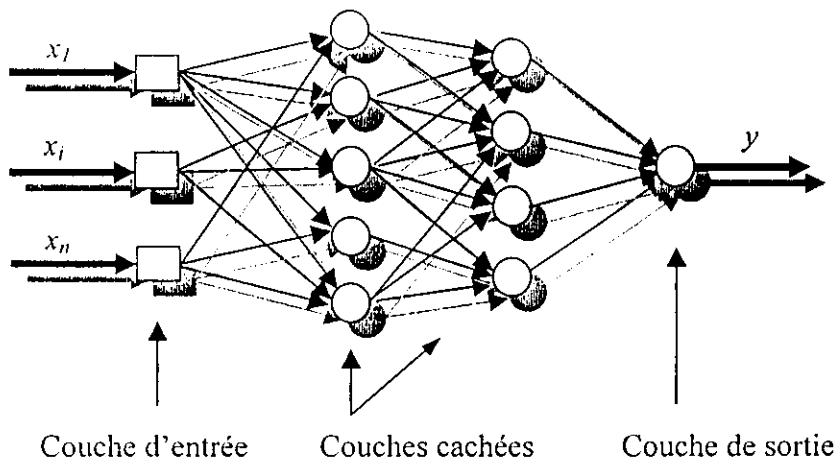


Figure II.5
Réseaux feedforward

Dans cette même structure de réseaux, on distinct aussi les réseaux entièrement connectés ou chaque neurone communique avec tous les autres. La manière avec laquelle le réseau s'organise fut définie par D.O.Hebb, et est basée sur des déductions biologiques ; elle stipule que l'activation simultanée de deux neurones tend à renforcer leur connexion.

La véritable approche de ce type de réseaux était établie par Hopfield en 1982, et selon lui le réseau doit chercher un état stable parmi plusieurs présentés lors de l'apprentissage. Bien qu'il semble parfois difficile à entraîner, ce réseau est le plus proche de la réalité.

II.4.6 Apprentissage des réseaux de neurones

L'apprentissage est défini comme étant n'importe quel changement opéré dans la mémoire du réseau. Ainsi cette modification affecte les poids synaptiques qui relient les neurones entre eux.

$$\text{Apprentissage} \equiv \frac{dw}{dt} \neq 0 \qquad \text{(II.9)}$$

L'apprentissage a comme objectif l'amélioration des performances futures du réseau, sur la base d'une connaissance acquise au fur et à mesure des expériences passées. Le mécanisme d'apprentissage diffère suivant la tâche pour laquelle ce réseau est utilisé. Il existe principalement deux types d'apprentissage différents :

a. Apprentissage supervisé

Il se fait en présence d'un superviseur (teacher) qui dirige le comportement du réseau en lui présentant les couples d'entrées et leurs sorties désirées. Cet apprentissage se fait toujours par l'intermédiaire d'un critère à optimiser définissant la performance du réseau à chaque étape. La dynamique de l'erreur du réseau pour les exemples présentés est donnée par la relation :

$$\Delta w = f_w(w, E, \eta) \qquad \text{(II.10)}$$

Ou :

Δw :la fonction qui calcule la modification des poids w.
E : l'erreur d'apprentissage .
η :le pas d'apprentissage.

Les réseaux dont l'apprentissage est fondé sur ce type sont :

- Perceptron de Rosenbalt.
- Réseau de Hopfield.
- Réseau de Hamming.

b. Apprentissage non supervisé

L'apprentissage non-supervisé nécessite la présence des entrées seulement sans l'intervention d'un superviseur. Cet entraînement se fait sur la base d'informations locales existant aux niveaux des neurones et découvre les propriétés collectives qui existent entre les données sur la base desquelles le réseau doit s'organiser.

c. Apprentissage semi supervisé

Ce type d'apprentissage est moins classique que les deux premiers qui sont les principaux. Il à d'une part en commun avec l'apprentissage supervisé, la présence d'un critère qui juge l'évolution de l'apprentissage, et d'autre part, il ne nécessite que des entrées sans définir les sorties désirées comme dans l'apprentissage non supervisé.

Dans se type d'apprentissage, le réseaux réajuste ses poids synaptiques suivant un critère de performance. Dans ce cas l'information est implicite, sous forme d'une simple appréciation (bon ou mauvais), sans mesure d'erreur.

II.4.7 Algorithmes d'apprentissage

La phase d'apprentissage est une étape déterminante dans la conception du réseau de neurones. Pour cela, des algorithmes ont été élaborés et développés au fil des années pour être appliqués, chacun à un type spécifique de réseaux. Ces méthodes sont basées sur des techniques mathématiques connues appliquées dans divers domaines où ils ont prouvé leur efficacité.

II.4.7.1 Algorithme de Backpropagation:

Cet algorithme est utilisé pour une topologie de réseaux multicouches, l'apprentissage y est supervisé. Il est basé sur une méthode numérique dite de relaxation qui effectue une descente de gradient sur la surface d'erreur quadratique moyenne.

Elle utilise donc les techniques de dérivées partielles. Ceci étant, on est amené à utiliser des fonctions d'activation dérivables appelées sigmoïdes,

$$F(x) = 1/(1+\exp.(-ax)). \qquad (II.11)$$

Le principe de la méthode est résumé dans la figure (II.6). Il s'agit de présenter au réseau un vecteur d'entrée et un vecteur de sortie désirée. Le réseau s'engagera à ce moment à calculer sa propre sortie par une propagation avant (forward) des calculs. Une fois la sortie calculée, celle-ci présente forcement une erreur par rapport à celle désirée. Cette erreur est utilisée pour l'adaptation des pondérations en faisant une propagation arrière (backward), de la couche de sortie vers la couche d'entrée, dans le but de minimiser cette erreur.

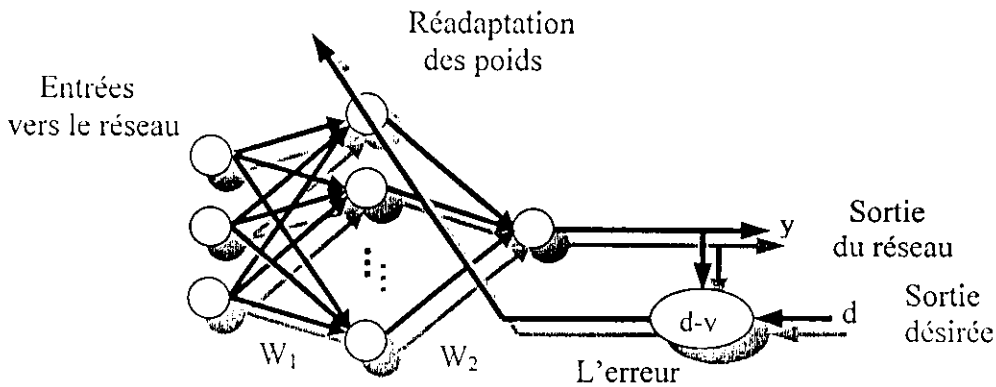


Figure II.6
principe de l'entraînement du réseau par rétropropagation de l'erreur

Remarques

- L'expérience a montré qu'il est, généralement, préférable d'initialiser les poids à des valeurs comprises entre 0 et 1 pour éviter de déstabiliser le réseau dés le départ.

- Le pas du gradient est choisi entre 0 et 1, et on le diminuera au fur et à mesure pour atteindre une valeur fixe.

-Le choix des exemples d'apprentissage est très important, un choix judicieux de ces derniers pourra faciliter la généralisation sur des exemples non appris. Ceci a poussé les chercheurs à faire l'étude statistique des données d'entraînement pour trier celles qui sont les plus représentatives du phénomène qu'on souhaite avoir.

II.4.7.1.1 Algorithme de rétropropagation

Considérons un réseau à n neurones en entrée, m en sortie et p couches cachées, chaque couche (numéro k) possède n_k neurones, on note par η le pas du gradient, on résume cette algorithme dans les cinq étapes suivantes :

1 . Initialiser tous les poids W_{ij} aléatoirement.

2 . Présenter au réseau une forme $(x_1, ..., x_n)$ en entrée, et sa sortie désirée $(y_1, ..., y_n)$.

3 . Calculer les sorties des neurones des couches cachées et de la couche de sortie par les relations suivantes :

Première couche cachée :

$$O_j^1(k)=f\left(\sum_{i=1}^{n} W_{ji}(k)x_i(k)\right) \tag{II.12}$$

deuxième couche cachée :

$$O_j^2(k)=f\left(\sum_{i=1}^{n1} W_{ji}(k)O_i^1(k)\right) \tag{II.13}$$

etc...

couche de sortie :

$$S_j(k)=f\left(\sum_{i=1}^{np} W_{ij}(k)O_i^p(k)\right) \tag{II.14}$$

4 . Modifier les poids des connexions récursivement par :

$$W_{ji}(k+1)=W_{ji}(k)-\eta\, d_j(k)O_i(k) \tag{II.15}$$

Avec :

$$d_j(k)=(S_i(k)-y_i(k))\,f'\left(\sum_{i=1}^{np} W_{ji}(k)O_i^p(k)\right) \tag{II.16}$$

pour la couche de sortie,

$$d_j(k)=\sum_{i=1}^{nk+1} d_i(k)W_{ji}(k)\,f'\left(\sum_{i=1}^{nk} W_{ji}(k)O_i^{nk}(k)\right) \tag{II.17}$$

pour la couche cachée numéro k.

5 . Refaire les étapes 2 à 4 jusqu'à stabilisation du réseau.

II.4.7.1.2 Difficultés et limites de l'algorithme

Des questions restent posées quant à l'utilisation de cet algorithme, à savoir :

- Combien faut-il utiliser de couches, et combien doit-il y avoir de neurones par couche ? Les réponses ne sont données que par l'expérience, et il n'existe aucune règle théorique précise qui puisse fixer le nombre de neurones ou de couches dans le réseau.

-le problème de minimisation n'est pas facile à résoudre. En effet, la surface d'erreur peut présenter des caractéristiques peu satisfaisantes, telles que des minimas locaux qui empêchent la convergence vers le minimum global, ainsi que des plateaux où les pentes sont très faibles.

- Le pas du gradient peut être difficile à choisir. S'il est faible, la convergence peut être très lente. S'il est élevé, l'erreur risque d'osciller sans autant converger.

- De plus, l'algorithme n'a aucune preuve théorique de sa convergence.

Malgré les difficultés qu'on vient de citer, l'algorithme de rétropropagation s'est révélé assez performant dans la résolution de plusieurs problèmes.

II.4.7.2 Etude du taux d'apprentissage

La version théorique de la rétropropagation réclame une variation infinitésimale des valeurs des poids à chaque itération [Rum86]. Cette variation est contrôlée par le taux d'apprentissage η. Conventionnellement ce facteur doit être petit assurant la convergence mais d'une manière très lente. Un facteur plus grand génère des changement plus important dans la valeur des poids synaptiques, pouvant accélérer ainsi l'apprentissage.
Malheureusement, ce n'est, le plus souvent, pas le cas. En effet avec une valeur importante de ce facteur la surface d'erreur est parcourue d'une manière « téméraire », engendrant ainsi des oscillations et une instabilité dans la recherche du minimum, ce qui compromet la convergence. De ce fait le choix de la valeur de ce paramètre est d'une importance capitale dans cet algorithme d'apprentissage.

Taux d'apprentissage adaptatif

Un choix d'un taux d'apprentissage décroissant avec l'évolution de l'apprentissage est, dans la plupart des cas, bénéfique. Une décroissance à raison de $\frac{1}{n}$ (n est la dimension du vecteur d'entrée) ou plus peut être retenue.

Le mieux est de choisir un taux d'apprentissage adaptatif. Actuellement il existe des méthodes qui choisissent un taux d'apprentissage optimal à chaque itération. Parmi ces méthodes, on peut citer la méthode de *Nash* ou celle de *Wolfe* qui consistent à vérifier des conditions sur la dérivée du critère à minimiser selon lesquelles le taux est soit augmenté soit réduit [Riv95].

Kung a cité une méthode d'adaptation qui consiste à adapter le taux d'apprentissage au même titre que les poids. Ainsi, à chaque étape une valeur optimal de ce paramètre qui peut nous rapprocher le plus des valeurs désirées en sortie est calculée. Nous expliquons ci dessous cette méthode.

Soit $\hat{Y}(X_p, W_p)$, le vecteur de sortie du réseau pour un vecteur d'entrée X_p.
En développant la sortie du réseau en série de *Taylor* de premier ordre autour des poids calculés par l'algorithme de rétropropagation à cette étape, on a :

$$\hat{Y}(X_p, W_{(p+1)}) \approx \hat{Y}(X_p, W_p) + \left[\frac{\partial \hat{Y}(X_p, W_p)}{\partial W}\right]^T \Delta W_p \qquad \text{(II.18)}$$

En égalisant cette sortie à la sortie désirée, on obtient aisément une valeur du taux pour cette étape :

$$\eta_p = \frac{1}{\left\| \frac{\partial \hat{Y}(X_p, W_p)}{\partial W} \right\|^2} \qquad \text{(II.19)}$$

Dans le présent travail, afin d'augmenter la rapidité de convergence, nous avons utilisé une autre technique plus simple pour la modification des poids du taux d'apprentissage. Cette technique consiste à mettre en œuvre un algorithme qui a pour rôle de contrôler l'erreur d'entraînement à chaque étape, ainsi pour un *Block Learning*, le principe de cette méthode est le suivant :

A chaque fois que l'erreur actuelle dépasse la précédente par un certain seuil précédemment fixé – ce qui veut dire une augmentation des oscillation d'où risque de divergence-, on rejette les poids générées, on revient au point précédent et on diminue le taux d'apprentissage. Si par contre l'erreur diminue, les poids générés sont retenus et le taux d'apprentissage est augmenté. De cette manière, on essaye à chaque étape d'avancer le plus rapidement possible vers l'optimum tout en évitant la divergence de l'algorithme. Cette méthode donne une convergence très rapide avec une bonne précision d'entraînement le principe général de cette méthode peut se résumé à travers le schéma suivant :

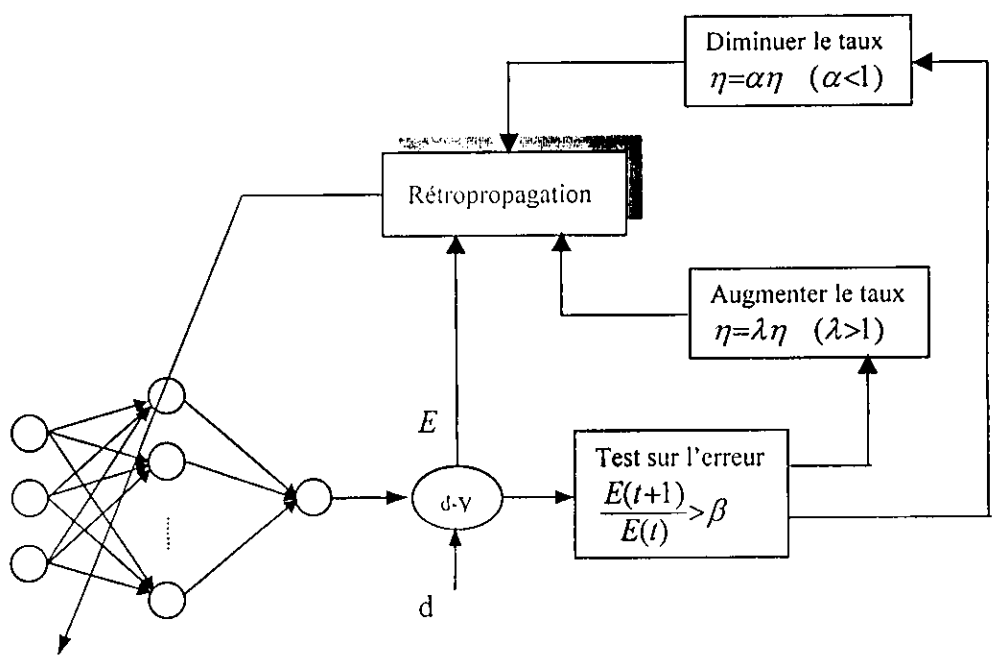


Figure II.7
Schéma de principe d'adaptation du taux d'apprentissage

II.4.7.3 Variantes de la Rétropropagation

Ce qui est recherché dans un algorithme d'apprentissage, c'est la stabilité du processus et la rapidité de convergence avec une bonne précision. Depuis l'apparition de la rétropropagation, quelques variantes visant à améliorer les performances de cet algorithme ont été mises en œuvre. Nous présentons ci-dessous les plus importantes.

a. Fast Backpropagation

Il est connu que la réduction de la fonction erreur est généralement plus importante pendant les premières itérations de l'apprentissage, où l'erreur estimée par la fonction objective est importante. Lorsque cette erreur décroît, la convergence de l'algorithme devient de plus en plus lente.

Afin de réactiver l'entraînement, *N.B.Karayannis* et *A.N.Venetsanopoulos* proposent d'appliquer l'algorithme de rétropropagation avec une fonction objective dite généralisée, cette méthode permet de changer le critère à minimiser au fur est à mesure que la convergence devient lente. Le critère proposé est décrit par :

$$G(\lambda)=\lambda E_1+(1-\lambda)E_2 \qquad (II.20)$$
$$=\lambda\sum_{k=1}^{M}\sum_{i=1}^{n_L}\frac{1}{2}(e_i^k)^2+(1-\lambda)\sum_{k=1}^{M}\sum_{i=1}^{n_L}\Theta(e_i^k)$$

Ou λ est un paramètre tel que $\lambda \in [0,1]$, et Θ une fonction appliquée sur l'erreur. Celle-ci doit être définit positive, continue et différentiable et doit approximer asymptotiquement l'erreur absolue e_i^k.

Cette fonction est choisie pour u réseau LBF comme étant :

$$\Theta(d_i^k-y_i^k)=d_i^k(d_i^k-y_i^k) \qquad (II.21)$$

Ou y_i^k représente la sortie du i ème neurone de la couche de sortie du réseau pour le k ème exemple en entrée.

Ainsi de la même manière que la rétropropagation classique on a à chaque itération pour un *Data learning :*

$$w_{ij}^{k+1}=w_{ij}^k-\eta_k\frac{\partial G(\lambda)}{\partial w_{ij}^k} \qquad (II.22)$$

Au début de l'apprentissage, on fixe le paramètre $\lambda=1$, ce qui revient à minimiser le premier terme seulement de l'équation précédente. Sa valeur doit, par la suite, diminuer progressivement avec la diminution de l'erreur et l'augmentation du nombre d'itérations jusqu'à atteindre zéro à la fin de l'apprentissage. Ceci, afin de permettre de prendre de l'importance au détriment du premier, dans l'expression du critère définie par l'équation II.20, et ainsi réactiver l'apprentissage. Une formule visant à faire diminuer λ de 1 à 0 avec la diminution de l'erreur peut être utilisée.

$$\lambda=\lambda(E)=\exp\left(\frac{-\mu}{E^n}\right) \qquad (II.23)$$

Avec μ un nombre positif réel et n un entier positive.

b. Rétropropagation avec Momentum

C'est une méthode très importante et efficace. Actuellement, la Rétropropagation est pratiquement toujours, utilisé avec le Momentum. Nous avons mentionné précédemment, qu'une valeur importante du taux d'apprentissage était nécessaire pour accélérer la convergence. Le problème d'oscillation nous empêche cependant de le faire augmenter. Une solution proposé par *D.E.Rumelhart* , qui consiste à utiliser les changements précédents des pour la réadaptation des poids actuels.

L'équation d'adaptation devient donc :

$$w_{ij}^l(t+1)=w_{ij}^l(t)+\eta\Delta w_{ij}^l(t)+\alpha\Delta w_{ij}^l(t-1) \qquad \text{(II.24)}$$

Le terme ajouté est appelé Momentum (élan, quantité de mouvement), en analogie avec la Mécanique Classique, ou un objet en mouvement garde l'élan acquis grâce à la quantité de mouvement qui lui a été communiquée précédemment pour accélérer son mouvement.

Le paramètre α est utilisé pour pondérer l'effet de ce terme. Sa valeur est généralement prise entre 0.8 et 0.9. Des recherches plus poussées ont montré que l'adaptation de sa valeur, comme c'est le cas pour le taux d'apprentissage η, donne des résultats très intéressant. *D.W.Patterson* a montré qu'en commençant l'apprentissage avec une valeur faible (environ 0.5) et en la faisant augmenter au fur et à mesure, l'entraînement s'en trouve sensiblement accélérer.

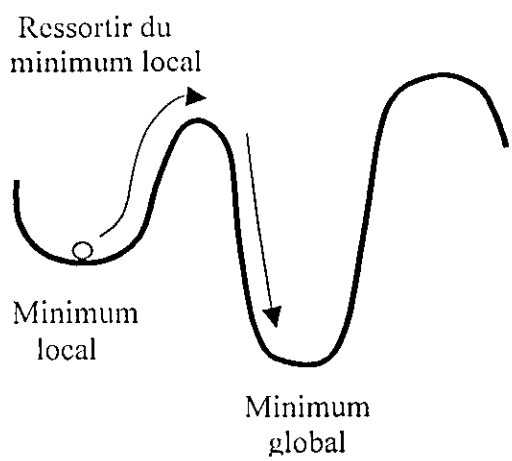


Figure II.8
Effet du Momentum pour échapper d'un minimum local

L'utilisation de cette méthode permet de faire sortir les poids des minimums locaux, afin de chercher d'autre optimums (figure II.8), ce qui donne beaucoup de chances d'aboutir à un minimum global.

Le Momentum tend d'autre part, à éviter les grand ''sauts'' pendant l'apprentissage, générés par le changement de la pente dans la surface de d'erreur. Celui-ci encourage ainsi le mouvement dans la même direction, pendant les étapes successive. Ce paramètre fait donc un lissage (filtrage passa bas) sur le parcours de la surface d'erreur.
Cette technique génère néanmoins des oscillations à la recherche d'un minimum global.

Notre utilisation de cette méthode nous a permis de constater qu'il est intéressent de l'appliquer avec un taux d'apprentissage dynamique.

II.4.7.3 Méthode d'optimisation du second ordre

Les méthodes d'optimisation de second ordre sont utilisées pour améliorer le choix de la direction à entreprendre dans l'espace des poids à la recherche du minimum. Elles sont basées sur l'utilisation de la dérivée seconde de la fonction *objectif* par rapport aux poids. L'utilisation de la second dérivée nous fournit, en effet, plus d'information sur cette fonction. Ainsi d'une manière directe ou indirecte, la matrice Hessien H doit être utilisée, en effet, c'est le Hessien qui nous informe sur la forme de la surface d'erreur dans l'espace des poids. Les algorithmes développés sont basés sur la méthode du Gradient du second ordre.
La méthode du Gradient du second ordre est une extension de la technique de recherche de minimum de newton, qui consiste à minimiser la fonction d'énergie qui est de la forme :

$$E(k+1)=E(k)+E(k)\Delta W(k)+\frac{1}{2}\Delta W(k)^{t}E(k)\Delta W(k) \qquad \text{(II.25)}$$

La surface de l'erreur est considérée comme étant quadratique, pour chercher le minimum, on doit choisir ΔW tel que :

$$\frac{\partial E(k+1)}{\partial \Delta W}=0 \qquad \text{(II.26)}$$

D'où l'on obtient :

$$\Delta W(k)=-\eta_k\left[E(k)\right]^{-1}E(k) \qquad \text{(II.27)}$$

ou η_k représente le taux d'apprentissage qui est positive et de valeur petite.

a. Méthode du Gradient Conjugué

Afin de minimiser l'équation II.25, on se donne une direction d vers laquelle la recherche des réadaptations des poids est orientée afin d'atteindre le minimum.

$$\Delta W(k)=\eta_k d_k \qquad \text{(II.28)}$$

Chaque direction est une combinaison linéaire du gradient actuel et la direction précédente.

$$d_k=-\frac{\partial E(k)}{\partial W(k)}+\beta_{k-1}d_{k-1} \qquad \text{(II.29)}$$

Dans un point représentant un minimum dans la surface d'erreur, le gradient de l'erreur est perpendiculaire à la direction de recherche. C'est la condition d'orthogonalité [Jer93].
D'où l'on a :

$$d_k^{t}\left.\frac{\partial E}{\partial W}\right|_{w+\eta_k d_k}=0 \qquad \text{(II.30)}$$

En utilisant l'équation II.30 dans II.25 et en l'appliquant sur II.28, on tire la valeur de η_k qui représente le valeur optimale avec laquelle les poids doivent être déplacés dans cette direction à chaque étape :

$$\eta_k = \frac{d_k^T \frac{\partial E(k)}{\partial W}}{d_k^T H d_k} \qquad \text{(II.31)}$$

ou H représente le Hessien de l'erreur à l'instant k.

De plus, il est même possible d'utiliser les relation 28 et 29 pour calculer la dérivée de l'équation II.25 par rapport au paramètre β_{k-1}, afin de tirer sa valeur optimale pour l'utiliser dans 29 [Kun93].

Si au bout de plusieurs itérations, le minimum voulu n'est pas obtenu, cette méthode permet de redémarrer à partir du dernier point, en réinitialisant les paramètres β_{k-1} à zéro, et entreprendre de nouvelles recherches. Cette procédure, appelée *Restart*, permet de changer rapidement de directions de recherche, afin d'obtenir l'optimum désiré [Jer93].

b. Méthode de Leavenberg Marquard

La méthode de Leavenberg-Marquard est l'une des plus utilisées. Cette méthode consiste à considérer la surface d'erreur quadratique, en se basant sur la fonction d'énergie II.25.

En posant pour chaque neurone i d'une couche l le vecteur de poids lui parvenant des neurones de la couche qui le précède qui incluse en plus le biais.

$$W_i^l = \left[w_{i1}^l \ , \ w_{i2}^l \ , \ \dots \ , \ w_{im-1}^l \ , \ \theta_i^l \right] \qquad \text{(II.32)}$$

Soit F_i^l les vecteur des dérivées de l'erreur à la sortie du réseau par rapport à ces vecteurs poids w_i^l pour $l=1,...,L$ et $i=1,...,n_l$.
A la présentation de P ème entrée, on a :

$$F_i^{l\cdot} = \sum_{p=1}^{M} \sum_{i=1}^{ml} (d_i^p - y_i^l) \left[-\frac{\partial y_i^{l\cdot}}{\partial W_i^l} \right] \qquad \text{(II.33)}$$

Ou M représente le nombre d'exemple d'entraînement, n_l le nombre de neurones dans la couche l, y_i^l la i ème sortie du réseau, et d_i^p sa sortie désirée pour la p ème entrée.

On définit F un seul grand vecteur rassemblant tous les vecteur F_i^l définit par l'équation II.32.
A partir de II.25 et II.26, afin de trouver l'optimum on obtient :

$$H \Delta W = -F \qquad \text{(II.34)}$$

Où H représente la matrice Hessien de la fonction erreur en sortie du réseau, l'équation II.33 représente un système d'équations linéaire dont la résolution peut être faite par la méthode de Gauss-Newton.

La méthode de Levenberg-Marquard remplace le calcul du Hessien par une approximation numérique donnée par :

$$H=FF^T+\lambda\Omega \qquad \text{(II.35)}$$

Ou λ est un coefficient positif. La matrice FF^T étant définie semi positive, la coefficient λ est utilisé afin de mieux conditionner la matrice H. Ainsi, la matrice Ω doit être d'un choix approprié.

Le choix d'une matrice diagonale dont les éléments sont égaux aux éléments diagonaux de la matrice FF^T donne de bons résultats. Le choix d'une matrice Identité diagonale peut aussi bien être fait [Kol 89]. Les valeurs que doit prendre le paramètre λ doivent être convenablement choisies. En effet, la recherche doit être orientée dans une région de l'espace Permettant de représenter adéquatement le problème non-linéaire par le modèle quadratique (II.25), afin de garantir la convergence de l'algorithme vers un minimum global. Il existe quelques techniques, qui ont été mises au point pour cela. La plus simple consiste à commencer avec une petite valeur et la faire augmenter jusqu'à ce que l'erreur diminue, la valeur maximale de ce paramètre doit être, cependant bornée.

II.4.7.4 Méthode d'optimisation aléatoire (ROM)

Cette méthode (Random Optimisation Method) est basée sur une technique de recherche aléatoire. Elle fut utilisée avec succès dans divers problèmes d'optimisation. Elle est performante quand la fonction est complexe et qu'on veut trouver son minimum global, sachant qu'elle présente des minimas locaux. La nécessité d'utiliser cette méthode survient lorsque la dimension du réseau devient grande et le temps de calcul, avec l'algorithme de rétropropagation, peut être problématique.

La méthode d'optimisation aléatoire converge vers le minimum global d'une fonction avec une probabilité égale à 1 dans un ensemble compact.

L'idée de cet algorithme, est d'entacher les poids du réseau d'une séquence de bruit blanc et de calculer la sortie du réseau avec ces nouvelles pondérations. Si l'erreur entre celle-ci et la réponse désirée est inférieure à l'erreur précédente, on garde cette séquence de poids. Sinon on garde la séquence précédente. On refait l'opération jusqu'à obtention de l'erreur voulue.

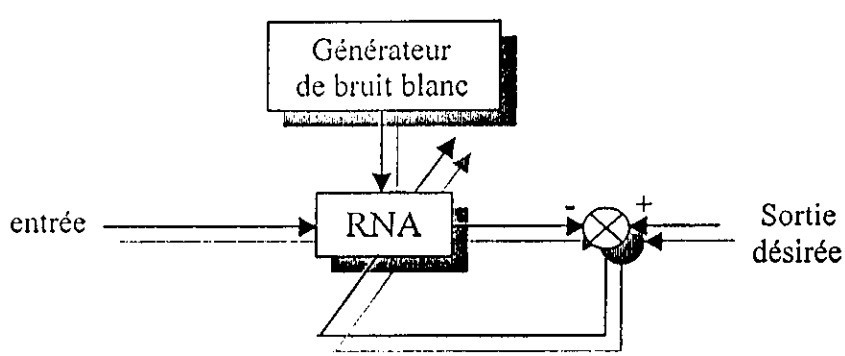


Figure II.9
adaptation des poids par ROM

L'algorithme d'optimisation aléatoire se résume dans les suivantes :

Soit :

$f(x)$ la fonction à minimiser.
$g(k)$ le vecteur de bruit Gaussien.
$b(k)$ la moyenne du bruit $g(k)$ à l'instant k.

1. Choisire un point initial $x(0)$, $b(0)=0$ pour $k=0$.

2. Générer un vecteur gaussien $g(k)$.

3.1. Si : $f(x(k)+g(k))<f(x(k))$ (II.36)

Alors on pose : $x(k+1)=x(k)+g(k)$ (II.37)

et : $b(k+1)=0.4g(k)+0.2b(k)$ (II.38)

3.2. Si : $f(x(k)+g(k))\geq f(x(k))$ (II.39)

et : $f(x(k)-g(k))<f(x(k))$ (II.40)

on pose dans ce cas on pose : $x(k+1)=x(k)-g(k)$ (II.41)

et : $b(k+1)=b(k)-0.4g(k)$ (II.42)

Sinon, on pose : $x(k+1)=x(k)$ (II.43)

et : $b(k+1)=0.5\ b(k)$ (II.44)

4. Si le minimum est satisfaisant, on stoppe les calculs, sinon il faut poser $k=k+1$ et aller en **2**.

Remarques

Dans le cas de l'apprentissage du réseau neuronal, la fonction à minimiser est, bien entendu, l'erreur quadratique (la somme des carrés des erreurs commise sur chaque exemple d'apprentissage), qui est fonction des pondérations.

Pour les besoins théoriques de l'algorithme, le vecteur poids doit appartenir à un ensemble compact. Par exemple, on prendra W comme étant un vecteur dont les composantes sont inférieures à 100. Ce n'est que dans ce cas qu'on sera capable de trouver le minimum global de la fonction d'erreur.

Durant la généralisation du bruit blanc, le choix de la variance est capital pour améliorer-la vitesse de convergence. Généralement, on la fixe à une valeur entre 0 et 1, pour des raisons purement expérimentales, et on la diminue au fur et à mesure qu'une stagnation de l'erreur est remarquée durant l'apprentissage.

II.4.7.5 Combinaison des algorithmes BP et ROM

Des travaux de comparaison des algorithmes BP et ROM ont été effectués dans le but de déterminer le meilleur outil pour l'entraînement des réseaux multicouches. Les résultats découlant de cette comparaison ont été avantageux à la méthode d'optimisation aléatoire du point de vue rapidité (nombre d'itérations relativement inférieur), et présentant une erreur finale inférieure à celle obtenue par l'algorithme BP.

Toutefois, ces résultats sont spécifiques à certaines applications, et la généralisation n'est pas pour bientôt. C'est pour cela qu'on continuera à utiliser les deux algorithmes pour l'entraînement des réseaux neuronaux.

Seulement, on peut mentionner une autre manière de faire concernant l'apprentissage. Elle concerne la combinaison des deux algorithmes. En effet, la qualité de l'erreur peut être améliorée et de loin, si l'on procède de la manière suivante :

- Débuter l'apprentissage avec l'algorithme BP pendant certain nombre d'itérations (généralement, jusqu'à ce que la convergence devienne lente), bénéficiant ainsi d'un bon conditionnement des poids.

- Terminer l'apprentissage à l'aide de l'algorithme ROM, avec pour poids initiaux ceux obtenus à la fin de l'apprentissage par BP, bénéficiant ainsi de la convergence vers une erreur finale encore meilleure.

II.4.7.6 Optimisation des réseaux neuronaux

On dispose d'un réseau neuronal, d'un échantillon de données empiriques pour entraîner le réseau et d'un algorithme d'apprentissage approprié. Mais pour obtenir la meilleure généralisation possible, quelle doit être la taille du réseau ?

L'approximation de fonctions complexes nécessite un réseau d'une grande taille avec des unités cachées à fonctions d'activation nonlinéaires. Cependant, outre les inconvénients qu'il présente, simulation lente et trop d'exemple d'apprentissage, un réseau sur-dimensionné risque de conduire à une mauvaise généralisation.

L'amélioration des capacités de généralisation a été essentiellement expérimentale. Aucune méthode ne permet de déterminer précisément la structure du réseau en fonction des exemples à apprendre pour fournir une généralisation optimale.

II.4.8 Réseaux à Fonction de Base Radiale RBF

II.4.8.1 Principe de base

Les réseaux RBF représente des réseaux à une couche cachée qui peuvent être utilisés aussi bien pour la classification que pour l'approximation de fonctions.
A travers une combinaison linéaire de fonction non-linéaire de base radiale, le fonctionnement de ces réseaux repose sur le principe des estimateurs à noyau, et le généralise du monovariable vers le multivariable.
Un estimateur à noyau considère des fonctions de R^+ vers R qui sont de la forme :

$$\Phi_i(\chi)=\Phi_i\left(\|\chi-\xi_i\|\right) \tag{II.45}$$

L'idée principale a été introduite par *M.J.D.Powell.* Elle est basée sur l'interpolation, toute fonction $f(\chi)$ d'une variable $\chi\in R^+$ peut être approchée par une interpolation composée par la somme de p fonction noyaux de forme fixée $\Phi(.)$.

$$f(\chi)=\sum_{i=1}^{p}\lambda_i\Phi\left(\|\chi-\xi_i\|\right) \tag{II.46}$$

ou ξ_i représentent les nœuds de l'interpolation pour $i=1,...,n$.

λ_i sont les paramètres que l'on déterminera à partir des exemples connus χ_i , γ_i en résolvant le système :

$$f(\chi_i)=\gamma_i \qquad i=1,...,n \tag{II.47}$$

et $\Phi(.)$ est une fonction assurant la continuité aux nœuds et la derivabilité d'ordre supérieur en ces points. Dans le cas général d'une interpolation, une fonction polynomiale peut, par exemple, être choisie [Khe94].

II.4.8.2 Architecture et fonctionnement des réseaux RBF

Broomhead & Lowe puis *al* ont conçu un réseau de neurones à une seule couche cachée dont le fonctionnement est basé sur l'idée des approximateurs à noyau introduit ci-dessus. Ainsi à partir de l'équation II.45 et en la considérant dans le cas multivariable. La sortie que doit délivrer ce réseau est de la forme (figure II.10).

$$f_k(\chi)=\sum_{i=1}^{n}\lambda_{ki}\Phi\left(\|\chi-c_i\|\right) \qquad k=1,...,n\ ;\ i=1,...,m \tag{II.48}$$

Où c_i sont des valeurs que nous attribuons à chaque neurone de la couche cachée appelés la fonction noyau Φ qui doit toujours assurer la continuité et la dérivabilité aux points de jonction est radialement symétrique. Ainsi, la contribution de cette dernière change en fonction de la position de l'entrée par rapport aux centres. Par ailleurs, elle doit produire des réponses localisées, ses valeurs ne sont significatives que dans un certain intervalle de l'espace des entrées[Ren95].

Parmi ces fonctions, on peut trouver les formes suivantes :

Forme Cubique	$\Phi(r)=r^3$	
Forme Multiquadratique	$\Phi(r)=(r^2+k^2)^{1/2}$	(II.49)
Forme Logarithmique décalée	$\Phi(r)=\log\left(r(r^2+k^2)\right)$	
Forme Gaussienne	$\Phi(r)=\exp\left(\frac{-r^2}{\beta^2}\right)$	

Où r représente ici un réel quelconque, remplaçant le discriminant des neurones.
Parmi ces formes, la fonction Gaussienne est la plus utilisée en raison des ses caractéristiques.

L'architecture d'un réseau RBF est représenté par la figure suivante :

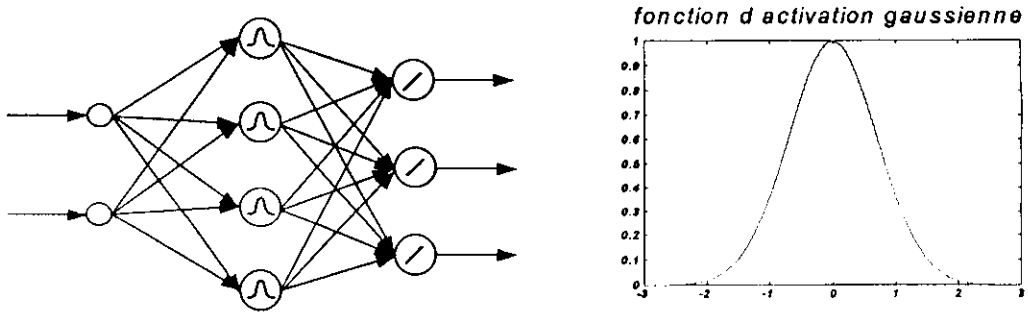


Figure II.10
Exemple de réseau à base radiale (activation gaussienne)

Celui-ci est composé de trois couches (figure II.10). la première couche reçoit les entrées injectées au réseau. La couche cachée est constituée de neurones. Dont le discriminant est à fonction radiale de base. Chacun de ses neurones est doté d'un vecteur C_i appelé centre. Cette couche effectue une classification des exemples d'entrées. En effet chaque entée du est rangée dans l'une des classes représentées par les neurones de la couche cachée. Les valeurs des centres doivent donc, représenter chacune le barycentre de la classe qui lui correspond. Ainsi plus le nombre d'exemples d'entraînement est important, plus cette couche est chargée de neurones.

Lorsqu'un vecteur X est présenté à l'entrée du réseau, chaque neurone calcule, à travers son discriminant, la distance euclidienne de cet exemple par rapport au centre de la classe qu'il représente.

$$d_i=\|X-c_i\| \tag{II.50}$$

la sortie de chaque neurone de cette couche est calculée en appliquant la fonction noyau :

$$s_i(X)=\Phi(\|X-c_i\|) \qquad 1\le i\le m \tag{II.51}$$

Avec m, le nombre de neurones cachés.

Cette fonction est généralement Gaussienne, d'où l'on a :

$$s_i(X)=\exp\left[-\frac{(X-c_i)^T(X-c_i)}{2\sigma_i^2}\right] \tag{II.52}$$

Où σ_i est un paramètre qui représente une mesure de la dispersion des données associées à chaque nœud ; il est souvent égal à la distance moyenne entre le centre de la classe et les exemples d'entraînement y faisant partie (Hus 95).

L'utilisation de la fonction Gaussienne dans cette couche rend ce réseau très puissant. En utilisant cette fonction à caractéristique locale, chaque neurone ne réagit d'une manière significative qu'à une partie restreinte de l'espace d'entrée. En partageant ainsi l'espace d'entrée, les aptitudes de ce réseau en approximation se trouvent significativement améliorer par rapport au réseau LBF dont les sorties sigmoïdes ne possèdent pas cette propriété de calcul local.

La classification étant effectuée, la couche de sortie, qui est constituée de neurones linéaires effectue l'approximation définie par la somme pondérée suivante :

$$y_k(X)=\sum_{i=1}^{m} w_{ki}\Phi\left(\|X-c_i\|\right) \qquad 1\le k\le n \tag{II.53}$$

avec n le nombre de neurones de sorties

cette relation peut être écrite sous une forme plus détailler et générale pour un réseau comportant n unités dans la couche cachée :

$$y=\frac{\sum_{j=1}^{n} w_j \rho_j(u)}{R(u)} \tag{II.54}$$

$R(u)$ est un terme de normalisation.

Avec : $R(u)=0$ pour un réseau non normalisé.

$R(u)=\sum_{i=1}^{n}\rho_i(u)$ pour un réseau normalisé.

Et :

$$\rho_j(u)=\exp\left(-\frac{1}{2}\sum_{k=1}^{m}\frac{(u_k-c_{kj})^2}{\sigma_{kj}^2}\right), \quad (m \text{ étant le nombre d'entrées}) \tag{II.55}$$

II.4.8.3 Apprentissage des Réseaux RBF

L'entraînement d'un réseau RBF comprend deux étapes :

- Apprentissage de la couche cachée : sélection des centres.
- Apprentissage de la couche de sortie : détermination des poids synaptiques de cette couche.

Dans le cas idéal, un centre doit être placé pour chaque exemple d'entraînement. Ainsi pour la détermination des valeurs de ces centres, on aura une matrice carrée constituant autant d'équations que d'inconnus.
Dans le cas réel, avec un nombre important d'exemples, ceci est impossible. Les centres doivent donc être choisis de sorte à effectuer un échantillonnage représentatif de l'espace des exemples d'entraînement
Quant aux poids synaptiques', leur détermination constitue la partie la plus simple de l'apprentissage. En effet la sortie étant linéaire, n'importe quel algorithme d'optimisation linéaire (LMS, par exemple) peut être aisément utilisé.

Dans ces méthodes d'entraînement, ils existent trois approches différentes qui peuvent être utilisées.

- Apprentissage non supervisé basé sur un algorithme de regroupement.
- Apprentissage supervisé, utilisant les algorithmes d'entraînement basés sur l'optimisation.
- Apprentissage supervisé de regroupement .*(Clustering)* hiérarchique

1. Méthode de Centrage Adaptatif (Adaptive Centering Method)

Comme pour les paramètres de pondération w_{ji} , à toute valeur attribuée aux centres c_i correspondra une certaine erreur en sortie. Il est aisé de remarquer, que cette erreur est aussi bien dérivable par rapport aux centres qu'aux poids synaptiques.

Ainsi tous ces paramètres peuvent, donc, être ajustés en utilisant la méthode de descente du gradient jusqu'à obtention d'un minimum satisfaisant.
nous présentons ci dessous les étapes de cette technique d'apprentissage pour un réseau normalisé, qui est le cas le plus général :

soit $J=\frac{1}{2}(y-y_d)^2$ le critère à minimisé.

Posons $R(u)=\sum_{i=1}^{n}\rho_i(u)$ un facteur de normalisation.

1. Initialiser les centres, les paramètres de la sortie Gaussienne de chaque neurone de la couche cachée, et les poids synaptiques.

2. choisir un taux d'entraînement variable ou fixe.

3. Présentation successive de P exemples d'entraînement.
Calcul de l'erreur quadratique en sorties entre la réponse désirée et celle fournie par le réseau.

4. Réajuster les poids synaptiques, les centres et éventuellement les paramètres des fonctions noyau (les Gaussiennes) par la règle :

$$\begin{aligned}\Delta w_j &= -\eta_w \frac{\partial J}{\partial w_j}\\ \Delta c_{jk} &= -\eta_c \frac{\partial J}{\partial c_{jk}}\\ \Delta \sigma_{jk} &= -\eta_\sigma \frac{\partial J}{\partial \sigma_{jk}}\end{aligned} \qquad \text{(II.56)}$$

avec :

$$\frac{\partial J}{\partial c_{jk}}=\frac{\partial J}{\partial \rho_j}\frac{\partial \rho_j}{\partial c_{jk}} \qquad \text{(II.57)}$$

$$\frac{\partial J}{\partial \sigma_{jk}}=\frac{\partial J}{\partial \rho_j}\frac{\partial \rho_j}{\partial \sigma_{jk}} \qquad \text{(II.58)}$$

$$\frac{\partial J}{\partial \rho_j}=(y-y_d)\left[\frac{w_j R(u)-\sum_{i=1}^{n}\rho_j(u)w_i}{R(u)^2}\right] \qquad \text{(II.59)}$$

$$\frac{\partial \rho_j}{\partial c_{jk}}=\rho_j.\frac{u_k-c_{jk}}{\sigma_{jk}^2} \qquad \text{(II.60)}$$

$$\frac{\partial \rho_j}{\partial \sigma_{jk}}=\rho_j.\frac{(u_k-c_{jk})^2}{\sigma_{jk}^3} \qquad \text{(II.61)}$$

Notons qu'il y a maintenant *(1+2m)n* paramètres ajustables (*m* étant le nombre d'entrées).
Cet Algorithme simplifie l'apprentissage, dans le sens où les poids et les centres sont réadaptés directement ensemble, et ceci en dépit du calcul des dérivées partielles de l'erreur par rapport à chaque centre. Ce calcul ne pose, cependant, pas problème. En effet, puisque le réseau ne contient qu'une seule couche cachée, les dérivées se calculent d'une manière automatique, contrairement au réseaux LBF multicouches. Un développement de cette méthode ainsi que les expressions détaillées de toutes les dérivées peuvent être trouvées sur (S.Khemaïssia & A.S.Moris, 1995) (Khe95).

Une autre technique, ne nécessitant pas le calcul des dérivées par rapport aux centres, peut être utilisée, qui repose sur le partage de l'espace des exemples en sous espaces, représentant chacun une classe différente.
Nous présentons ci dessous cette méthode.

2. Méthode Basée sur l'Algorithme de Regroupement (Clustering Algorithm)

Cet algorithme comprend un apprentissage non supervisé de classification (Hér94). Basé sur l'approche du *Nearest Neighborhood.* Les étapes de l'algorithmes d' apprentissage du RBF dans ce cas, sont les suivantes :

1. Initialiser les centres $C_i(0)$, avec des valeurs aléatoires et choisir un taux d'apprentissage initial.
2. Calculer la distance euclidienne de l'exemple X^p par rapport à chaque classe.

$$dist(p)=\left\|X^p-C_i(p-1)\right\| \tag{II.62}$$

3. Noter l'argument *k* du centre pour lequel la distance est minimale :

$$k=arg[min(dist_i(p))] \tag{II.63}$$

Tel que *arg*[.] identifie le rang du neurone *(i)* dans la couche cachée.
4. Réadapter les centres :

$$C_i(p)=C_i(p\text{-}1) \qquad \text{pour } 1\le i\le m \text{ et } i\neq k \tag{II.64}$$

$$C_i(p)=C_i(p\text{-}1)+\alpha(p)\left[X^p-C_i(p-1)\right] \qquad \text{pour } i=k \tag{II.65}$$

5. Réduire le taux d'apprentissage.

$$\alpha(p)=\frac{\alpha(p-1)}{1+\text{int}\left[\frac{p}{M}\right]^{1/2}} \tag{II.66}$$

(*int*[.] : Partie entière). Avec *M* le nombre d'exemples que contient la base d'apprentissage.

6. Refaire les étapes de **1.** à **5.** jusqu'à ce que chaque exemple soit classé et que les classes ne changent plus.
Pour les poids synaptiques, la méthode des moindres carrés peut aisément être utilisée après la détermination des centres.

Contrairement au premier, cet algorithme détermine d'abord les centres, puis les poids synaptiques. L'apprentissage non supervisé utilisé pour la détermination des centres, peut limiter le nombre de neurones, suivant le nombre de classes auquel cet apprentissage a abouti. L'algorithme supervisé, par contre, nous oblige à fixer le nombre de neurones dés le début. Cette méthode nous permet donc, d'appliquer les techniques d'apprentissage compétitif, qui démarre avec une structure minimale, et crée de nouveaux neurones pour de nouvelles classes, dés que nécessaire.

Il existe une autre famille d'algorithmes d'entraînement plus rapides et plus économiques en nombre de neurones, conduisant vers des réseaux évolutifs. Ces techniques d'apprentissage supervisé basés sur la classification, sont plus récentes et sont appelée *Algorithmes de regroupement hiérarchique (Hus 95).*
Nous présentons ci dessus cette technique.

3. Algorithmes d'apprentissage supervisé de Regroupement Hiérarchique

Ces méthodes basées sur la classification des exemples d'entraînement, nécessitent la présence des couples d'entrées et leurs sorties désirées. elles permettent, toute en procédant avec un apprentissage supervisé, de rendre l'architecture du réseau évolutive; ce qui permit de limiter le nombre de neurones dans la couche cachée et d'accélérer l'apprentissage.

Il existe deux techniques différentes pour ce genre d'apprentissage. La première consiste à commencer l'apprentissage avec un seul neurone dans la couche cachée, et augmenter leur nombre par la suite, pendant qu'avec la seconde, l'apprentissage commence avec un nombre important de neurone, pour le faire diminuer pendant l'apprentissage.(Hus95)

La première technique constitue une extension des réseaux RBF vers des réseaux Gaussiens appelé GPFN *Gaussien Potentiel Fonction Neural Networks* . Ces réseaux sont entraînés par la méthode supervisée Hiérarchique (voir Annexe A.) développée par S.Lee et M.Kill appelée *HSOL : Hierrachical Self-organized Learning*(Lee91).

Dans cette méthode d'entraînement, on définit pour chaque neurone une grandeur H_i appelée contours d'accommodations, qui est similaire au paramètre de vigilance défini pour le *ART*. Celle ci est caractérisée par le rayon r_i. de la classe limitant le champ d'action de la sortie du neurone qui la représente.

$$H_i = \left\{ x / d(x, c_i, \sigma_i) \leq r_i^2 \right\} \qquad \text{(II.67)}$$

Ainsi, à chaque itération, quand l'erreur en sortie est supérieur à un certain seuil prédéfini, si un exemple est repéré à l'intérieur d'une hyper-sphère H_i d'un neurone caché, dont la sortie est de la même classe que l'une des sorties du réseau, les paramètres de ce neurone à savoir les centres C_i , les poids synaptiques et les paramètres de dispersion sont ajustés suivant la méthode du gradient. Si, par contre, aucune classe n'est repérée, un nouveau neurone, qui doit représenter ce nouvel exemple, est crée dans la couche cachée. Dans le cas où l'erreur est inférieure à la marge tolérée, les paramètres du réseau sont réadaptés et l'entraînement reprend pour d'autres exemples. (Lee88, Lee91).

La deuxième technique consiste à commencer avec un nombre important de neurones cachés, puis le faire diminuer au fur et à mesure que l'entraînement avance, en fusionnant les unes avec les autres les classes les plus corrélés entre elles. A chaque itération, le neurone représentant la classe la plus éloignée des autres dont la sortie est la plus corrélée avec le vecteur des sorties désirées est repéré. Ce neurone est retenu, son vecteur centre est égal à l'exemple d'apprentissage qu'il représente et ses poids reliés aux sorties du réseau sont déterminés par les méthodes d'optimisation linéaire. L'exemple d'apprentissage que celui ci représente est ôté de la base d'apprentissage et les exemples qui lui sont fortement corrèles,

dans la base d'entraînement sont progressivement rangés avec lui dans la même classe. A la création de chaque classe les poids du réseau sont recalculer et la précision en sortie est testée; Ainsi, les neurones cachés nécessaires sont recrutés, les uns après les autres, créant une classe par itération, jusqu'à satisfaction du critère en sortie.

De cette manière l'entraînement est plus rapide et peut nécessiter, au maximum, un nombre d'itération égale au nombre d'exemples d'entraînement. Dans ce cas échéant, le nombre de neurones cachés est égal au nombre d'exemples d'entraînement. (Mus92, Hus95J). Nous avons présenté dans l'annexe A., un algorithme d'apprentissage Hiérarchique qui répond au premier cas. C'est le *HSOL : Hierrachical Self-organized Learning.* destiné pour les GPFN *Gaussian Potential Fonction Neural Nebvorlk* développé par S.Lee et M.Kill .

II.4.8.4 Réseaux RBF et Approximation de Fonction

A la recherche d'une bonne approximation Giroso et Pogio ont introduit en 1990 la notion de meilleur approximative qu'ils définissent comme celui ayant la fonction la plus proche de celle à estimer (Hungo).
Sur la base de ce critère, ils en ont déduit que les LBF ne représentent pas de meilleurs approximateurs et que les RBF en revanche, ont la propriété de meilleur approximateurs.
Ainsi, comme le confirmeront plus loin nos applications, les RBF représentent une bonne alternative pour l'approximation de fonctions complexes.

Le théorème démontrant les aptitudes de ces réseaux, est basé sur la théorie de Weierstrass sur l'approximation de fonctions. Celui ci a été rapporté par Kung (Kun93), et stipule que toute fonction continue dans un sous espace peut être approchée par un réseau RBF.

Pour l'entraînement de ces réseaux, S. Broomhead et D. Lowe (Hun92) ont démontré que l'utilisation des techniques d'optimisation linéaire, garantit une solution globale.
Les réseaux RBF constituent des modèles de réseaux très efficaces notamment pour l'approximation de fonctions. L'utilisation de la fonction gaussienne permet de bénéficier de sa caractéristique locale pour faciliter l'apprentissage et améliorer l'approximation. Par ailleurs, la procédure d'entraînement, basée sur le fonctionnement de ce réseau, qui consiste en une classification suivie d'une optimisation rend l'apprentissage beaucoup moins difficiles et plus rapide que celui des réseaux LBF. Par ailleurs, ces réseaux sont toujours utilisés avec une seule couche cachée. Ceci libère l'utilisateur du choix du nombre de couches.

Néanmoins ces réseaux présentent quelques inconvénients. En effet, lorsque la dimension de l'entrée augmente ou le nombre d'exemples d'apprentissage est important, la couche cachée risque d'être surchargée en neurones compliquant ainsi les calculs et ralentissant l'apprentissage.

En outre, afin d'alléger l'apprentissage, il est possible de fixer les paramètres de dispersion dés le début de l'apprentissage à une valeur fixe commune pour tous les neurones. Dans ce cas, il faut savoir qu'une valeur très petite de ce paramètre nécessite beaucoup de neurones dans la couche cachée, ce qui peut diminuer les capacité de généralisation du réseau. Un paramètre de valeur très importante, par contre, peut entraîner un chevauchement entre les classes rendant ainsi l'apprentissage impossible.

II.4.9 Réseaux Dynamiques (Récurrents)

Les réseaux récurrents diffèrent par rapport aux réseaux statiques par le fait que leurs structures contiennent des feed-backs entre les neurones. En général, la sortie de chaque neurone peut être envoyée vers l'entrée de tous les autres neurones du réseau. Cette structure

intéressante peut être exploitée notamment par l'entraînement du réseau à suivre une séquence temporelle. Elle lui confère par ailleurs la possibilité d'approximation des fonctions très complexes. Le principal défaut de ce type de réseaux est la complexité de calcul et de l'entraînement.

Un réseau récurrent n'est pas défini par son architecture seulement, mais aussi par son algorithme d'apprentissage. L'algorithme d'apprentissage est défini suivant l'architecture, les simplifications adoptées et l'objectif à atteindre par ce réseau.

Les algorithmes d'apprentissage sont classés en deux groupes importants :

Le premier cas consiste à faire apprendre au réseau une séquence temporelle (une trajectoire), ce type d'apprentissage est ainsi appelé Trajectory Learning. Le second cas vise à exploiter les aptitudes de ce type de réseaux à pouvoir faire l'approximation de fonctions complexes.

C'est un simple apprentissage de valeurs fixes. ce type d'apprentissage est alors appelé Fixed Point Learning. Dans ce deuxième type d'apprentissage, ce qui nous intéresse est le régime permanent, Le régime transitoire par lequel le réseau pourrait passer pour atteindre ce point fixe doit disparaître.

Une analogie tout à fait intéressante de ces deux types d'apprentissage peut être faite avec deux cas de commande en automatique. Le premier avec le *Tracking* (la poursuite) le second avec la *régulation.*

Nous partageons les réseaux récurrents en deux types principales d'architectures différentes que la figure (II.11) résume. Dans ce qui suit, nous allons étudier ces types de réseaux, ainsi que leurs algorithmes d'apprentissage.

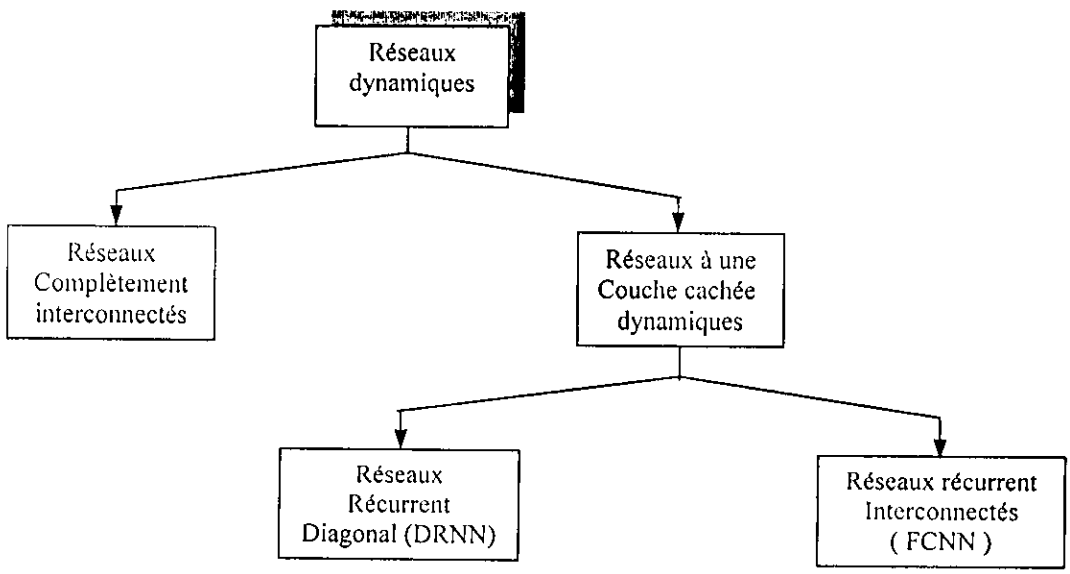


Figure II.11
Classification des réseaux récurrents suivant leurs architectures

II.4.9.1 Réseaux de neurones complètement connectés :

C'est le modèle le plus général de réseaux de neurones. Celui ci est, généralement, constitué par une couche d'entrée ordinaire et une couche de neurones entièrement

interconnectés, dont certains seulement sont considérés neurones de sorties[Ner92]. F.J Pineda a montré que ces réseaux peuvent être entraînés pour faire une correspondance entre entrées et sorties[Wil89]. Ce type de réseaux constitue des modèles très complexes en calcul et leur entraînement est, par conséquent, plus délicat. Ainsi, ils sont moins utilisés[Cha95]. Pour leurs entraînements, c'est la méthode de Backpropagation qui est utilisée, avec, néanmoins, la difficulté d'adapter cette méthode en raison de la complexité déjà signalée de ce type de réseaux.

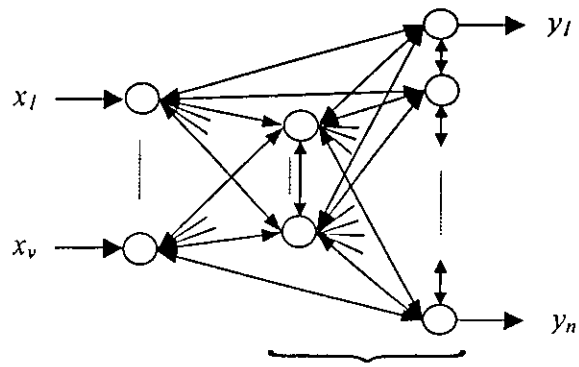


Figure II.12
Architecture d'un réseau dynamique
à structure complètement interconnecté

II.4.9.2 Récurrent backpropagation :

Applicable aux réseaux entièrement interconnectés, la Récurrent Backpropagation est basée sur la méthode d'apprentissage proposée par F.Almeida[Kar93]. Sur la figure II.12 nous avons représenté ce réseau par une couche d'entrée ordinaire et une couche dynamique entièrement connectée. Le comportement de la couche dynamique est régi par le système d'équations différentielles suivantes :

$$\tau \frac{dz_i}{dt} = -z_i + f(u_i) + I_i \qquad , \qquad i = 1 \ldots . n \tag{II.68}$$

n : nombre de neurones dynamiques.
Où : z_i représente la sortie du i ème neurone.
I_i représente un biais tel que l'on a :
$I_i = x_i$ si le neurone i est lié aux neurones d'entrée.
$I_i = 0$ sinon. (II.69)
Où :
x_i : représente l'entrée extérieure injectée au réseau.

Avec :

$$u_i = \sum_j w_{ij} z_i \tag{II.70}$$

f(.) : est une fonction de type sigmoïde.

L'équation (II.68) donne à l'équilibre :

$$z_i^* = f(u_i^*) + I_i \tag{II.71}$$

$$= f(\sum_j w_{ij} z_j) + I_i \tag{II.72}$$

Dans un apprentissage à point fixe, on est intéressé par l'état d'équilibre de ce réseau. L'erreur ne sera donc comptabilisée que sur cet état.
L'erreur à minimiser est donc définie par :

$$E = \frac{1}{2} \sum_k E_k^2 \tag{II.73}$$

tel que :

$$E_k = \begin{cases} d_k - z_k^* = d_k - y_k^* & k \in \Pi(\text{sortie}) \\ 0 & \text{sinon} \end{cases} \tag{II.74}$$

où :
y_k : représente la sortie du k ème neurone et d_k : la sortie désirée.
Pour cela la méthode de Backpropagation est appliquée.
On a donc :

$$\Delta w_{pq} = -\alpha \sum \frac{\partial E_i}{\partial w_{pq}} = \sum \frac{\partial z_i^*}{\partial u_i^*} \frac{\partial u_i^*}{\partial w_{pq}} \tag{II.75}$$

(α : le taux d'apprentissage)

Or, contrairement aux réseaux statiques, le calcul de la dérivée par rapport à un certain poids ne concerne pas les deux neurones qui y sont directement connectés seulement, mais tous les neurones ayant un lien direct ou indirect avec ce poids.
De l'équation II.75, on tire la relation suivante propre aux réseaux dynamiques :

$$\frac{\partial z_i^*}{\partial w_{pq}} = f'(u_i^*)(\delta_{ip} z_q^* + \sum w_{ij} \frac{\partial z_j^*}{\partial w_{pq}}) \tag{II.76}$$

où :δ représente le symbole de Kröneker.

La méthode de F.J Pineda, consiste à considérer un second réseau identique au premier, à chaque étape de l'apprentissage. Ce réseau est défini par une relation identique à l'équation II.68.

$$\dot{v}_i = -v_i + \sum_p v_p f'(u_p) w_{pi} + E_i \tag{II.77}$$

Cette méthode d'entraînement peut être résumée par l'application des étapes suivantes :
Obtenir l'état d'équilibre du réseau régi par l'équation II.68 et calculer l'erreur E à sa sortie.

Obtenir les états d'équilibre v_p^* du réseau auxiliaire défini par l'équation II.75.

La réadaptation des poids synaptiques est déduite à partir des équations II.75 et II.76.

$$\Delta w_{pq} = \alpha f'(u_p^*) v_p^* z_q^* \tag{II.78}$$

Pour les valeurs initiales des états du réseau principal, du fait que ce qui est recherché n'est rien d'autre que les états d'équilibre, elles peuvent être initialisées à des valeurs aléatoires[Hun92].

Ce réseau doit converger vers des minimums (points d'attractions) qui sont représentés par les états désirés. En pratique, la considération des deux réseaux risque de rendre l'apprentissage lent, notamment pour l'établissement des états d'équilibres des équations II.68 et II.77. De plus, les entrées doivent être maintenues jusqu'à l'établissement du régime permanent, c'est ce qui constitue les inconvénients de ce réseau.

Ces réseaux peuvent aussi être entraînés à suivre une trajectoire dans le temps. Dans ce cas, c'est un Trajectory Learning[Bar89]. Williams et Zipser ont proposé une méthode que nous présentons ci dessous.

Real time recurrent learning algorithm

Cet algorithme, qui a été proposé par R.J Williams et D.Zipser[Wil89] minimise l'erreur en sortie sur une trajectoire entre $t=t_0$ et $t=t_1$.
Sous sa version discrète, le réseau de la figure II.12 est régi par :

$$\begin{gathered} u_k(t)=\sum_{l\in\Omega} w_{kl} y_l \sum_{l\in A} w_{kl} x_l = \sum_{l\in\Omega\cup A} w_{kl} z_l(t) \\ y_k(t+1)=f(u_k(t)) \end{gathered} \tag{II.79}$$

Notons que contrairement au cas précédent, les poids reliant les neurones à la couche d'entrée ne sont pas fixés à 1.
Nous définissons donc à chaque instant t l'erreur :

$$\begin{aligned} &E_k(t)=d_k-y_k(t) \quad k\in T \text{ (champs de sortie)} \\ &E_k(t)=0 \qquad\quad \text{sinon} \end{aligned} \tag{II.80}$$

L'erreur définie sur toutes les unités à l'instant t est :

$$J(t)=\frac{1}{2}\sum_{k\in\Omega}\left[E_k(t)\right]^2 \tag{II.81}$$

Ainsi, si le réseau fonctionne $t=t_0$ et $t=t_1$, le critère à minimiser devient :

$$J_{total}(t_0,t_1)=\sum_{t=t_0+1}^{t_1} J(t) \tag{II.82}$$

A chaque instant t, en appliquant la Backpropagation sur le critère défini par 79, on cumul un terme de réadaptation sur les poids.

$$\Delta w_{pq}(t)=-\alpha\frac{\partial J(t)}{\partial w_{pq}} \tag{II.83}$$

de II.80 on a :

$$\frac{\partial J(t)}{\partial w_{pq}} = -\sum_{k\in\Omega} E_k(t)\frac{\partial y_k(t)}{\partial w_{pq}} \tag{II.84}$$

Le problème est donc, toujours, le calcul de la dérivée sur l'équation II.84. A partir deII.79 et II.80 on a :

$$\frac{\partial y_k(t+1)}{\partial w_{pq}} = f'(u_k(t))\left(\delta_{pk} z_q + \sum_{j\in A\cup\Omega} w_{kj}\frac{\partial z_j}{\partial w_{pq}}\right) \tag{II.85}$$

En posant :

$$p_{pq}^k(t) = \frac{\partial y_k(t)}{\partial w_{pq}} \tag{II.86}$$

Et en utilisant II.80, on tire une relation récurrente :

$$p_{pq}^k(t+1) = f'(u_k(t))\left[\sum_{l\in\Omega} w_{kl} p_{pq}^l(t) + \delta_{pk} z_q(t)\right] \tag{II.87}$$

Avec :

$$p_{pq}^k(0) = 0$$

L'application de cette méthode peut donc, être résumée par les étapes suivantes :
A chaque étape entre $t=t_0$ et $t=t_1$:
Calculer $p_{pq}^k(t)$ en utilisant les équations II.80 et II.88
Calculer la réadaptation des poids :

$$\Delta w_{pq}(t) = \alpha \sum_{k\in\Omega} E_k(t) p_{pq}^k(t) \tag{II.88}$$

La correction totale à appliquer aux poids, à la fin de la trajectoire se calcule par la somme des termes de réadaptation calculées à chaque instant.

$$\Delta w_{pq} = \sum_{t=t_0+1}^{t_1} \Delta w_{pq}(t) \tag{II.89}$$

Cet algorithme garde les poids constants pendant tout l'intervalle. Williams et Zipser ont proposé une autre version, appelée RTRL (*Real Time Reccurent Learning*), ou la réadaptation des poids est effectuée à chaque étape t . Enfin pour forcer les sorties du réseau à prendre les valeurs désirées plus rapidement. Une version plus complète de cet algorithme, semblable au RTRL, consiste à remplacer des sorties de réseaux dans les équations (II.79) et (II.87) par les sorties désirées. Cette méthode est appelée : *Teached-Forced Real Time Reccurent Learning* .

II.4.9.3 Réseaux des neurones à une couche cachée dynamique :

Ce modèle en un réseau à une couche cachée qui est la seule à être récurrente, les couches d'entrée et des sorties étant statiques.
Il a été établi que ce type de réseau est plus efficace que les réseaux entièrement interconnectés. En effet, les performances du réseau du neurones ne s'améliorent pas par la simple intensification de feed-back entre neurones. Bien au contraire dans la plupart des cas

les réseaux fonctionnent mieux lorsque le feed-back ne concerne qu'un groupe limité de neurones. En identification par exemple, L.Ljung a mentionner que le choix d'un modèle ayant un minimum de paramètre, pouvant identifier le système, est toujours mieux [cha95]. Un nombre réduit de poids pour ce cas ne peut donc être que préférable.

Réseaux à une couche cachée entièrement interconnecté

Ce type de réseaux contient une seule couche cachée dynamique, qui envoie ses sorties vers la couche de sorties dont les neurones sont linières (figure II.13)
A.Karakasoglu a proposé un modèle, qu'il a utilisé en identification et en commande d'un bras manipulateur [Kark93]. C'est ce modèle que nous allons étudié.

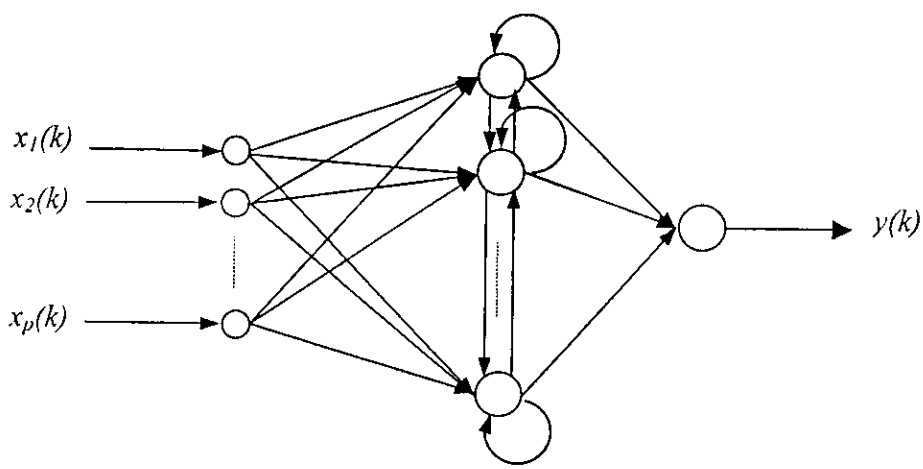
$$U=-U+Wf(U)+BX(k) \tag{II.90}$$

$$y(k)=h^{T}U^{*} \tag{II.91}$$

ou U $\in\Re^{q}$ et f : $\Re^{q}$ ⟶ $\Re^{q}$ est un vecteur composé de fonctions sigmoïde. W $\in\Re^{q}$ $\Re^{q}$

B$\in\Re^{q}$ $\Re^{q}$ et h $\in\Re^{q}$ représentent les matrices des poids de connections entre neurones.

U dans (II.73) représente le vecteur des états d'équilibre stable de l'équation (II.90).pour les entrées $X(k)$ à l'instant k. le réseau décrit ci-dessus est à une seule sortie dans sa dernière couche.



Couche cachée complètement connectée

Figure II.13
Architecture d'un réseau à une couche cachée dynamique

A l'équilibre, les équations (II.90) et (II.91) donnent :

$$u_i^{*}=\sum_{j=1}^{q} w_{ij} f(u^{*})+\sum_{l=1}^{p} b_{il} x_l(k) \tag{II.92}$$

$$y(k)=\sum h_i u_i^{*} \tag{II.93}$$

ou $u_i^{*}, w_{ij}, f_j, x_l, b_{il}, et\ h_i$ sont respectivement les éléments des matrices U^{*}, W, f, X, B et h.
les équations (II.93) et (IV. *91*) indiquent que la sortie du réseau est linéaire.

Si à l'instant K, la sortie désirée est d (k), l'erreur à minimiser sera :

$$E(k) = [d(k) - y(k)]^2 = [d(k) - h^T U^*] \tag{II.94}$$

La méthode de Backpropagation est appliquée :

$$h_i(k+1)=h_i(k)-\eta_i \frac{\delta E(k)}{\delta h_1(k)} \tag{II.95}$$

$$w_{ij}(k+1)=w_{ij}(k)-\eta_2 \frac{\delta E(k)}{\delta w_{ij}(k)} \tag{II.96}$$

Des équations (II.94) et (II.95) peut aisément déduire :

$$h_i(k+1)=h_1(k)+\eta_1[d(k)-y(k)]u_i^* \tag{II.97}$$
$$i=1,\ldots,q$$

$$w_{ij}(k+1)=w_{ij}(k)+\eta_2 h_i(k)[d(k)-y(k)]f_j(u^*_j) \tag{II.98}$$
$$i,j=1,\ldots,q$$

$$b_{ij}(k+1)=b_{ij}(k)+\eta_3 h_j(k)[d(k)-y(k)]x_j(k) \tag{II.99}$$
$$i=1,\ldots,q \text{ et } j=1,\ldots,p$$

La stabilité de réseau dépend beaucoup des poids initiaux et des choix des taux d'apprentissage. Ces derniers doivent être, de préférence, décroissant en fonction du temps. De plus, certaines conditions sur les paramètres des fonctions d'activations, doivent être vérifiés pour assurer la convergence. Des fonctions qui se saturent rapidement devraient permettre une convergence rapide[Karc93].

II.4.9.4 Réseaux diagonalement récurrents DRNN :

La différence de ce réseau par rapport au réseau précédent, est que sa couche cachée ne comporte de feed-back qu'entre chaque neurone et lui-même (figure II.14). Chao-Chee Ku a démontré que ce réseau, de structure moins complexe, est plus efficace que le réseau dont la couche cachée est entièrement interconnectée. Cette efficacité est notamment sensible en identification et en contrôle, où le nombre des paramètres du réseau est minimisé[Cha95].

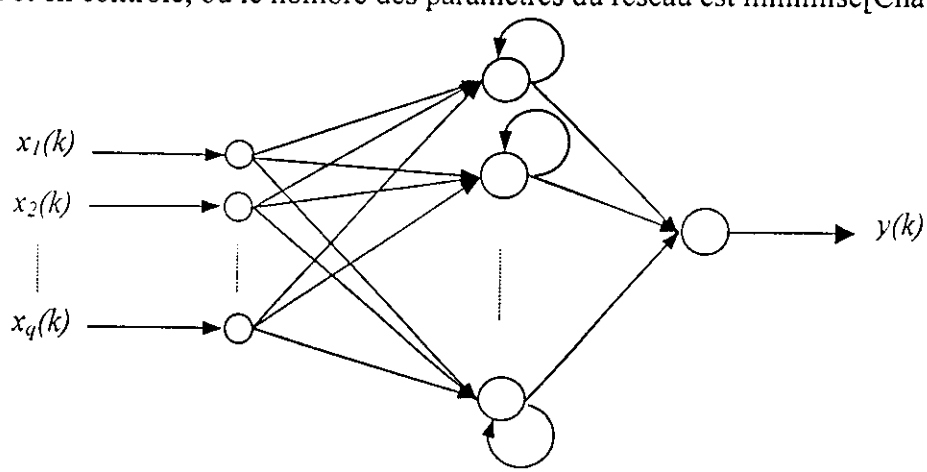


Figure II.14
Architecture d'un réseau récurrent diagonal (DRNN)

Pour un modèle discret, le fonctionnement du réseau est décrit par la figure II.14 :

$$y(k)=\sum_{j=1}^{d} w_j^o z_j(k) \tag{II.100}$$

$$z_j(k)=f(u_j(k)) \tag{II.101}$$

$$u_i(k)=w_i^D z_i(k-1)+\sum_{j=1}^{q} w_{ij}^I x_j(k) \tag{II.102}$$

Où pour chaque instant k :
$x_i(k)$: représente la *ième* entrée.
$z_i(k)$: est la sortie du *jème* neurone récurrent.
$u_i(k)$: est la somme pondérée des entrés du *jème* neurone récurrent.
$O(k)$: la sortie du DRNN.

La fonction $f(.)$ dans l'équation II.98 est la fonction sigmoïde et les matrices $W^I \in R^q$ et $w^D \in R^d$ et $W^O \in R^{dxq}$ dans les équations II.98 et II.100 sont respectivement les poids en provenance de la couche d'entrée vers la couche cachée, ceux reliant chaque neurone avec lui même dans la couche récurrente, et ceux allant de la couche cachée vers la sortie. Notons que dans notre cas, on a qu'une seule sortie.

Si à l'instant k, la sortie désirée est $d(k)$, l'erreur à minimiser sera :

$$E(k)=\frac{1}{2}[d(k)-y(k)]^2=\frac{1}{2}[d(k)-W_o z(k)]^2 \tag{II.103}$$

En appliquant la Backpropagation, on a :

$$W(k+1)=W(k)-\eta\frac{\partial E(k)}{\partial W(k)} \tag{II.104}$$

A partir des équations 97,98,99 et 100, on obtient :

$$\begin{aligned} w_i^O &= w_i^O(k)+\eta_1[d(k)-y(k)]z_i(k) \\ w_i^D &= w_i^D(k)+\eta_2[d(k)-y(k)]W_i^O P_i(k) \\ w_{ij}^I &= w_{ij}^I(k)+\eta_3[d(k)-y(k)]W_i^O Q_{ij}(k) \end{aligned} \tag{II.105}$$

Avec :

$$\begin{aligned} P_i(k) &= \frac{\partial z_i(k)}{\partial w_i^D} \\ Q_{ij}(k) &= \frac{\partial z_i(k)}{\partial w_i^I} \end{aligned} \tag{II.106}$$

A partir de II.98 et II.99, on peur déterminer les expressions de II.103 à chaque instant par un calcul itératif

$$P_i(k)=f'(u_i)(z_i(k-1)+w_{iD}P_i(k-1))$$
$$Q_{ij}(k)=f'(u_i)(x_i(k-1)+w_{iD}Q_{ij}(k-1)) \quad \text{(II.107)}$$

Avec :

$P_i(0)=0.$
$Q_{ij}(0)=0.$

Cette méthode d'apprentissage garantit la convergence, sous conditions que les taux d'apprentissage ne soient pas grands. Un choix d'un taux adaptatif consiste à calculer des valeurs optimales pour les trois paramètres η_1, η_2 et η_3. Chao-Chee Ku a montré dans un problème d'identification, pour lequel il a utilisé ce réseau, que les valeurs optimales de ces paramètres peuvent être calculées par la formule suivante :

$$\eta^* = \frac{1}{g_2(k)_{max}} \quad \text{(II.108)}$$

II.4.9.5 Réseaux Récurrents et approximation de fonctions dynamiques

L'architecture des réseaux récurrents permet, à travers sa structure interne, de faire une représentation des systèmes dynamiques. Mais dans ce contexte, on ne peut transposer directement les propriétés d'approximation universelle, dont jouissent les réseaux statiques vers ce type de réseaux qui ont une structure différente.

La propriété d'approximation universelle des réseaux de neurones prend un sens différent s'il s'agit d'un réseau Dynamique.
Cette propriété s'énonce comme suit:

Soit un processus défini par la relation:

$$y(k+1)=f(y(k),u(k)) \quad \text{(II.109)}$$

Pour tout système de la forme (II. 106), pour toute précision désirée ε, pour un intervalle de temps fini $[0,T]$, pour des entrées $u(.) : [0,T] \rightarrow U \subseteq \Re^n$ et un état initial $x(0) \in X \subseteq \Re^n$, il existe un réseau de neurone bouclé qui approche le comportement Entrée-Sortie du système dont le comportement est décrit par l'équation (II.88) avec la précision ε sur l'intervalle $[0,T]$ *et* sur les ensembles U et X (Riv 95).

La définition de l'approximation du comportement d'un processus dynamique par un réseau récurrent n'est donc pas globale, mais restreinte à un certain intervalle fini. Un tel approximateur peut donc ne pas refléter toutes les caractéristiques de la fonction à approximer, notamment dans le cas de l'identification des systèmes.

A la fin de ce chapitre, il est important de noter que les recherches sur les différentes architectures possibles des réseaux de neurones et leurs algorithmes d'apprentissage se poursuivent toujours. Ceci, afin de rendre les possibilités de ces réseaux en approximation universels, notamment dans le cadre de modélisation de phénomènes physiques plus intéressantes et leur apprentissage moins ardu. Dans ce cadre, nous citons le réseau développé par T.J.van der Walt (Van 95) et al, appelé *Regression Network*. Ce réseau constitue une

architecture de neurones très souple dont l'architecture contient des fonctions d'activation et des discriminant de différentes formes. Ainsi on peut trouver dans un même réseau des neurones avec des fonctions d'activation sigmoïdes et d'autre avec des fonctions exponentielles, linéaires, sinusoïdales, ou logarithmiques. Par conséquent, à l'entrée de chaque neurone, la fonction de base peut être additive ou multiplicative, Une telle structure offre au réseau une grande flexibilité, En effet, moyennant quelques connaissances a priori sur le système à modéliser, un choix judicieux des fonctions d'activation peut être fait, Une fonction connue comme étant linéaire, par exemples, nécessite un nombre important de neurones à fonction d'activation sigmoïde, mais peut être appropriée avec une structure très simple, à base de neurones linéaire. L'apprentissage des régression Network est une application directe de la méthode de Backpropagation, où la forme de la dérivée de l'erreur diffère suivant la fonction d'activation du neurone en question. Cette structure a été utilisé notamment dans la modélisation des phénomènes physiques et a montré des résultats encouragent (Van95).

Exemple d'approximation de fonction

La fonction utilisée est $y=\sin(\pi x^3)$ pour $x \in [-1 \quad 1]$, le réseau utilisé possède une seul couche cachée à quatre neurones avec des fonctions d'activations tangente hyperbolique pour les neurones de la couche cachée et linéaire pour la sortie du réseau. Le réseau est entraîné avec l'algorithme de rétropropagation avec une minimisation de l'erreur sur l'ensemble des exemples (bloc learning), l'apprentissage est effectué pondant 50 itérations. une itération est le traitement complet de tous les exemples de la base d'apprentissage qui dans ce cas formé par 80 exemples d'apprentissage, ce qui fait 4000 itérations pondant l'apprentissage dans le cas d'un data learning. La courbe d'apprentissage est représentée par la figure suivante :

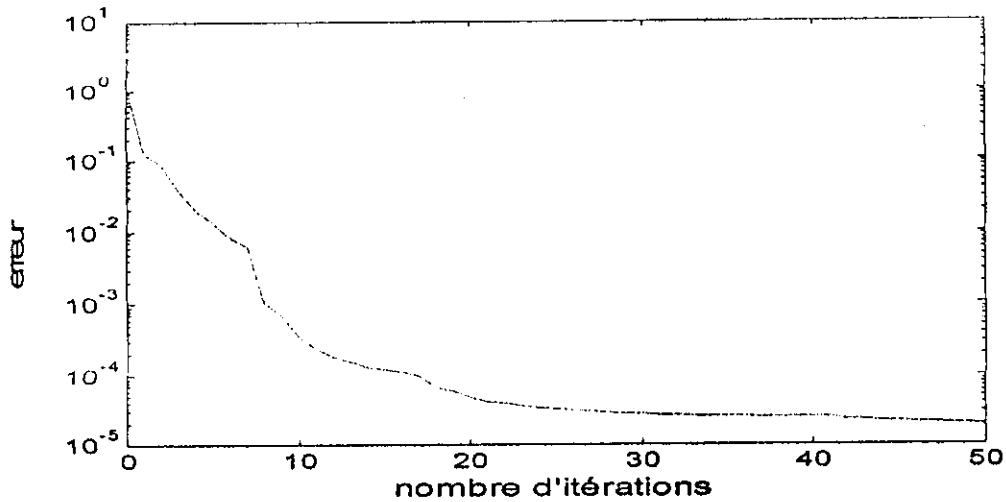


Figure II.15
Représentation de l'évolution de l'erreur en fonction des itérations
(L'échelle des ordonnées est une échelle logarithmique)

après l'apprentissage en présente au réseau des exemples de teste de généralisation. Les deux sorties (sortie réelle et celle du réseau) sont représentées par la figure II.16.

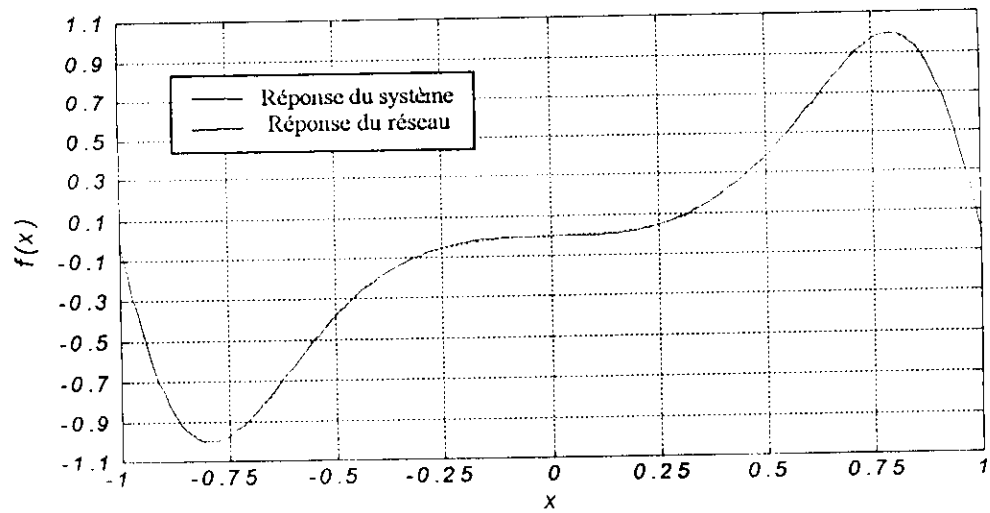


Figure II.16
Test de généralisation

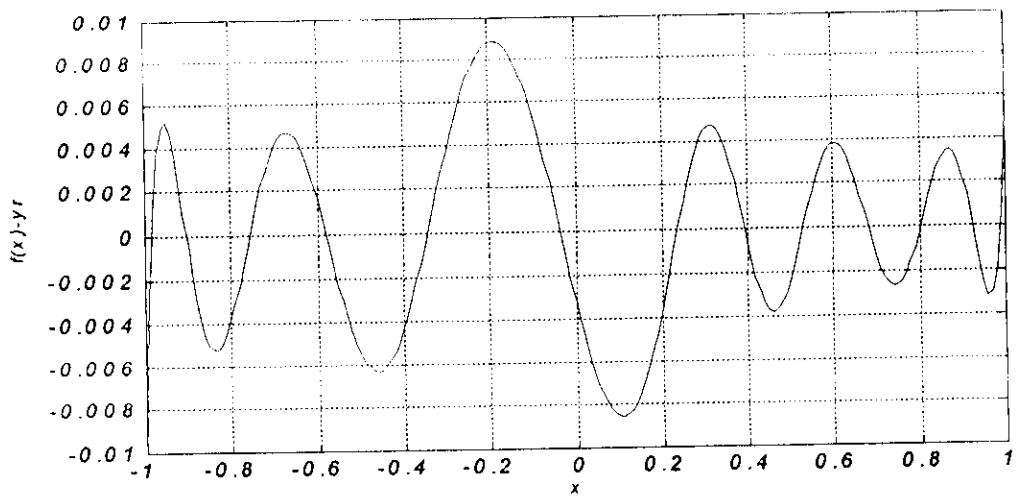


Figure II.17
Erreur de validation du réseau après l'apprentissage

Afin de voir l'influence du nombre de neurone sur la possibilité d'approximation du réseau, on a approximé la même fonction précédente par différents types de réseau avec le même nombre d'itérations qui fixé à 20 itérations dans le cas d'un bloc learning, (2000 itérations dans le cas d'un data learning).

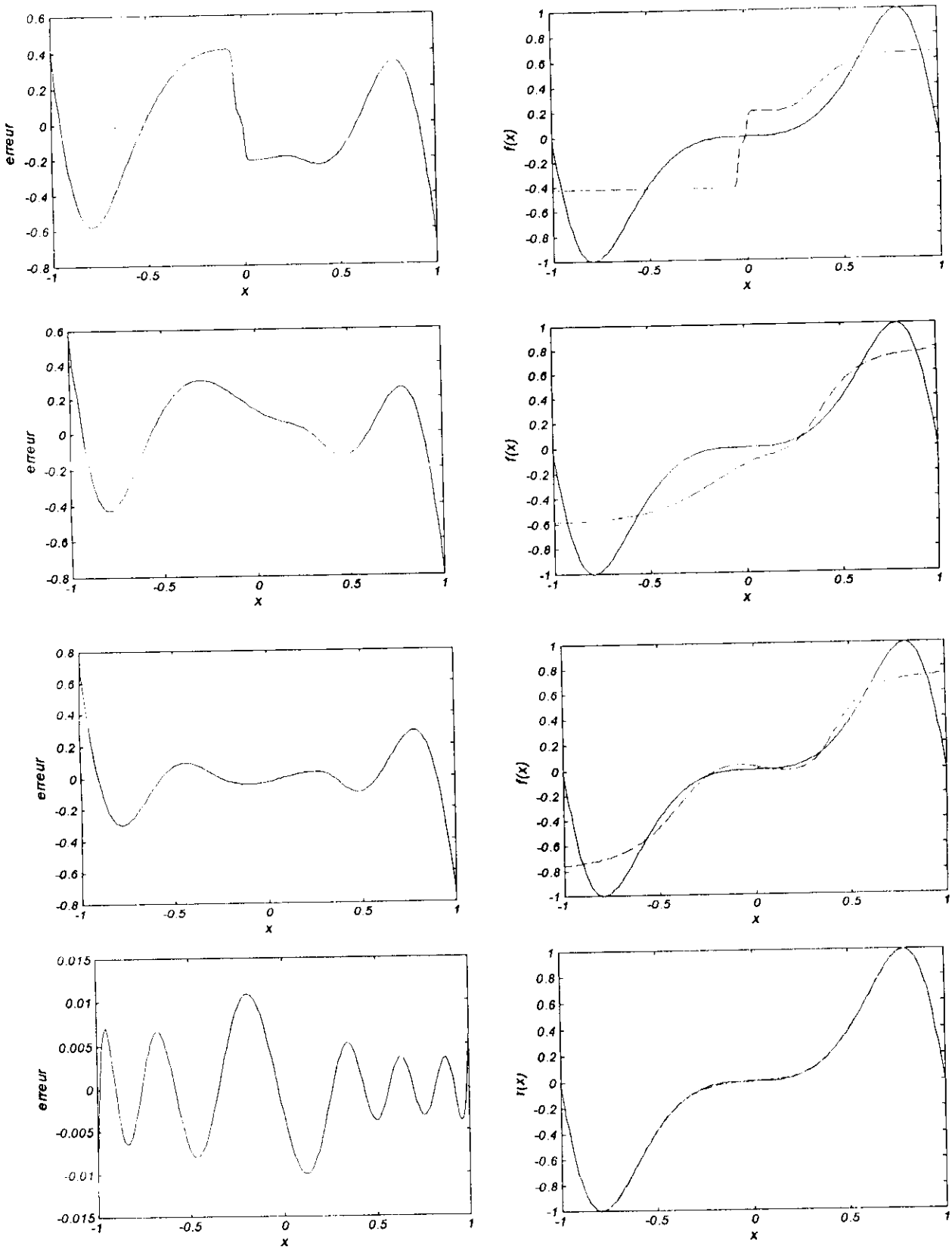


Figure II.18
Influence du nombre de neurone sur la capacité d'apprentissage.
(pour 3, 4, 5 et 7 neurones respectivement dans la couche cachée)

III.1 Structures d'identification neuronale

L'aptitude des RNA à simuler des relations non linéaires, les met au premier rang pour qu'ils soient appliqués à l'identification des processus. Entraîner des réseaux en utilisant les entrées\ sorties du système non linéaire, n'est autre qu'un problème d'approximation de fonction. De la même manière qu'une fonction de transfert représente la boite noire d'un système linéaire. Un réseau neuronal peut représenter la boite noire d'un système non linéaire.

Un certain nombre de résultats ont été publiés, montrant qu'un réseau multicouches peut bien approximatif des fonctions continues.

Le problème d'identification neuronale est décrit, d'une manière générale, dans la figure (IV.8). Ca consiste à ajuster les paramètres du modèle d'identification, dans le but d'optimiser une fonction performance basée sur l'erreur entre la sortie du modèle et celle du système.

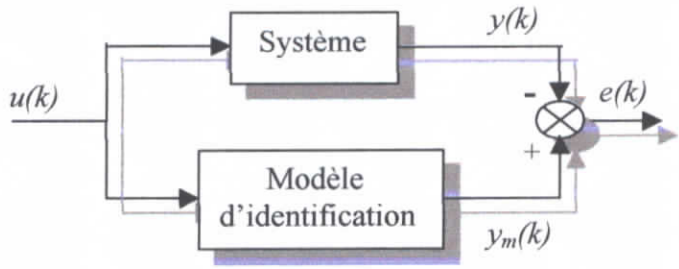


Figure III.1
Structure générale d'identification

III.2 Identification du modèle statique du système

Le RNA à pour rôle de reproduire la statique du système, il s'agit de placer le RNA en parallèle avec le système. L'erreur de prédiction est utilisée comme signal d'entraînement du réseau.
Puisque le RNA est concerné uniquement par la statique du système, il n'a donc pas besoin de poursuivre sa dynamique .La stabilité du système dans la plage de variation des données d'apprentissage (off-line), est exigée pour la convergence de l'erreur.

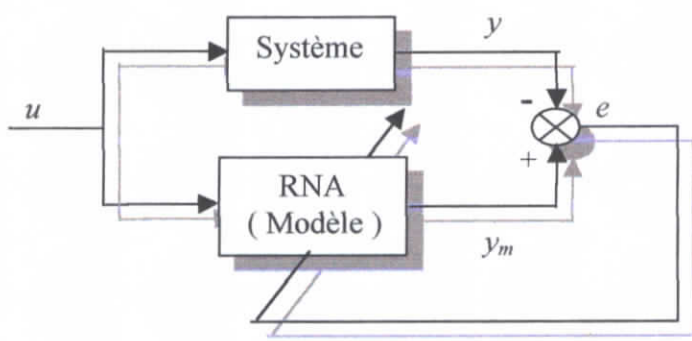


Figure III.2
Identification par RNA du modèle statique du système

La procédure d'entraînement des réseaux de neurones pour représenter les systèmes dynamiques va être référée par l'identification directe. Une structure permettant de réalisé cette tache est schématisée dans la figure III.3.

Le comportement dynamique des système physiques étudiés pose un problème pour l'intégration de cette dynamique dans les réseaux de neurones. Cette dynamique n'est une caractéristique intrinsèque des réseaux statiques. Ainsi, ces derniers peuvent échouer lorsqu'on veut capturer les principales caractéristiques du système. On peut résoudre ce problème soit en utilisant les réseaux récurrents, soit en introduisant un comportement dynamique dans les neurones. Soit en augmentant le nombre d'entrées du réseau, en considérant les sorties et les entrées précédentes [Nar90].

Dans ce cadre, quatre modèles sont introduits pour la représentation des systèmes non-linéaires SISO avec la possibilité de généraliser aux systèmes MIMO. Ces modèles ne sont en fait qu'une extension des modèles utilisés dans les systèmes linéaires aux systèmes non-linéaires.

En tenant compte de ceci, des modèles d'identification vont être proposés en utilisant les réseaux de neurones. Les quatre classes des systèmes considérés peuvent être décrits par les équations aux différences non-linéaire suivantes :

Modèle I

Dans ce modèle, la sortie du système non-linéaire inconnu est assumée dépendante linéairement de ses valeurs précédentes et d'une manière non-linéaire des valeurs passées de l'entrée (figure III.3). L'avantage de ce modèle est qu'il permet de discuter la stabilité en régime libre des systèmes non-linéaires. La classe de ces modèles se caractérise par l'équation au différence suivante :

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

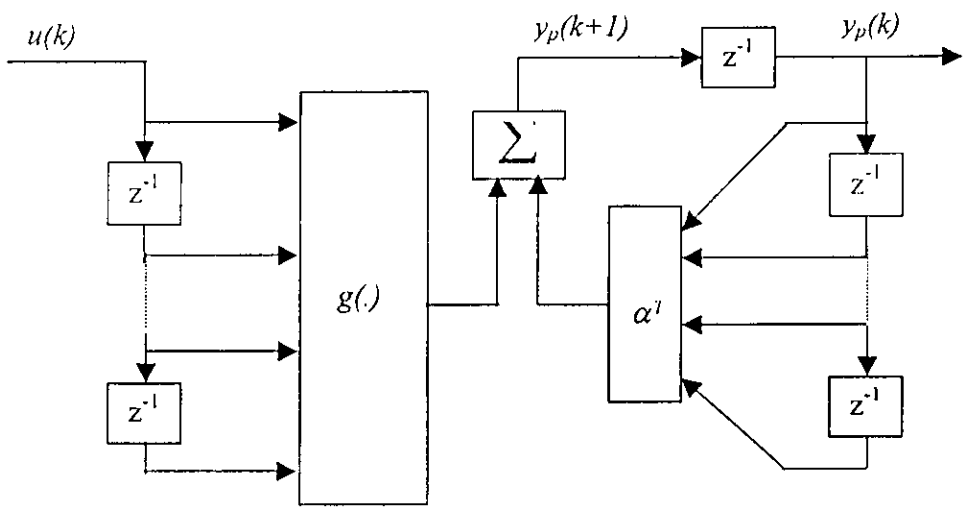


Figure III.3
Représentation du modèle I avec les lignes de retards.

Modèle II

La sortie du système non-linéaire dans ce cas est assumée dépendante linéairement de l'entrée *u(k)* et de ses valeurs passées, et non linéairement de ses propre anciennes valeurs,. L'avantage de ce modèle est qu'il est directement applicable dans la commande par réseaux de neurones.

Ce type de modèle est géré par l'équation au différence suivante :

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

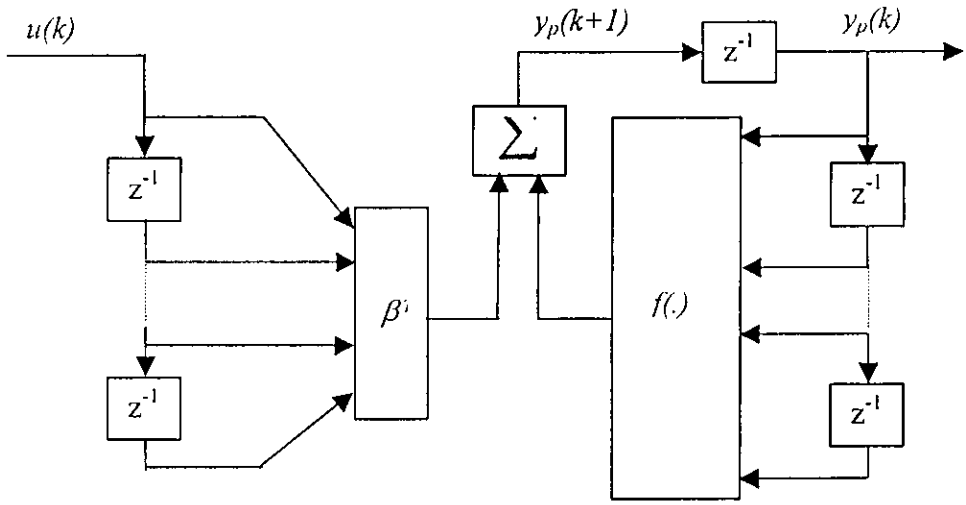


Figure III.4
Représentation du modèle II avec les lignes de retards

Modèle III

Dans ce cas, la sortie du système dépend non-linéairement à la fois de ses valeurs passées et des valeurs passées de l'entrées comme indiqué dans l'équation au différence qui suit :

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

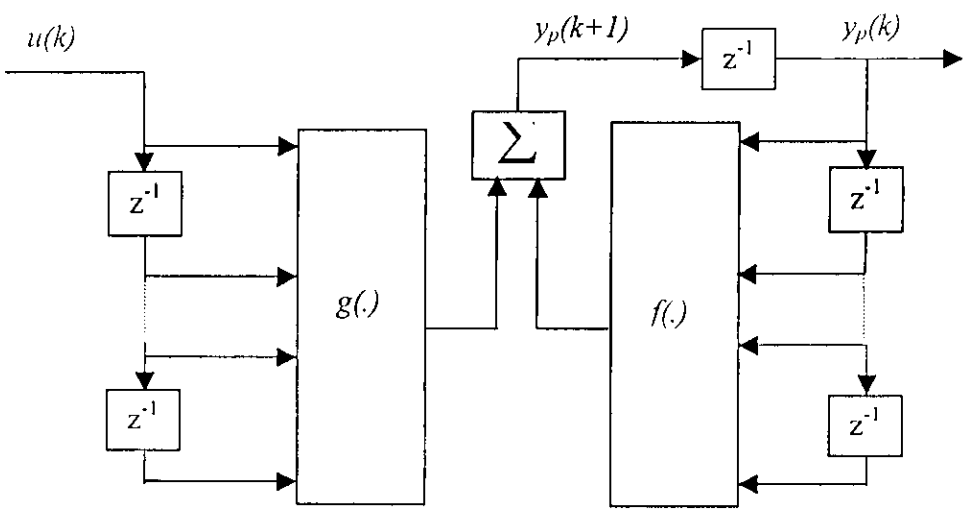


Figure III.5
Représentation du modèle III avec les lignes de retards

Modèle IV

Ce modèle est le plus général, puisqu'il englobe les modèles précédents. la sortie à n'importe quel instant dans ce cas est une fonction non-linéaire des valeurs passées de l'entrée et de sortie à la fois. La représentation de ce modèle sous forme d'une équation au différence et en utilisant les lignes de retards est la suivante :

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

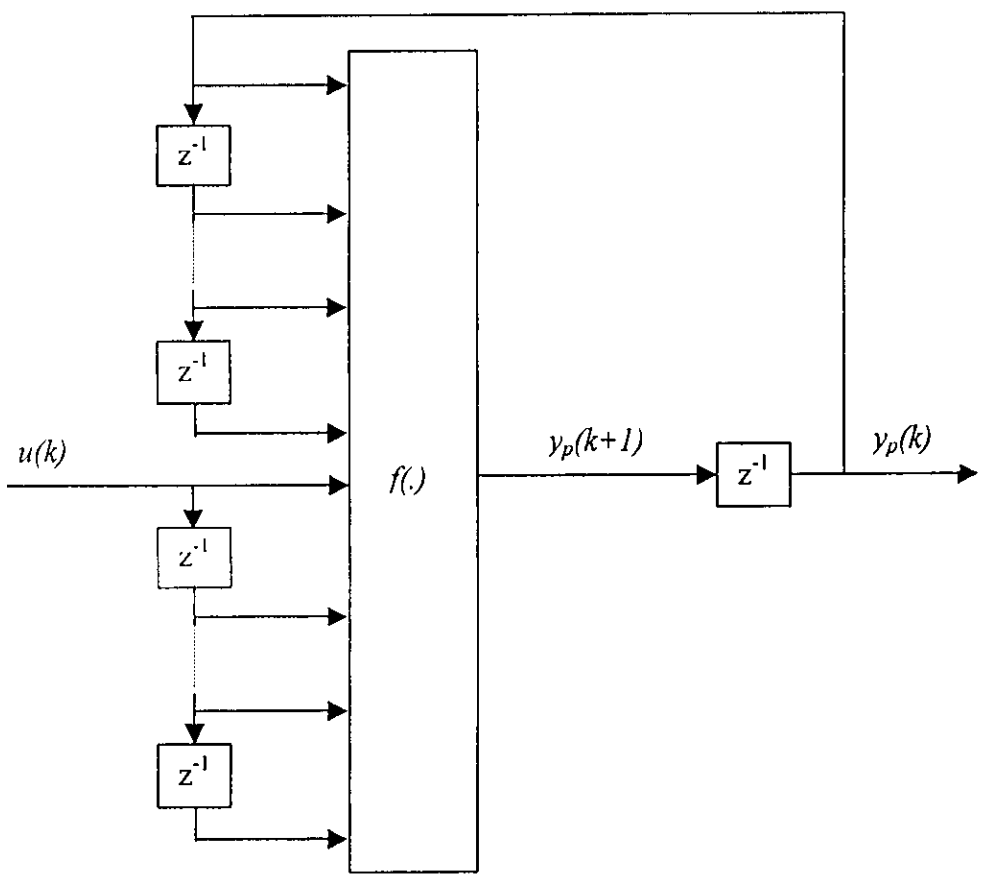


Figure III.6
Représentation du modèle IV avec les lignes de retards

Avec : [$u(k)$, $y_p(k)$] représentant la paire d'entrée sortie du système.
Les fonctions f et g sont supposées dérivables par rapport à leurs arguments.

III.3 Procédure d'identification du modèle dynamique du processus

Le choix de la structure et de la topologie du réseau n'est pas simple non plus. Le modèle d'identification du système est composé du réseau de neurones et des lignes de retards (les entrées et les sortie précédentes). Dans chaque cas le réseau de neurones est supposé contenir suffisamment de paramètres (poids), afin d'être apte à représenter exactement la caractéristique entrée-sortie du processus. Le principe d'apprentissage du réseau est illustré à la figure suivante .

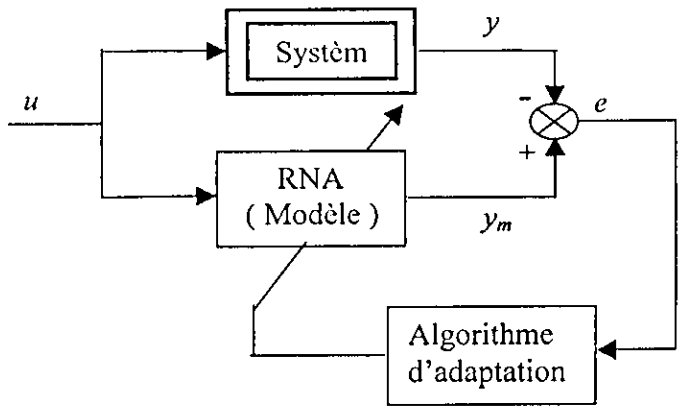


Figure III.7
Identification par RNA du modèle statique du système

Pour identifier le processus un modèle d'identification est choisi basé sur des informations à priori concernant le système. Si $y(k+1)$ et $\hat{y}(k+1)$ sont respectivement la sortie à l'instant $k+1$ du processus et du modèle. L'erreur $e(k+1)= y(k+1)-\hat{y}(k+1)$, est utilisée pour ajuster les poids du réseau de neurones par l'algorithme de la Rétropropagation. Pour cela deux structures générales peuvent être adoptées :

- Identification avec le modèle parallèle.
- Identification avec le modèle serie-parallèle.

III.3.1 Modèle parallèle

La méthode d'identification par le modèle parallèle consiste à utiliser les propre prédiction (sortie) du réseau pour construire ses futures entrées comme indiqué à la figure III.8. Et le modèle d'identification sera le suivant :

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

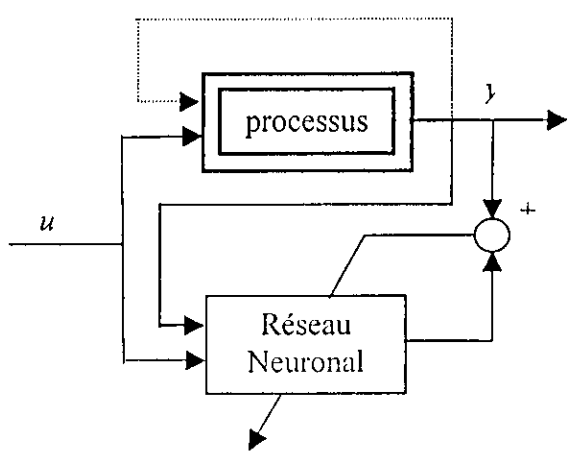


Figure III.8
Architecture d'identification par un modèle parallèle.
(les traits pointillés représentent les rétroactions internes du processus.
Les traits pleins comprennent non seulement le signal indiqué,
mais aussi les valeurs passées du signal).

III.3.2 Modèle serie-parallèle

Contrairement au modèle parallèle, le modèle serie-parallèle utilise les sortie réelles du processus pour construire le vecteur d'entrée du réseau de neurones, figure III.9.
Cette propriété nous permettra d'ajuster les poids à l'aide de la rétropropagation classique, dans ce cas le modèle d'identification sera le suivant :

$$\hat{y}(k+1)=\eta[y(k),y(k-1),...,y(k-n+1),u(k),u(k-1),...,u(k-m+1)] \quad \text{(III.6)}$$

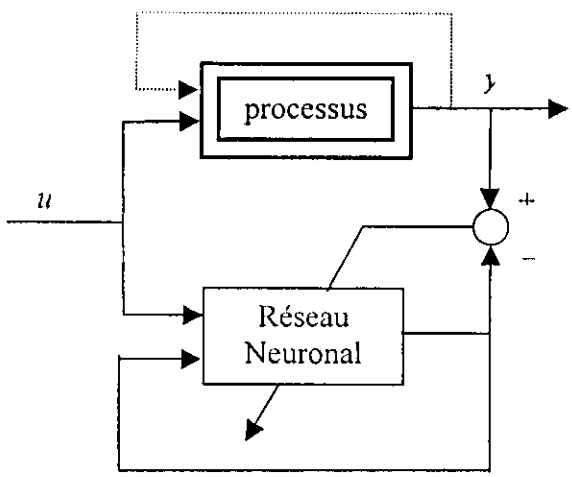


Figure III.9
Schéma d'identification par un modèle serie-parallèle

La deuxième structure (serie-parallèle) est préférable du point de vue de la stabilité, des performances et des simplicités des algorithmes. La première est cas particulier d'une structure plus générale de réseau récurrent. Dans cette veine on pourrait utiliser un réseau tout à fait récurrent (figure III.10), dont l'entrée unique serait $u(k)$ et la sortie $\hat{y}(k)$. Cette approche, rend le réseau libre de développer sa propre représentation d'état, adapter au problème, en particulier, il n'est pas nécessaire de connaître l'ordre du processus, ni son temps mort (nombre de pas de temps entre l'application d'une action de commande u et son effet sur la sortie y).

La représentation interne de la dynamique du processus est donc distribuée à la fois spatialement et temporellement.

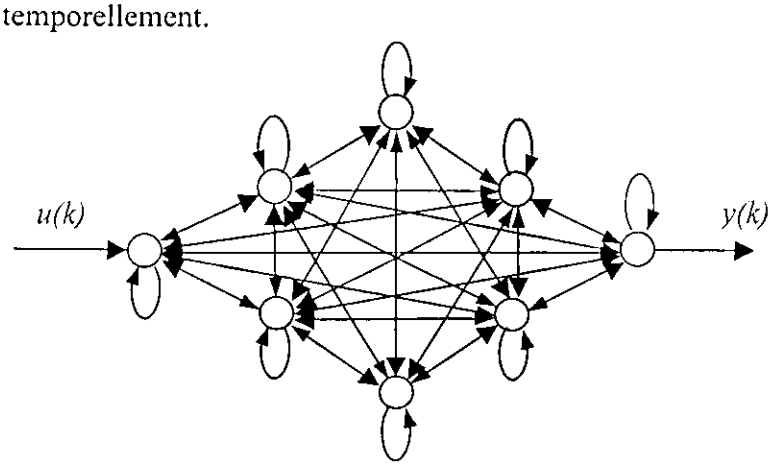


Figure III.10
Réseau récurrent pour l'identification des systèmes dynamiques

III.4 Identification inverse

L'utilisation des réseaux de neurones pour l'identification de la dynamique inverse des systèmes non-linéaires présente une grande promesse pour la recherche, car les modèles inverses des systèmes dynamique ont une utilité immédiate dans la commande. Le problème majeur avec l'identification inverse arrive quand plusieurs entrées produisent la même sortie, autrement dit quand l'inverse du système n'est bien définit, dans ce cas le réseau tend à faire correspondre les mêmes entrées au différentes sorties.

Plusieurs approches existent pour identifier l'inverse des systèmes dynamiques non-linéaires, ils seront discutées dans le prochain chapitre. La figure III.11 montre le schéma général de l'identification inverse.

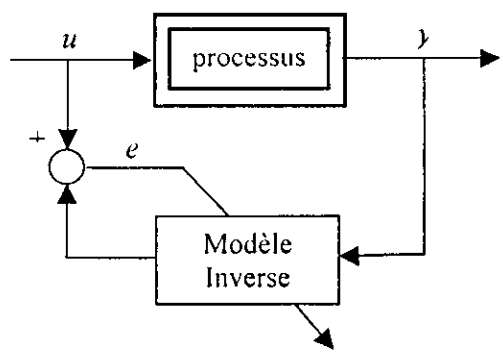


Figure III.11
Structure d'identification du modèle inverse.

III.5 Convergence et Excitation Persistante

Comme en identification classique ou signal d'entraînement des réseaux de neurones est très important pour que le modèle d'identification soit proche du système réel.

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

Une méthode en boucle ouverte est basé sur l'utilisation d'un signal aléatoire uniformément distribué à travers tout l'espace de travail.

III.6 Identification du modèle d'état

Considérons le système dynamique discret non linéaire d'ordre n, régit par l'équation d'état suivante :

$$x(k+1)=f(x(k),u(k)) \qquad \text{(III.7)}$$

si la fonction f est inconnue, un réseau de neurones peut être conçu pour l'approximer, ayant pour exemples d'apprentissage, les état observés du système ainsi que l'action de commande comme indiqué à la figure suivante :

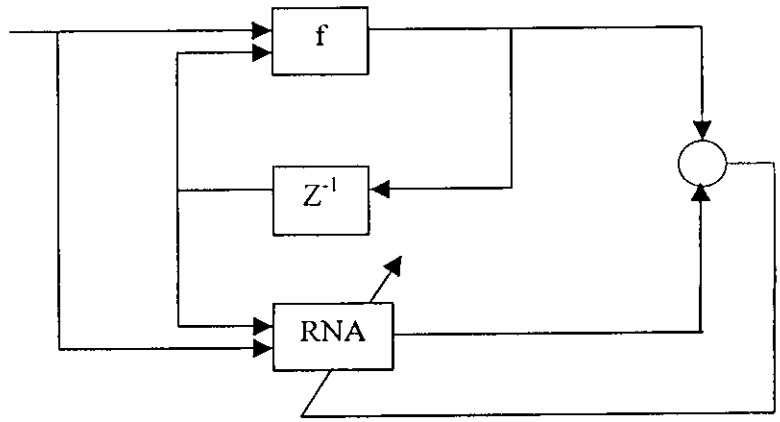


Figure III.12
Identification du modèle d'état du système

L'entraînement du RNA peut s'effectuer on line et pour que l'identification soit possible, quelques informations doivent être connues à priori, telles que l'ordre du système et la nature de la fonction *f*. Les états du système quant à eux, doivent être mesurables. De plus, *f* est assumée être bornée pour tout x et u appartenant à la classe des exemples d'apprentissage.

L'identification du prédicteur associé à un modèle d'état lorsque l'état du processus n'est pas mesuré impose l'utilisation d'un réseau de neurones bouclé. La structure de l'apprentissage du réseau prédicteur est montrée à la figure suivante :

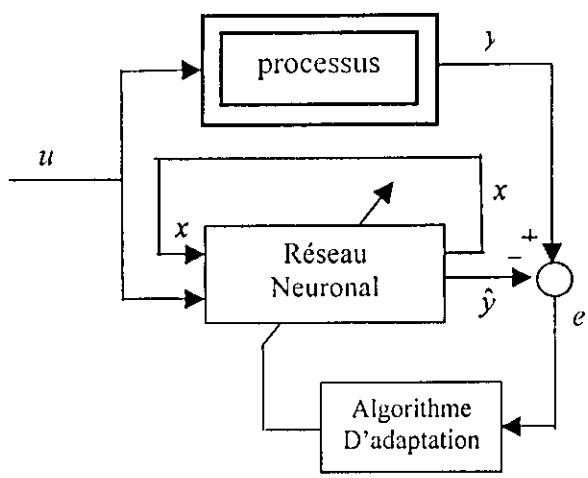


Figure III.13
Structure d'identification du Modèle d'état
(l'état du processus n'est pas mesuré,
le système d'apprentissage utilise un prédicteur bouclé sur l'état).

Le prédicteur (réseau de neurones) qui est utilisé dans le système d'apprentissage précédent est représenté schématiquement sur la figure suivant :

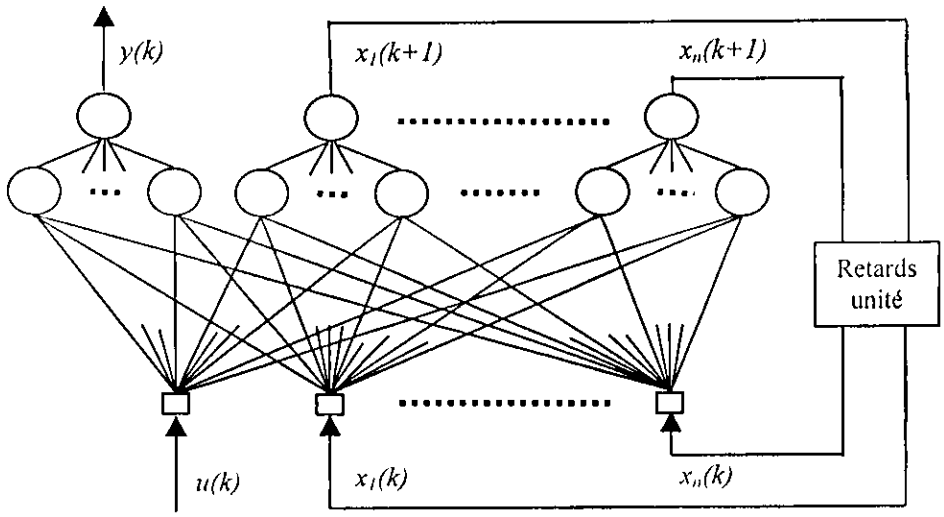


Figure III.14
Structure du Prédicteur associé à un modèle d'état,
lorsqu'on ne mesure pas l'état du processus.

III.7 Identification des paramètres du modèle d'état linéaire

Une autre propriété intéressante, intrinsèque aux réseaux de neurones, est utilisée pour l'identification des paramètres du modèle d'état linéaire. Elle diffère des autres méthodes présenté, du fait que c'est une identification paramétrique, alors que les autres sont des méthodes d'identification non paramétriques, qui ont pour rôle d'imiter le comportement du système. Cette méthode se présente comme suit :

Soit le système linéaire discret d'ordre n décrit par l'équation d'état suivante :

$$x(k+1)=Ax(k)+Bu(k) \tag{III.8}$$

x : vecteur d'état du système.
u : vecteur des action de commande.

L'objectif est d'identifier les matrices A et B qui représentent les paramètres du système, à l'aide d'un réseau neuronal adéquat. Prenant le cas d'un système SISO (le raisonnement est le même pour les systèmes multivariabes). Le modèle précédent devient :

$$x(k+1) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} x(k) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} u(k) \tag{III.9}$$

Augmentons le vecteur x de l'ordre n à l'ordre $n+1$, en lui ajoutant le scalaire u. Ainsi, on obtient :

$$x(k+1)=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & & a_{2n} & b_2 \\ \vdots & & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{bmatrix} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} = A' \begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \tag{III.10}$$

On choisit un réseau ayant $[x(k),u(k)]$ comme entrées, et devra estimer $x(k+1)$ (figure III.15). Si ce réseau possède m couches cachées, alors sa sortie sera exprimée par :

$$\hat{x}(k+1)=\Gamma(W_{m+1}\,\Gamma(W_m....\Gamma(W_1(x(k)u(k))^T)....)) \tag{III.11}$$

avec W_i $(i=1,m+1)$ représentent les matrices poids de la couche d'entrée jusqu'à celle de sortie, et Γ représente le vecteur des fonctions d'activation appliquées aux sommes pondérées des neurones de chaque couche.

Jusqu'ici, on fait juste une identification du modèle d'état du système. Seulement, si on choisit des fonctions d'activation identité (linéaire), ce qui donne naissance à un réseau de neurones linéaire.

La relation précédente devient :

$$\hat{x}(k+1)=W_{m+1}\,W_m....W_1(x(k)u(k))^T \tag{III.12}$$

une fois l'apprentissage est terminé, on aura $\hat{x}(k+1)\approx x(k+1)$, et de fait :

$$A'=W_{m+1}\,W_m....W_1 \tag{III.13}$$

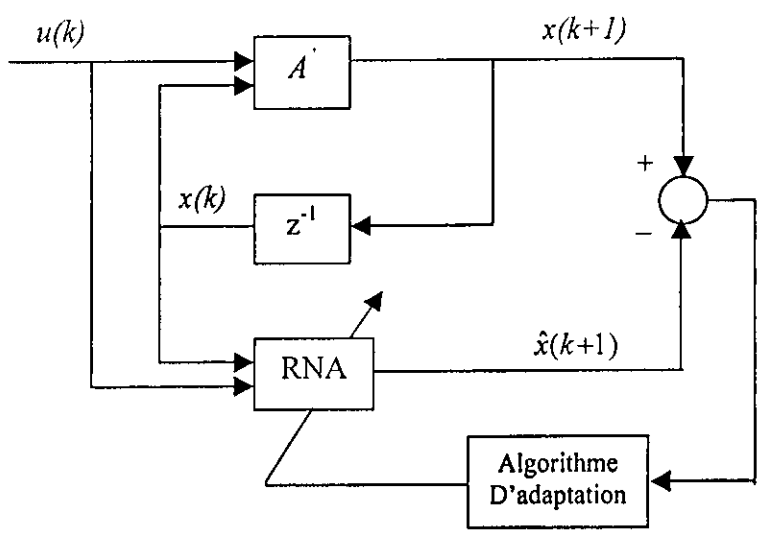


Figure III.15
Identification paramétrique du modèle d'état linéaire.

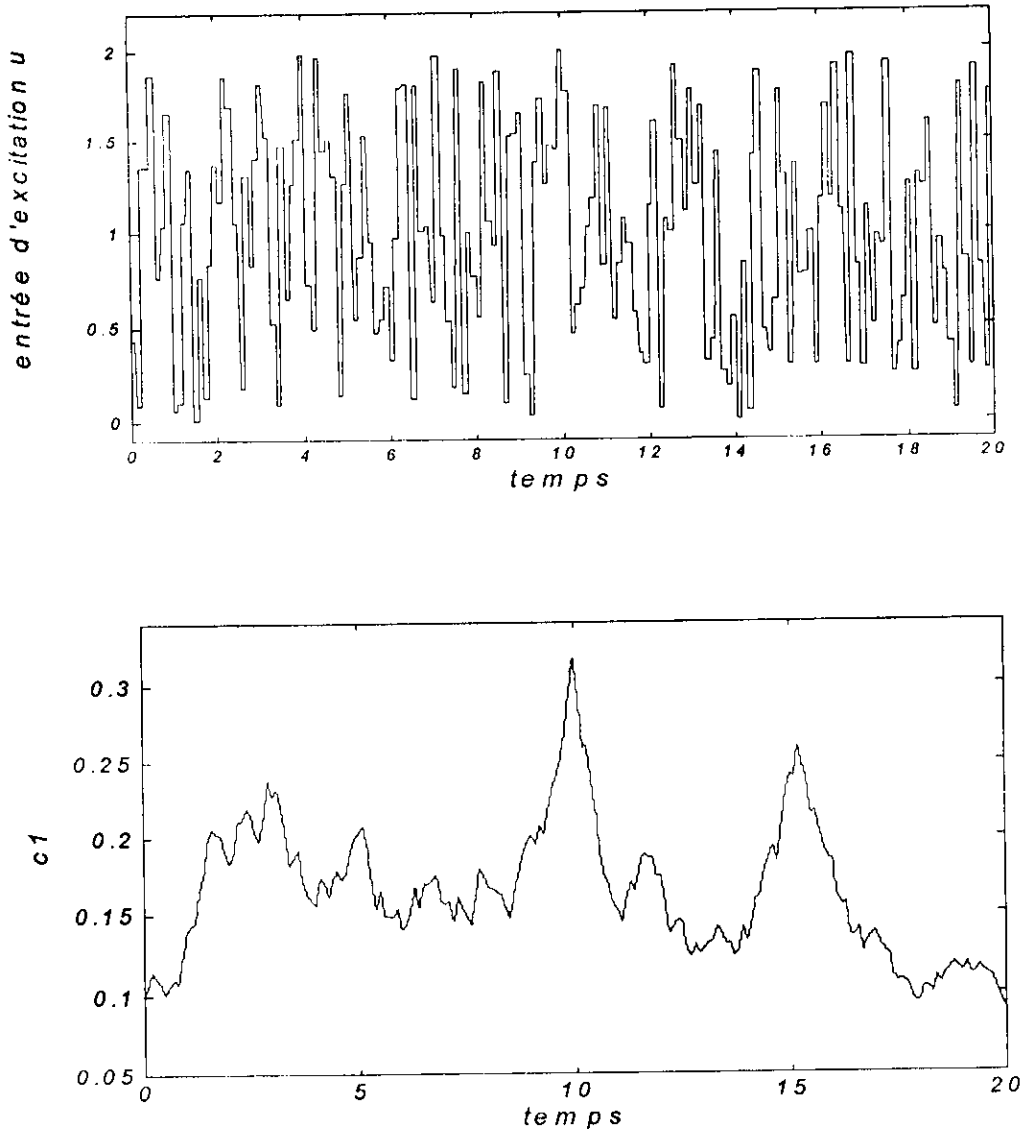
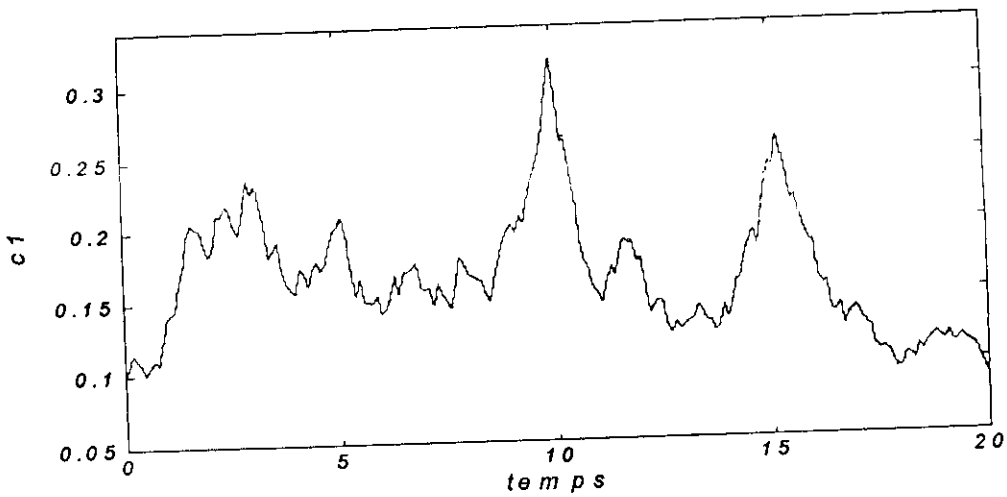
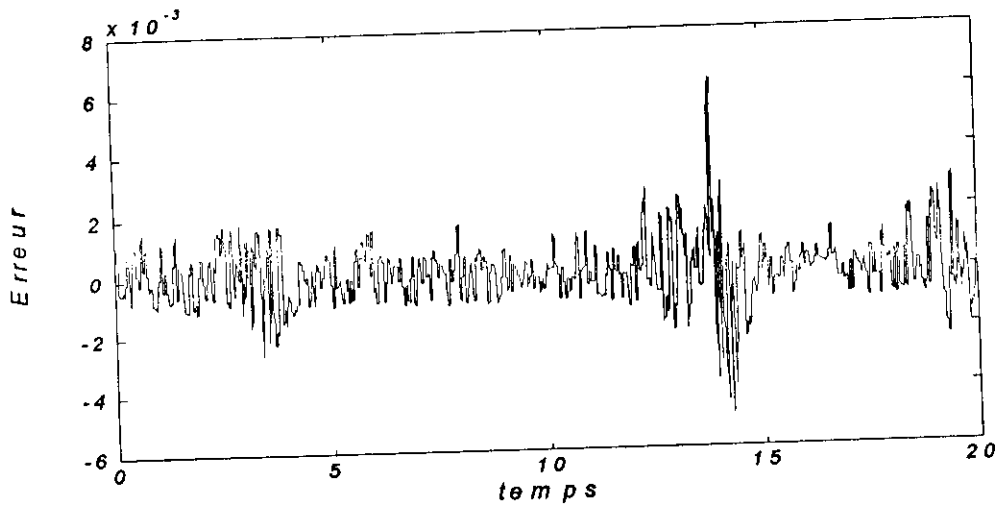


Figure III.16
Représentation du fichier d'entrée sortie du processus



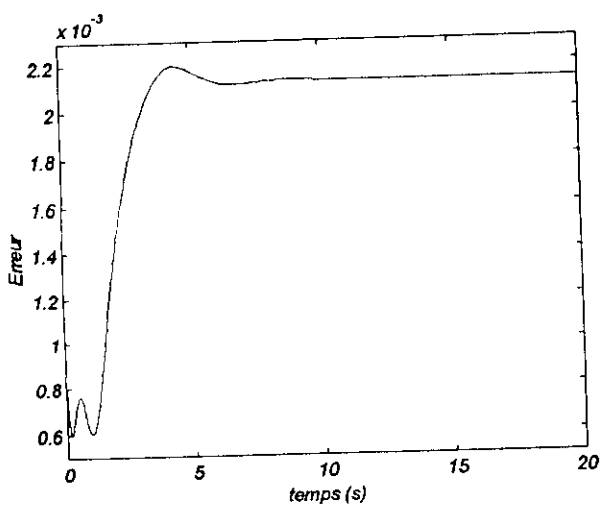
(a)



(b)

figure III.17

(a) validation du modèle , (b) erreur de test



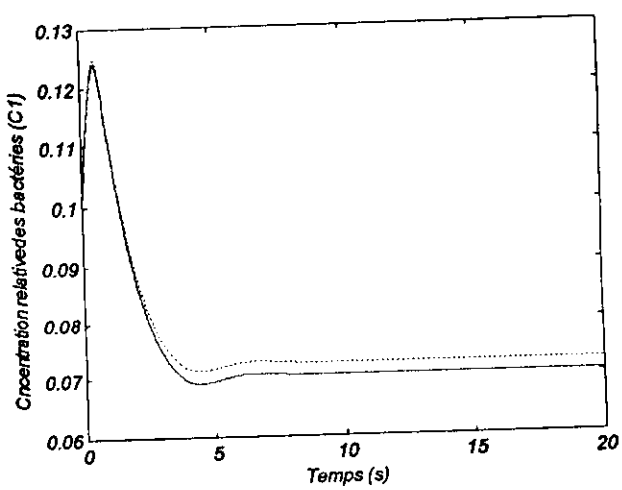


Figure III.18

Simulation en boucle ouverte du réseau de neurones dans la plage de stabilité

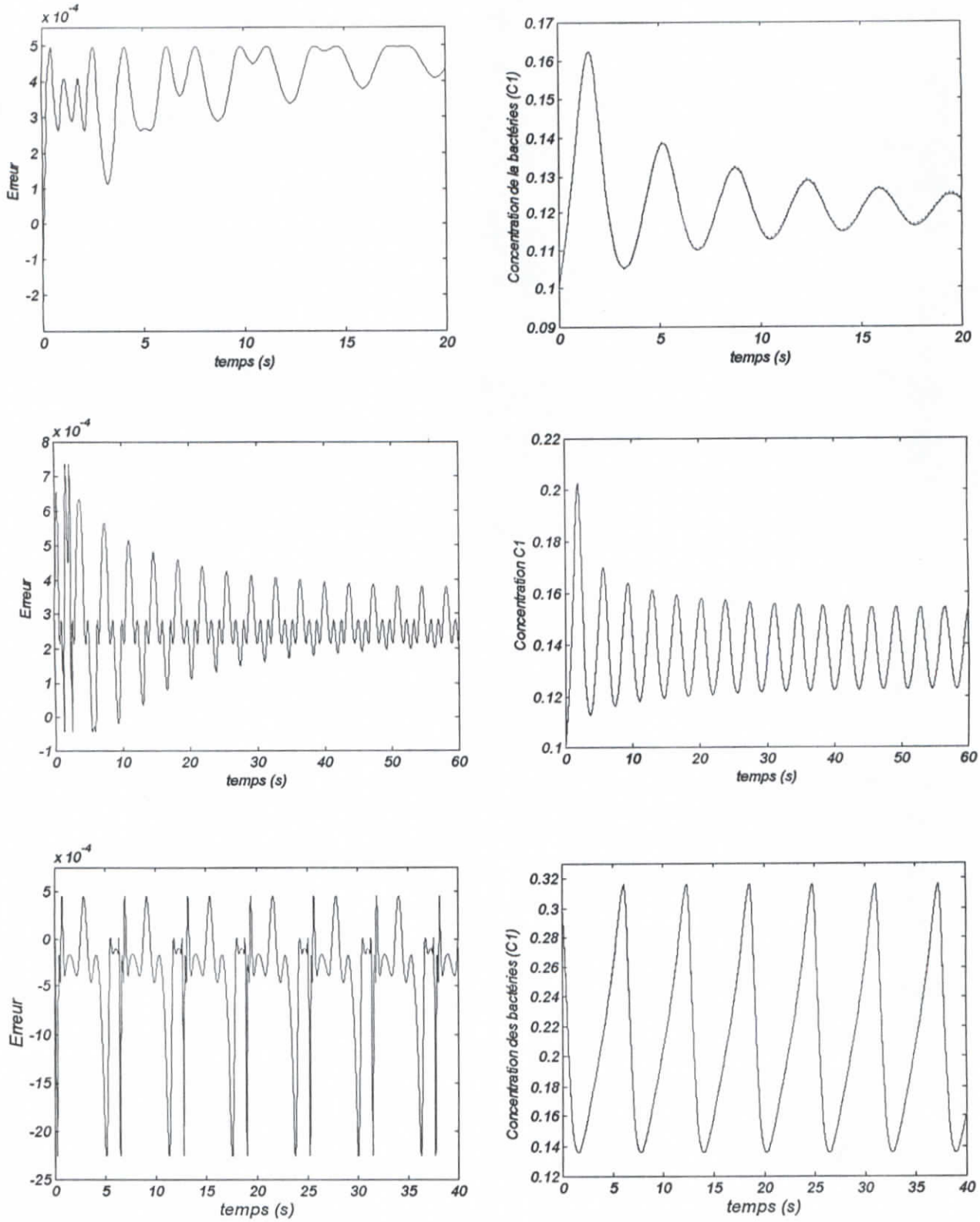
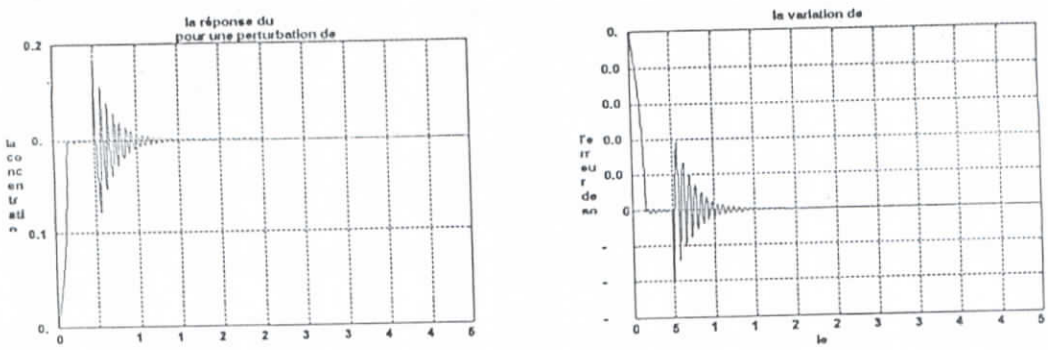
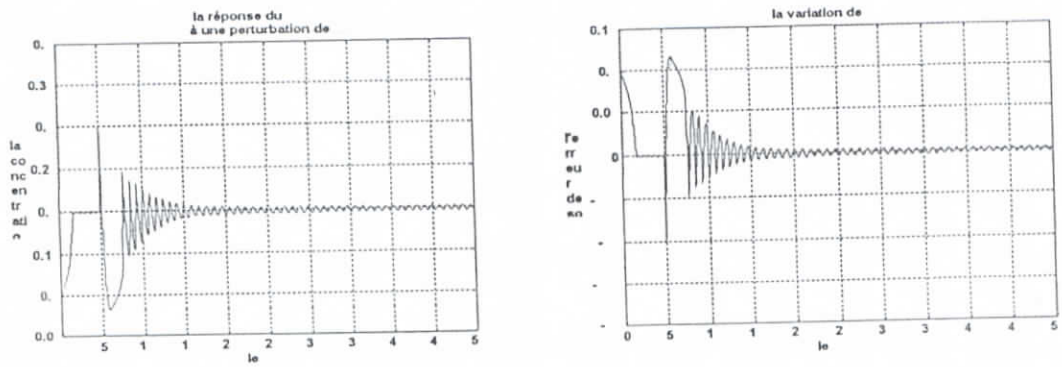


Figure III.19
Simulation du réseau de neurones dans la plage d'instabilité

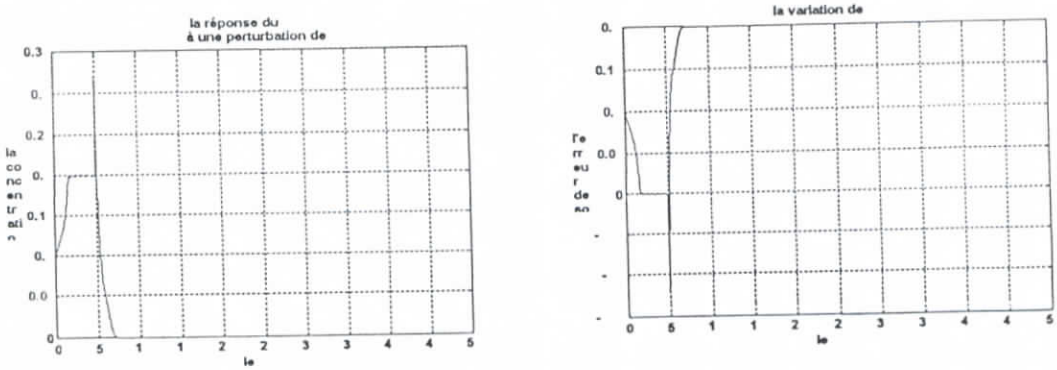
Le rejet de perturbations :



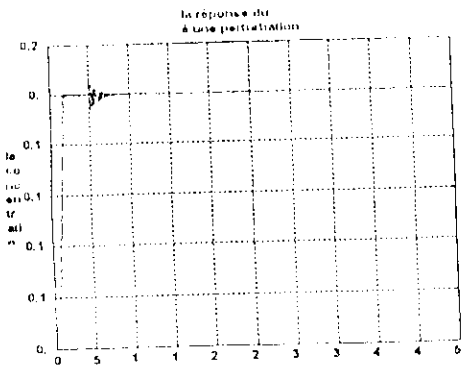
La réponse du système à une perturbation de 20%

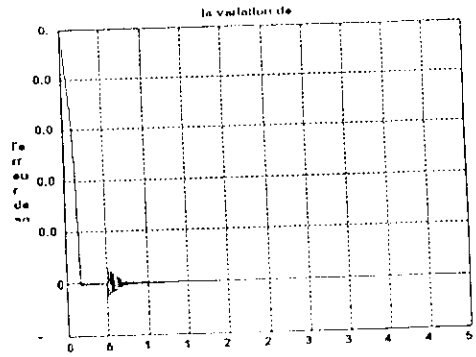


La réponse du système à une perturbation de 20%



La réponse du système à une perturbation de 70%



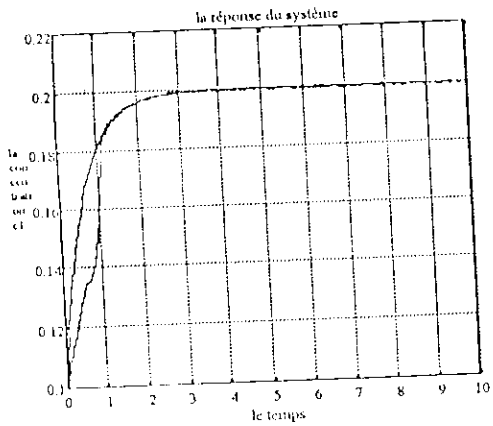


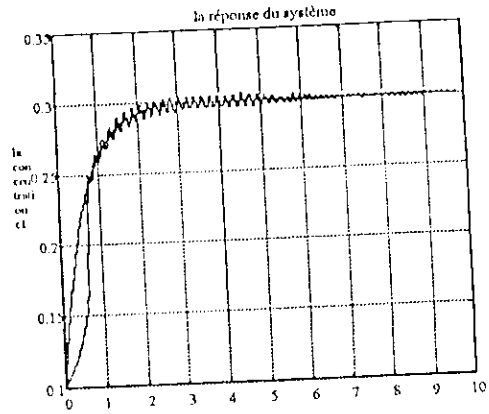
Pour un taux d'apprentissage plus grand => sensibilité aux perturbations

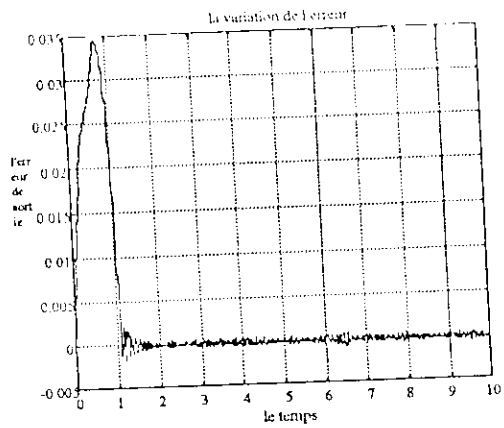
On remarque que le système peut rejeter une perturbation qui peut aller jusqu'à 50%, mais au delà le système devient instable.

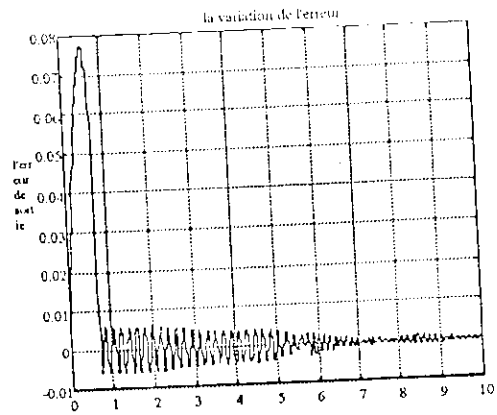
Lorsque le taux d'apprentissage est petit le système présente une robustesse vis à vis des perturbations mais lorsque le taux d'apprentissage est grand, on a une sensibilité par rapport aux perturbations qui doivent être inférieures à 4%.

Poursuite d'un modèle de référence :









On voit que le système suit la référence, après un certain temps à cause de la dynamique lente du système, mais on a une erreur accéptable.

IV. COMMANDE PAR RESEAUX DE NEURONES

IV.1 Commande inverse

La commande inverse utilise le modèle inverse du système. simulé par le RNA. Celui-ci est simplement, relié en cascade avec le système à contrôler. pour que le système global se comporte comme une fonction identité, qui lie la réponse désirée (la consigne) à la sortie du système. Le réseau est ainsi un contrôleur en boucle ouverte. Une conséquence immédiate de l'utilisation d'une commande en boucle ouverte est l'augmentation de la vitesse de réponse du système global.

Durant l'entraînement, les caractéristiques du système, initialement inconnues ou pas prises en compte, sont apprises par le réseau. De cette manière. les incertitudes de modélisation sont éliminées, et ainsi de bons résultats en commande peuvent être obtenus.
Dans ce qui suit, on présente trois méthodes d'apprentissage du neuro-contrôleur: et une version adaptée de l'algorithme de rétropropagation étendue aux problèmes dont l'erreur d'entraînement n'est pas celle mesurée à la sortie du réseau.

IV.1.1Architecture d'apprentissage générale (directe)

L'architecture montrée dans la figure II.1 fournie une méthode d'apprentissage du neuro-contrôleur qui minimise l'erreur *e*. Le réseau est entraîné pour reproduire u à partir de la sortie du système y. Une fois entraîné, le réseau doit être capable de générer la commande u à partir de la sortie désirée d, forçant la sortie du système y à suivre d.
Le succès de cette méthode dépend largement de l'aptitude du RNA à généraliser correctement sur des entrées pas nécessairement utilisées durant l'apprentissage. Dans cette architecture, on ne peut pas entraîner, sélectivement. le réseau à répondre correctement dans des régions qui nous intéressent parce qu'on ne connaît pas, généralement, quelles sont les entrées u du système qui correspondent aux sorties désirées d. Ceci étant, l'apprentissage est effectué Off-line, et une grande quantité de données d'apprentissage, non nécessaires. doivent être utilisées car les entrées, désirables et essentielles du système. Sont inconnus. Une solution possible à ce problème est de combiner l'architecture d'apprentissage générale avec l'architecture spécialisée développée ci-dessous.

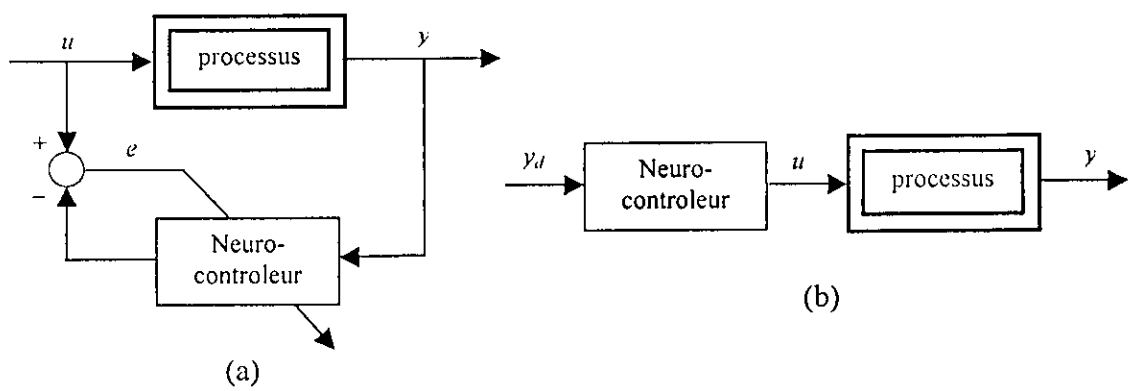


Figure IV.1
Structure d'apprentissage général.
(a) phase d'apprentissage (b) phase opérationnelle.

IV.1.2 Architecture d'apprentissage indirecte

La figure (IV.2) montre un RNA utilisé en boucle ouverte, sa sortie u commande le système. Le neuro-contrôleur agit comme le modèle inverse du système. Le but de l'apprentissage est de trouver les poids du réseau qui poussent la sortie du système y à suivre celle désirée d, alors le RNA doit reproduire la commande u donnant y. Pour cela, on adapte les poids d 'un autre réseau dans le but de minimiser l'erreur $e_1=u-t$, avec t sortie du réseau émulateur, et ainsi minimiser l'erreur $e=d-y$.
L'avantage de cette méthode et le fait que le réseau peut être entraîné seulement dans la région qui nous intéresse. puisqu'on utilise la réponse désirée d, et tous les autres signaux sont générés à partir du signal d. Malheureusement, cette méthode d'apprentissage n'est pas valide, parce que minimiser l'erreur e_1 ne veut pas dire, nécessairement minimiser l'erreur e. Cependant, cette technique reste intéressante, puisqu'elle peut être utilisée en collaboration avec l'une des procédures décrites ci-dessous: qui elles minimisent e.

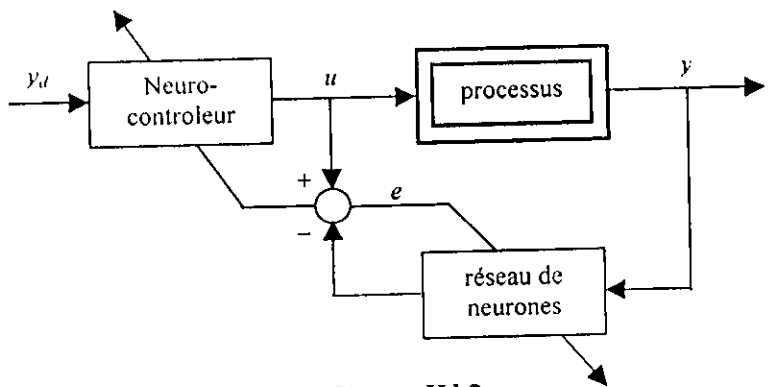


Figure IV.2
Architecture d'apprentissage indirecte.

IV.1.3 Commande par apprentissage spécialisée

Pour palier au problème sus mentionné, la figure (IV.3) présente une architecture d'apprentissage du neuro-contrôleur pour qu'il opère uniquement dans les régions de spécialisation. Le réseau est entraîné à trouver la commande u qui oblige la sortie du système y à suivre la sortie désirée d. Cette architecture peut. spécifiquement, apprendre à fonctionner dans la région d'intérêt, et le réseau peut être entraîné on-line. Intuitivement, le système à commander ne doit présenter aucun problème de stabilité dans l'intervalle de variation des données d'apprentissage. De plus, si le modèle du système est inconnu. une identification neuronale de celui-ci est nécessaire pour pouvoir entraîner le neuro-contrôleur.

Comme on l'a déjà mentionné, une combinaison entre la structure générale et la structure spécialisée, est possible. Il suffit d'entamer l'apprentissage général pour approximer le comportement inverse du système, suivi de l'apprentissage spécialisé pour affiner les poids du réseau dans la région d'opération. La méthode générale a pour rôle de créer de meilleurs poids initiaux pour la méthode spécialisée. Cette combinaison est aussi bénéfique. puisqu'elle permet au réseau de s'adapter plus facilement, si jamais l'on change l'intervalle de fonctionnement. Enfin, en basculant d'une méthode vers l'autre, on peut parfois éviter de tomber dans des minima locaux.

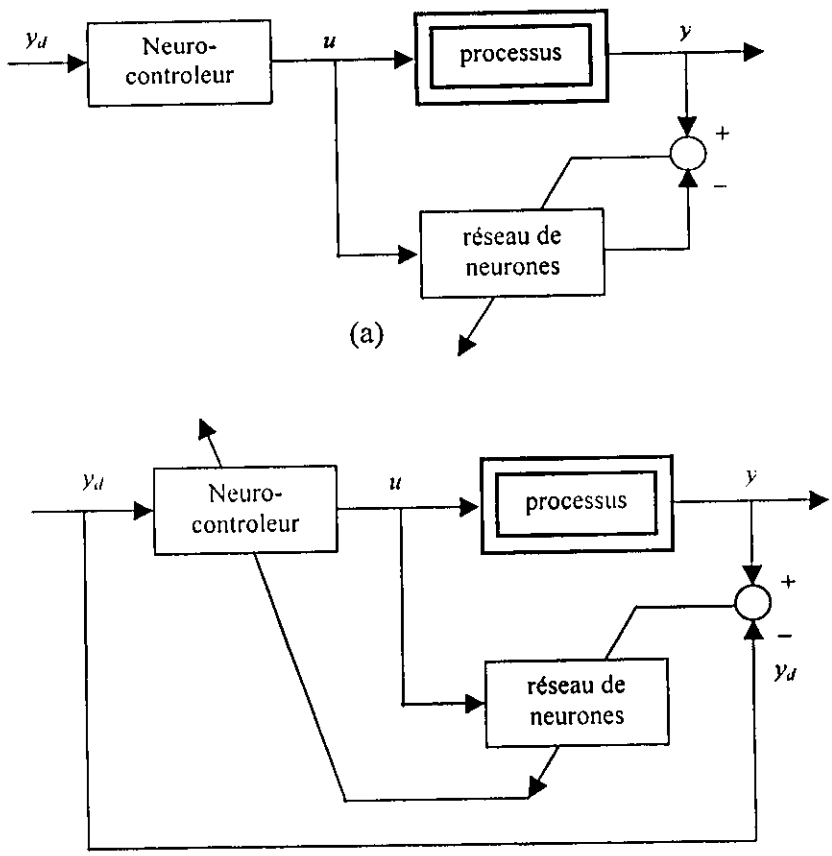


Figure IV.3
Apprentissage spécialisé avec réseau émulateur
(a) phase d'identification ; (b) phase d'apprentissage du régulateur

L'entraînement des réseaux multicouches, dans les deux méthodes précédentes, est accompli par l'algorithme de rétropropagation de base. Les poids W_{ab}, entre le neurone a de la couche m et le neurone b de la couche m-l , sont modifiés par :

$$\Delta W_{ab}^{m} = -\frac{\partial \|e\|^2}{\partial W_{ab}^{m}} \tag{IV.1}$$

$$\Delta W_{ab}^{m} = \mu \delta_a^m q_b^{m-1} \tag{IV.2}$$

avec:

$$q_i^m = f_i^m (p_i^m) \tag{IV.3}$$

et:

$$p_i^m = \sum_j W_{ij}^m q_j^{m-1} \tag{IV.4}$$

μ : est le pas du gradient, q_k est la sortie du neurone k et p_k son entrée. δ_k est l'erreur rétropropagée donnée, pour la couche de sortie n, par :

$$\delta_a^n = f'^n(p_a^n)(u_a - t_a) \tag{IV.5}$$

et Pour toutes les autres couches, par :

$$\delta_a^m = f'^m(p_a^m)\sum_i \delta_i^{m+1} W_{ia}^{m+1} \qquad \text{(IV.6)}$$

Pour ce qui concerne l'apprentissage spécialisé, on ne peut pas appliquer la rétropropagation de l'erreur directement à cause de la position du système. L'idée est de considérer le système comme une couche supplémentaire non modifiable. Alors, l'erreur $e = d - y$ est rétropropagée à travers le système en utilisant ses dérivées partielles au point de fonctionnement :

$$\delta_a^n = f'^n(p_a^n)\sum_i \delta_i^p \frac{\partial y_i}{\partial u_a} \qquad \text{(IV.7)}$$

$$\delta_a^n = d_a - y_a \qquad \text{(IV.8)}$$

avec y_i, étant la $i^{\text{ème}}$ sortie du système correspondant a l'entrée $u_.$.
Il est clair que cette procédure nécessite la connaissance du Jacobien du système. Si ceci n'est pas le cas. et qu'un RNA remplace le modèle du système, le calcul des δ_i^j, s'effectue normalement avec le réseau modèle, sans pour autant modifier ses pondérations. Une fois arriver à la couche de sortie du neuro-contrôleur. l'adaptation des poids commence à avoir lieu. La structure d'apprentissage spécialisé peut se faire aussi en imbriquant les deux structures précédentes dans une seule opération comme indiqué à la figure suivante :

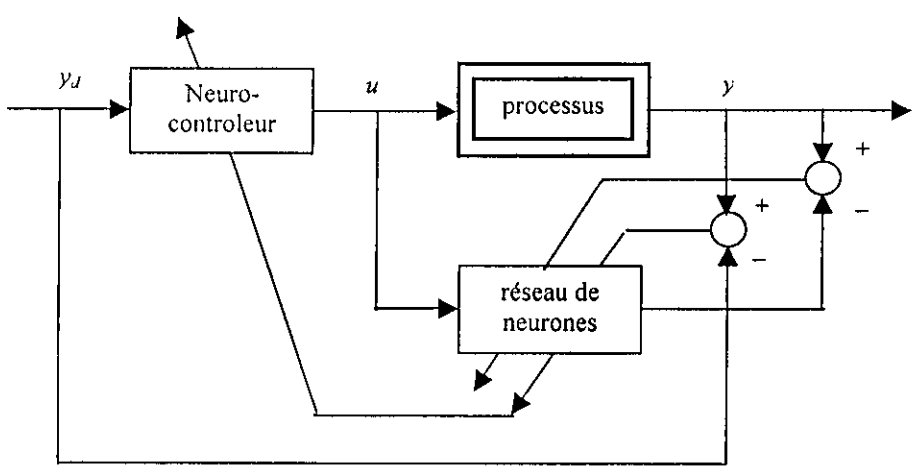


Figure IV.4
Apprentissage spécialisé avec une structure imbriquée

IV.1.4 Commande inverse dynamique

La commande inverse qu'on vient de présenter. met à notre disposition un ensemble de techniques intéressantes pour la commande des systèmes statiques. Cependant, il n'est pas clair comment ces techniques peuvent-elles être appliquées pour la commande des systèmes dynamiques ? La méthode qu'on va présenter peut être considérée comme une conception d'un contrôleur adaptatif pour les systèmes non linéaires invariants, ayant une dynamique complètement inconnue.

La conception de notre neuro-contrôleur se fait en deux étapes. premièrement, une procédure pour la modélisation de la dynamique inverse du système on-line, puis une procédure pour générer les signaux de commande. Pour cela, deux hypothèses sont posées :

H1. L'ordre du système (le nombre des états) est connu.
H2. Les états sont mesurables.

IV.1.4.1 Identification du modèle dynamique inverse du système

Considérons le système non linéaire discret d'ordre *n*, décrit par l'équation d'état suivante :

$$x(k+1)=f(x(k),u(k)) \tag{IV.9}$$

x(k) est l'état du système à l'instant k, et *u(k)* est la commande au même instant.
De même. les états à l'instant *k+2* sont tractions de *u(k+1)* et de *x(k+1)* qui est, à son tour fonction de *x(k)* et de *u(k)* :

$$x(k+2)=f(x(k+1),u(k+1))=f[f(x(k),u(k)),u(k+1)] \tag{IV.10}$$

ceci implique que les états à l'instant *k+2* sont déterminés par les états à l'instant k. et les actions de commande entre les instants k et k+2. En répétant ce raisonnement, on en déduit que les états à l'instant k+n sont déterminés par les états à l'instant k et les actions de commande entre k et k+n.

$$x(k+n)=f_n(x(k),U) \tag{IV.11}$$

$$U=[u(k),u(k+1),...,u(k+n-1)]^T \tag{IV.12}$$

On pose maintenant une troisième hypothèse sur le système :
H3. L'équation (IV.11) est uniquement inversible pour *U*. Alors *U* peut être fonction de *x(k+n)* et de *x(k)* :

$$U=g(x(k),x(k+n)) \tag{IV.13}$$

L'équation (IV.11) est une relation fondamentale représentant la dynamique inverse du système. Elle implique que les actions de commande, qui conduisent les états *x(k)* vers les états *.x (K+n)*, Sont contenues dans le vecteur *U*.

Maintenant, un RNA peut être utilisé pour approximer la fonctions g, ayant *x(k)* et *x(k+n)* comme entrées et *U* comme sortie désirée. Si la dynamique du système est inconnue, le réseau peut être entraîné on-line. avec les signaux du système pour capturer la dynamique inverse de celui-ci. Si un modèle précis du système est connu, à priori. on peut simplement entraîner le réseau off-line avec les données générées à partir du modèle.

IV.1.4.2 Commande du système par le RNA

La section précédente nous a présenté la représentation neuronale de la dynamique inverse du système nonlinéaire. On explique maintenant comment le même réseau peut être utilisé pour générer' les signaux de commande.

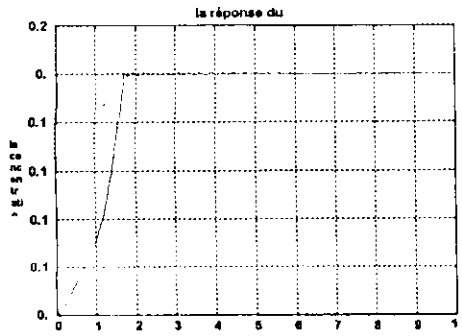
Assumons que le réseau a, complètement, appris la dynamique inverse du système de l'équation (IV.13) . Alors: donnant les mesures des états actuels *x(k)* et des états futurs désirés *Xd (k+n)*), le réseau se charge de délivrer l'action de commande *U* par :

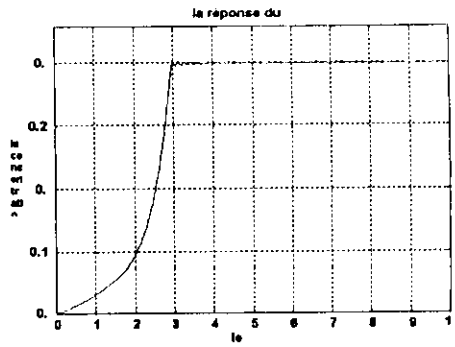
$$U=g(x(k),x_d(k+n)) \tag{IV.14}$$

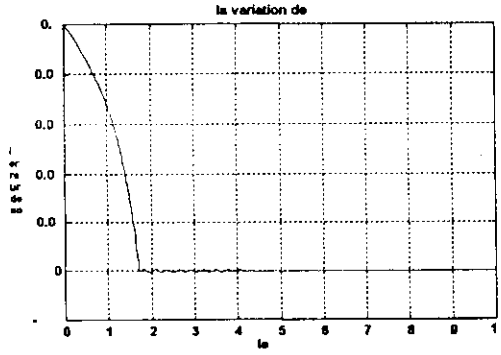
qui conduira l'état du système de *x(k)* à $x_d(1+n)$ en, exactement, *n* étapes.

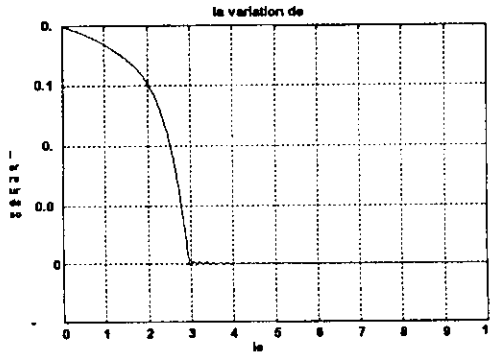
Le réseau utilisé pour la modélisation et la commande, possède 2n entrées et n sorties. Une architecture plausible pour ce réseau est de trois couches. avec 2n neurones dans la couche d'entrée, un certain nombre de neurones ayant des fonctions d'activation sigmoïdes, dans la couche cachée, et n neurones linéaires dans la couche de sortie. Les neurones à fonctions d'activation sigmoïdes sont utilisées pour capturer la nonlinéarité du système, et les neurones à fonctions d'activations linéaires sont utilisées pour permettre une approximation de U dans des intervalles arbitraires.

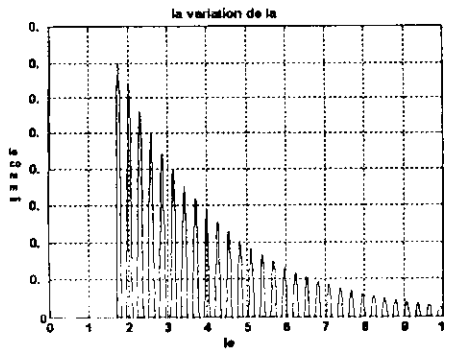
La réponse du système à un échelon :

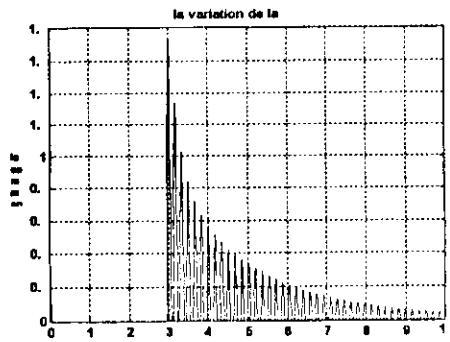








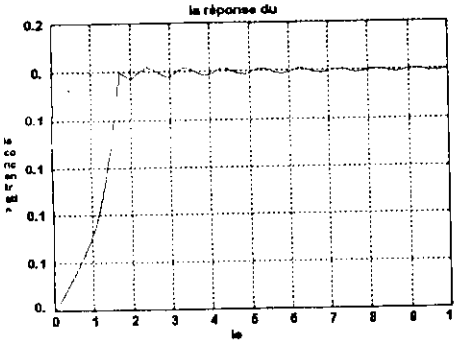


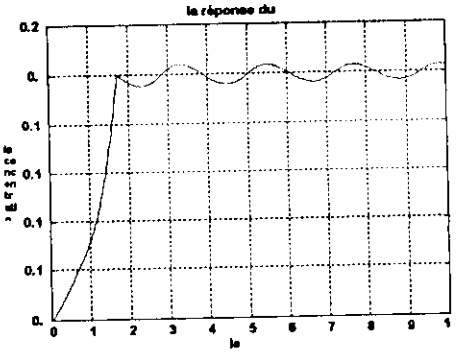


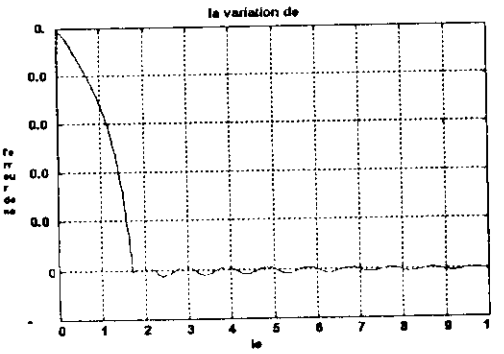
La réponse à un échelon de 0.2
Avec C_1=0.1, C_2=0.8

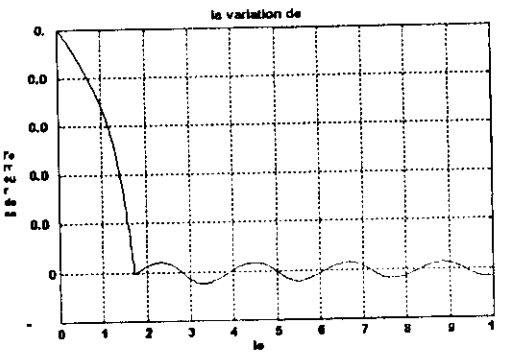
La réponse à un échelon de 0.3
Avec C_1=0.1, C_2=0.8

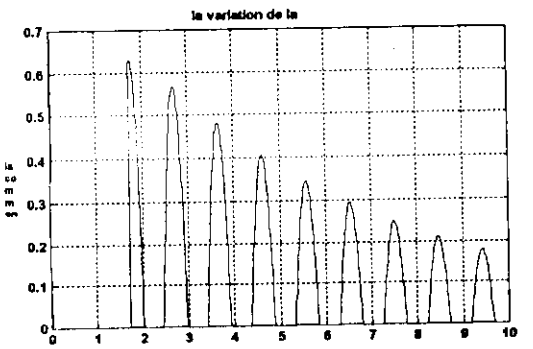
On voit que le système converge vers la consigne, ensuite il oscille autour de la consigne avec une erreur acceptable, mais la commande est rapide, qui peut dépasser les limites de l'électrovanne chargée du contrôle du débit.

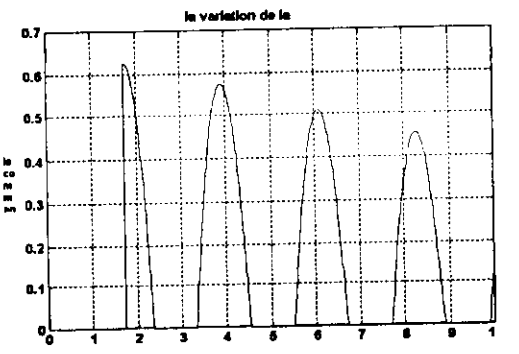












Apprensissage avec un taux =0.1

apprentissge avec un taux =0.01

L'influence du taux d'apprentissage

On remarque que le taux d'apprentissage a une grande influence sur la qualité du réglage, cependant si le taux d'apprentissage est grand on a une commande rapide et une erreur plus petite en comparaison avec un taux d'apprentissage plus petit, d'où un compromis qui s'installe, entre rapidité de la commande et l'erreur en sortie.

IV.2 Régulateur auto-ajustable par réseau de neurones

Le régulateur auto-ajustable est essentiellement un retour classique avec des paramètres ajustés en temps réel, ce régulateur est composé de deux boucles, la boucle interne est constituée du système à commander et d'un régulateur ordinaire, la boucle externe contient l'algorithme d'adaptation qui permet d'ajuster les paramètres du régulateurs [Ast1987], elle est applicable aux systèmes non linéaires qui possèdent la forme :

$$y(k+1)=f(y(k), y(k-1), ..., y(k-p), u(k-1), u(k-2), ..., u(k-p))+$$

$$g(y(k), y(k-1), ..., y(k-p), u(k-1), u(k-2), ..., u(k-p))\, u(k) \qquad \text{(IV.15)}$$

si les fonctions $f(.)$ et $g(.)$ sont connues, alors pour que la sortie du processus à commander $y(k+1)$ soit égale à la sortie désirée $d(k+1)$, on doit avoir :

$$u(k)=-\frac{f(.)}{g(.)}+\frac{d(k+1)}{g(.)} \qquad \text{(IV.16)}$$

Si les fonctions $f(.)$ et $g(.)$ sont inconnues, un réseau de neurones peut être utilisé pour approximer ces fonctions et pour générer la commande nécessaire.
Dans ce qui suit on considère le système peut être mis sous la forme (IV.15)

$$y(k+1) = f(y(k)) + g(y(k))\, u(k) \qquad \text{(IV.17)}$$

On suppose aussi que malgré que les fonction $f(.)$ et $g(.)$ sont inconnues, le signe de la fonction $g(.)$ est connue. Le système peut être modélisé par un réseau de neurones montré par la figure IV.5, et possédant la forme :

$$\hat{y}(k+1)= \hat{f}(y(k), y(k-1), ..., y(k-p), u(k-1), u(k-2), ..., u(k-p), W(k))+$$

$$\hat{g}(y(k), y(k-1), ..., y(k-p), u(k-1), u(k-2), ..., u(k-p), V(k))\, u(k) \qquad \text{(IV.18)}$$

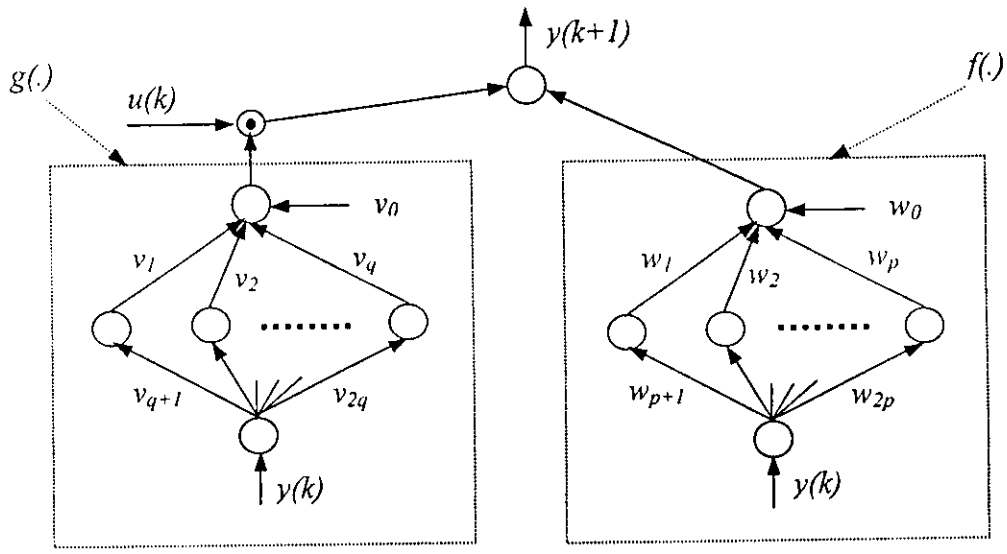


Figure IV.5
Représentation du système non linéaire par un réseau de neurones

L'objectif de commande est de ramener la sortie du système à la sortie désirée *d(k+1)*, le schéma de la commande auto-ajustable est montré à la figure IV.6 . en effet, la commande à appliquer à l'instant k est sous la forme suivante :

$$u(k)=-\frac{\hat{f}[y(k),W(k)]}{\hat{g}[y(k),V(k)]}+\frac{d(k+1)}{\hat{g}[y(k),V(k)]} \qquad \text{(IV.19)}$$

et en injectant cette équation dans IV.16, nous obtenons le système bouclé qui possédera la forme :

$$y(k+1)=f(y(k)) + g(y(k))\left[-\frac{\hat{f}[y(k),W(k)]}{\hat{g}[y(k),V(k)]}+\frac{d(k+1)}{\hat{g}[y(k),V(k)]}\right] \qquad \text{(IV.20)}$$

Ainsi, il ne reste plus qu'à minimiser le critère :

$$E(k)=\frac{1}{2}e(k)^2=\frac{1}{2}(d(k+1)-y(k+1))^2 \qquad \text{(IV.21)}$$

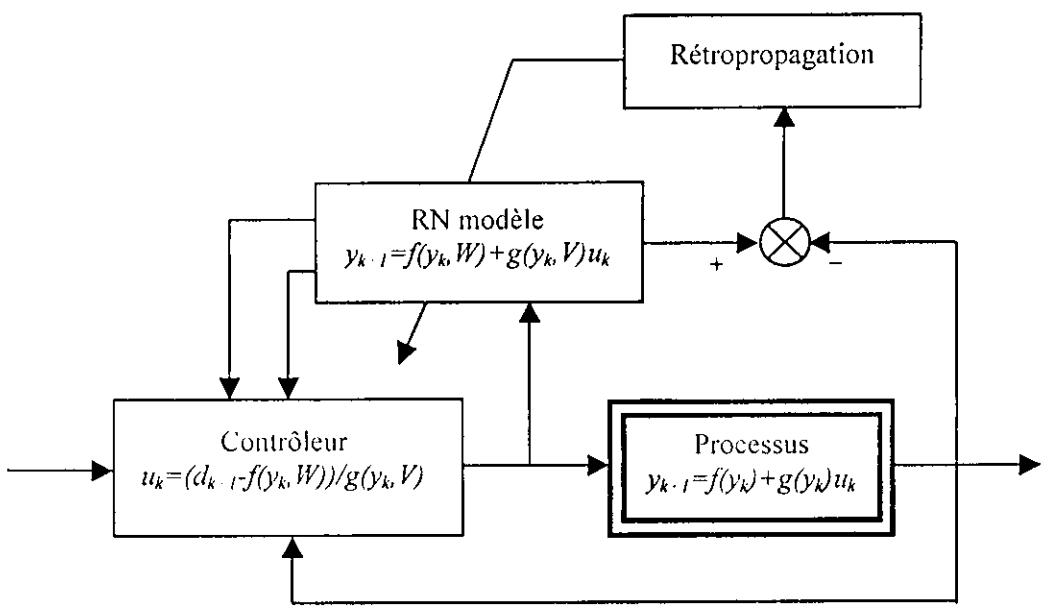


Figure IV.6
Structure de la commande auto-ajustable
Par réseau de neurones

Les paramètres du réseau neuronal (les matrices des poids W et V sont ajustées de telle façon à minimiser le critère *E(k)*, pour cela il faut vérifier :

$$\frac{\partial E(k)}{\partial w_i(k)}=\frac{g(y(k))}{\hat{g}[y(k)\ ,\ V(k)]}\cdot\left\{\frac{\partial\hat{f}[y(k)\ ,\ W(k)]}{\partial w_i(k)}\right\}e(k+1) \qquad \text{(IV.22)}$$

Et :

$$\frac{\partial E(k)}{\partial v_j(k)} = \frac{g(y(k))}{\hat{g}[y(k)\ ,\ V(k)]} \cdot \left\{ \frac{\partial \hat{g}[y(k)\ ,\ V(k)]}{\partial v_j(k)} \right\} u(k)\, e(k+1) \qquad \text{(IV.23)}$$

les quantités $\frac{\partial \hat{f}[y(k)\ ,\ W(k)]}{\partial w_i(k)}$ et $\frac{\partial \hat{g}[y(k)\ ,\ V(k)]}{\partial v_j(k)}$ peut se calculer on utilisant une méthode similaire à celle de la rétropropagation du gradient, la fonction *g(y(k))* dans les équations 22, 23 est inconnue, mais sont signe est supposé connu, dans ce cas la valeur de *g(y(k))* est remplacée par le *sgn[g(y(k))]* dans les équations 22, 23, les règles d'adaptation des matrices W et V sont donc :

$$w_i(k+1) = w_i(k) - \eta_k \frac{\operatorname{sgn}[g(y(k))]}{\hat{g}[y(k)\ ,\ V(k)]} \cdot \left\{ \frac{\partial \hat{f}[y(k)\ ,\ W(k)]}{\partial w_i(k)} \right\} e(k+1) \qquad \text{(IV.24)}$$

$$v_i(k+1) = v_i(k) - \mu_k \frac{\operatorname{sgn}[g(y(k))]}{\hat{g}[y(k)\ ,\ V(k)]} \cdot \left\{ \frac{\partial \hat{g}[y(k)\ ,\ V(k)]}{\partial v_i(k)} \right\} u(k)\, e(k+1) \qquad \text{(IV.25)}$$

Où η_k et μ_k sont les pas d'adaptation à l'instant k, ils sont choisit de la manière suivante :

$$\eta_k = \mu_k \frac{\sum_j \left\{ \partial \hat{g}[y(k)\ ,\ V(k)] / \partial v_j(k) \right\}^2}{\sum_i \left\{ \partial \hat{f}[y(k)\ ,\ W(k)] / \partial w_i(k) \right\}^2} \qquad \text{(IV.26)}$$

Les fonctions $\hat{f}[y(k+1)\ ,\ W(k+1)]$ et $\hat{g}[y(k+1)\ ,\ V(k+1)]$ sont calculer par un réseau de neurones, puis $\hat{f}[y(k+1)\ ,\ W(k+1)]$, $\hat{g}[y(k+1)\ ,\ V(k+1)]$ et *d(k+2)* sont utiliser pour générer la commande *u(k+1)* avec IV.19.

Application

Dans cette partie, la commande auto ajustable a été appliquée au bioréacteur présenter au premier chapitre. Le réseau contrôleur utilisé est composé de deux réseaux de neurones, le premier réseau est mono entrée mono sortie possédant une seule couche cachée de 15 neurones avec une fonctions d'activations tangente hyperbolique, le deuxième réseau est réduit à un seul neurone avec un seul poids à adapter (car dans ce cas la fonction *g(.)* est linéaire par rapport à *C1*), et la structure du réseau complet est celle présenter à la figure IV.5. L'entraînement des deux réseaux se fait par l'algorithme de rétropropagation avec un pas variable (technique présenter au chapitre I).

Afin de tester la commande on présente un signal de référence défini par :

$$r(t) = 0.07\left[\sin(2\pi.0.04t) + \sin(2\pi.0.04t)\right] \qquad \text{(IV.27)}$$

La figure suivante, montre les résultats de la commande. On remarque que le système suit la référence et il est robuste au perturbations externes.

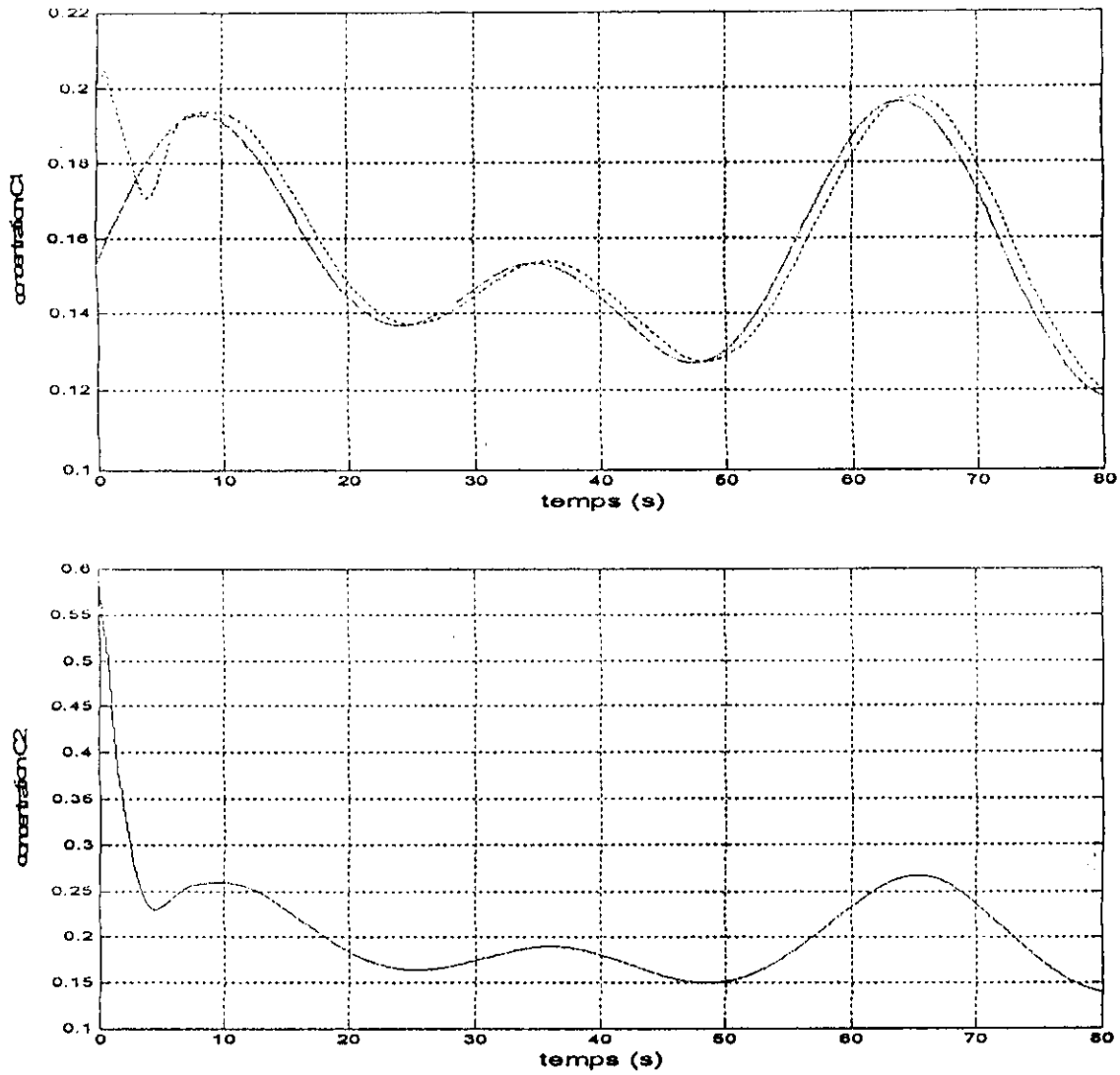


Figure IV.7
Réponse du système (----), Référence(——)
Pour C1(0)=0.2, C2(0)=0.6

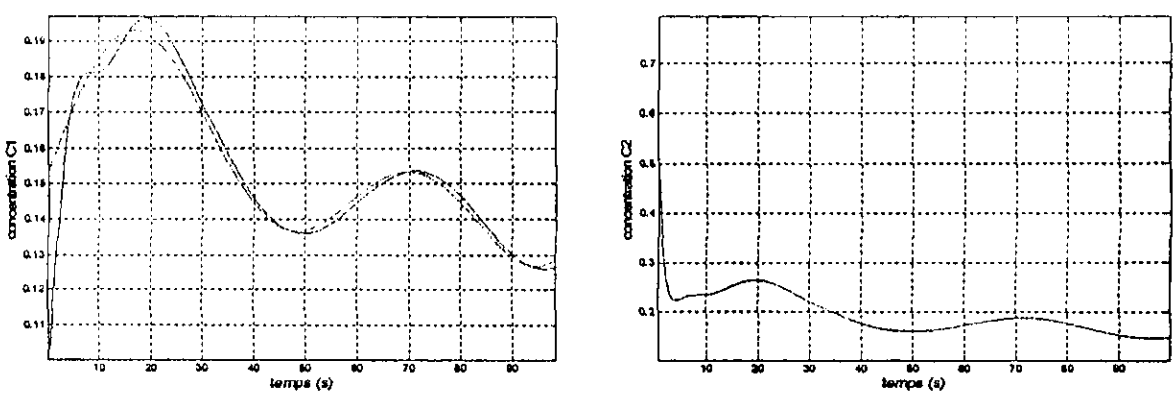


Figure IV.8
Réponse du système pour les condition initiales C1(0)=0.1, C2(0)=0.6.

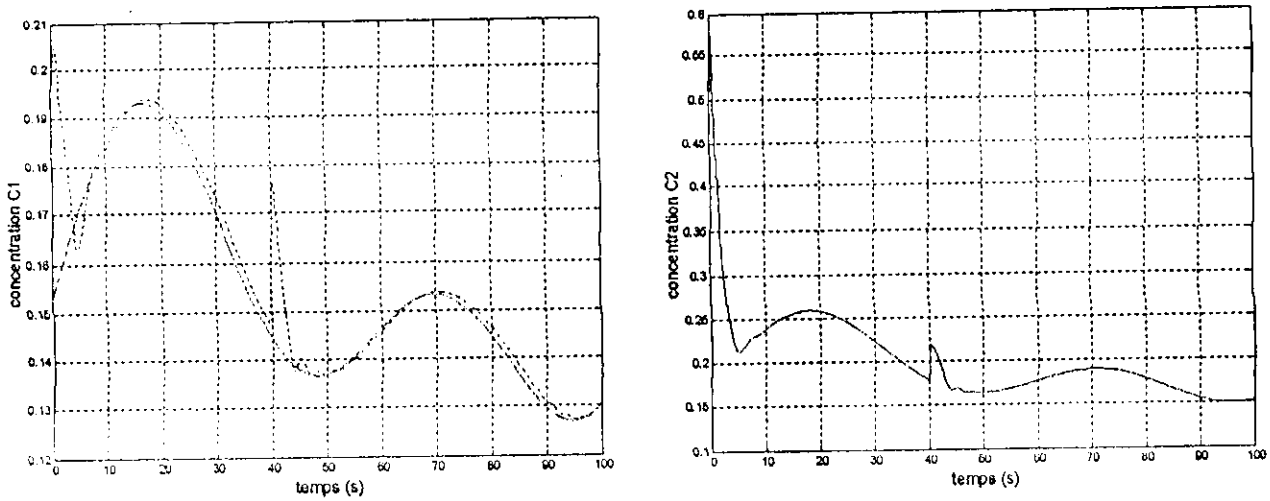


Figure IV.9
Réponse du système (----), Référence(——)
Avec une perturbation de 40% à t=40 s
Pour C1(0)=0.2, C2(0)=0.6.

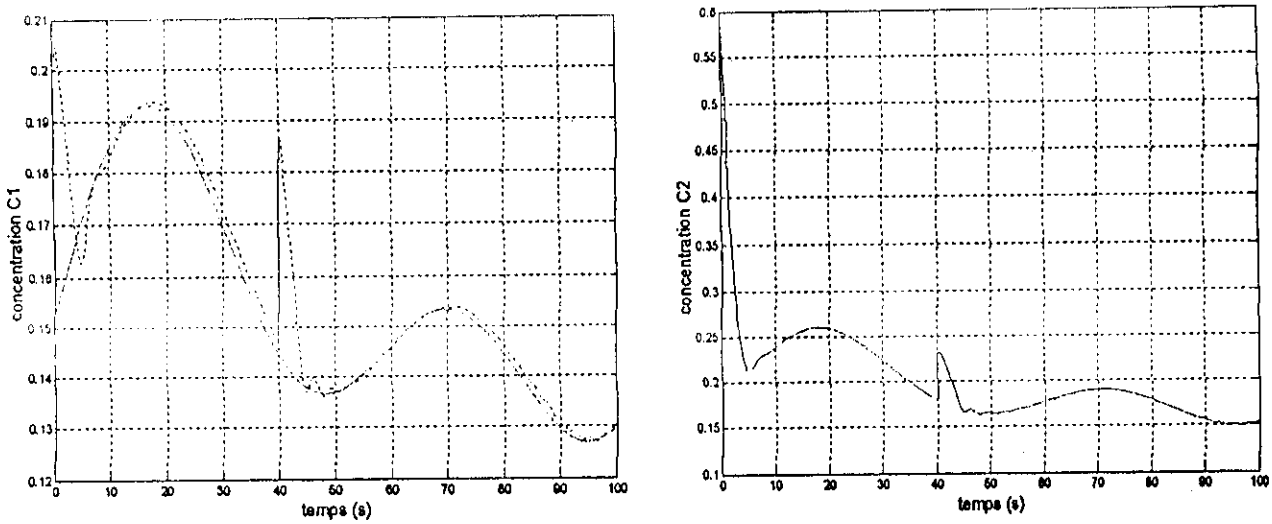


Figure IV.9
Réponse du système (----), Référence(——)
Avec une perturbation de 60% à t=40 s
Pour C1(0)=0.2, C2(0)=0.6.

L'objectif de commande est de ramener la sortie du système à la sortie désirée $d(k+1)$, le schéma de la commande auto-ajustable est montré à la figure IV.6 . en effet, la commande à appliquer à l'instant k est sous la forme suivante :

$$u(k)=-\frac{\hat{f}[y(k),W(k)]}{\hat{g}[y(k),V(k)]}+\frac{d(k+1)}{\hat{g}[y(k),V(k)]} \qquad \text{(IV.19)}$$

et en injectant cette équation dans IV.16, nous obtenons le système bouclé qui possédera la forme :

$$y(k+1)=f(y(k)) + g(y(k))\left[-\frac{\hat{f}[y(k),W(k)]}{\hat{g}[y(k),V(k)]}+\frac{d(k+1)}{\hat{g}[y(k),V(k)]}\right] \qquad \text{(IV.20)}$$

Ainsi, il ne reste plus qu'à minimiser le critère :

$$E(k)=\frac{1}{2}e(k)^2=\frac{1}{2}(d(k+1)-y(k+1))^2 \qquad \text{(IV.21)}$$

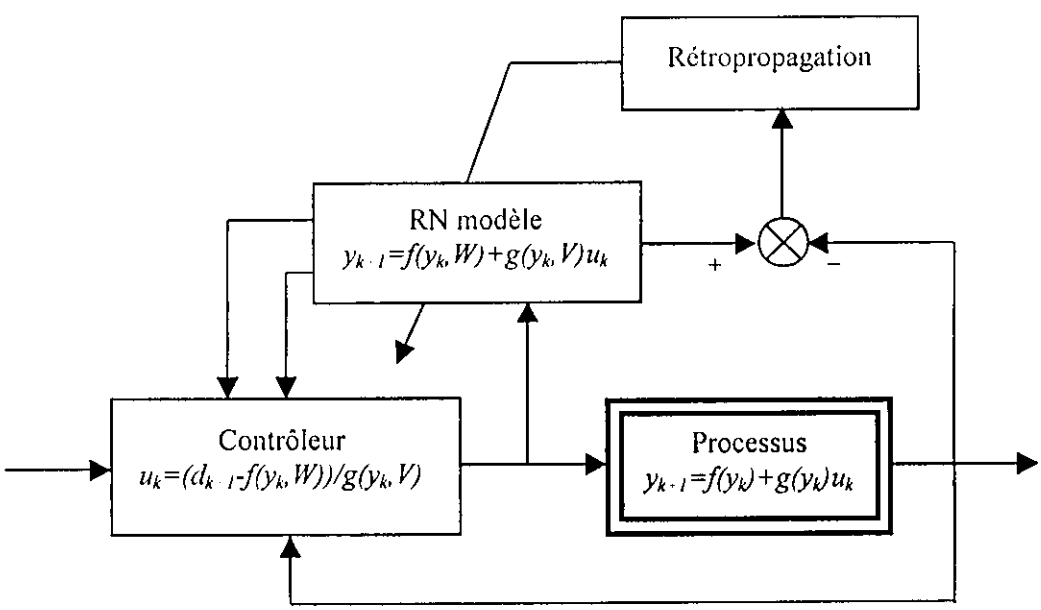


Figure IV.6
Structure de la commande auto-ajustable
Par réseau de neurones

Les paramètres du réseau neuronal (les matrices des poids W et V sont ajustées de telle façon à minimiser le critère $E(k)$, pour cela il faut vérifier :

$$\frac{\partial E(k)}{\partial w_i(k)}=\frac{g(y(k))}{\hat{g}[y(k)\ ,\ V(k)]}\cdot\left\{\frac{\partial\hat{f}[y(k)\ ,\ W(k)]}{\partial w_i(k)}\right\}e(k+1) \qquad \text{(IV.22)}$$

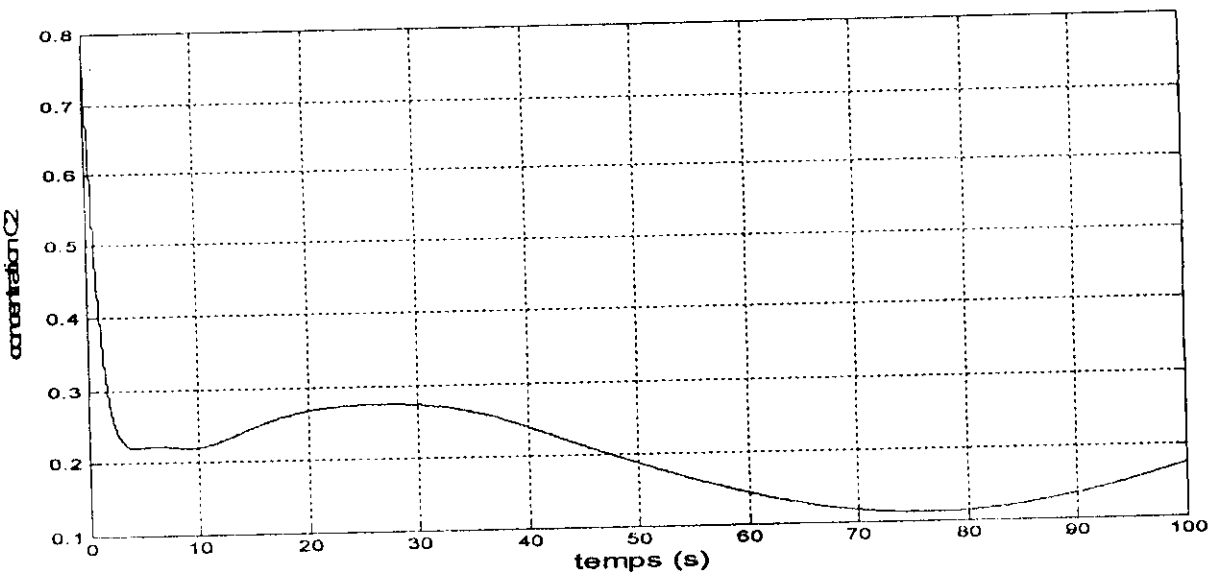


Figure 20
Sortie C2 pour une référence sinusoidale
avec C1(0)=0.1, C2(0)=0.8

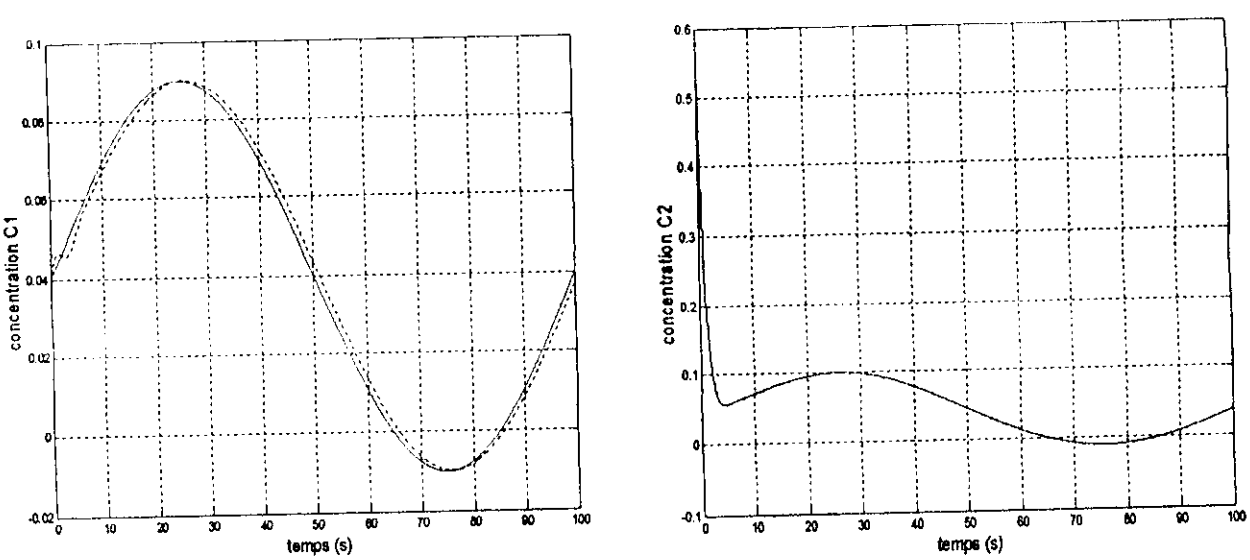


Figure 21
Réponse du système à une référence sinusoïdale
Avec les conditions initiales C1(0)=0.04, C2(0)=0.45.

IV.4 Commande adaptative à modèle de référence

Dans cette stratégie l'optimisation du critère à minimiser se fait par rapport aux poids du réseau, la commande a pour rôle d'obliger le système a suivre un modèle de référence défini. Pour une commande par réseaux de neurones, ce modèle est défini par des couples entrées – sorties.

Les réseaux de neurones ont les capacités d'approximation suffisantes qui leur permettent d'approcher la loi de commande nécessaire (si elle existe) pour la poursuite du modèle de référence.

Comme dans le cas de la commande adaptative classique, deux approches peuvent être utilisées :

IV.4.1 Commande adaptative indirecte :

Dans ce cas, les paramètres du système sont estimés à chaque instant. Les paramètres du régulateur sont calculés en supposant que les paramètres du modèle estimé coïncidents avec ceux du système réel (fig.IV.7). C'est ce qui est appelé en commande adaptative classique le principe d'équivalence certaine.

La réussite d'une telle commande dépend étroitement de la précision avec laquelle le réseau de neurones arrive à estimer les paramètres du système.

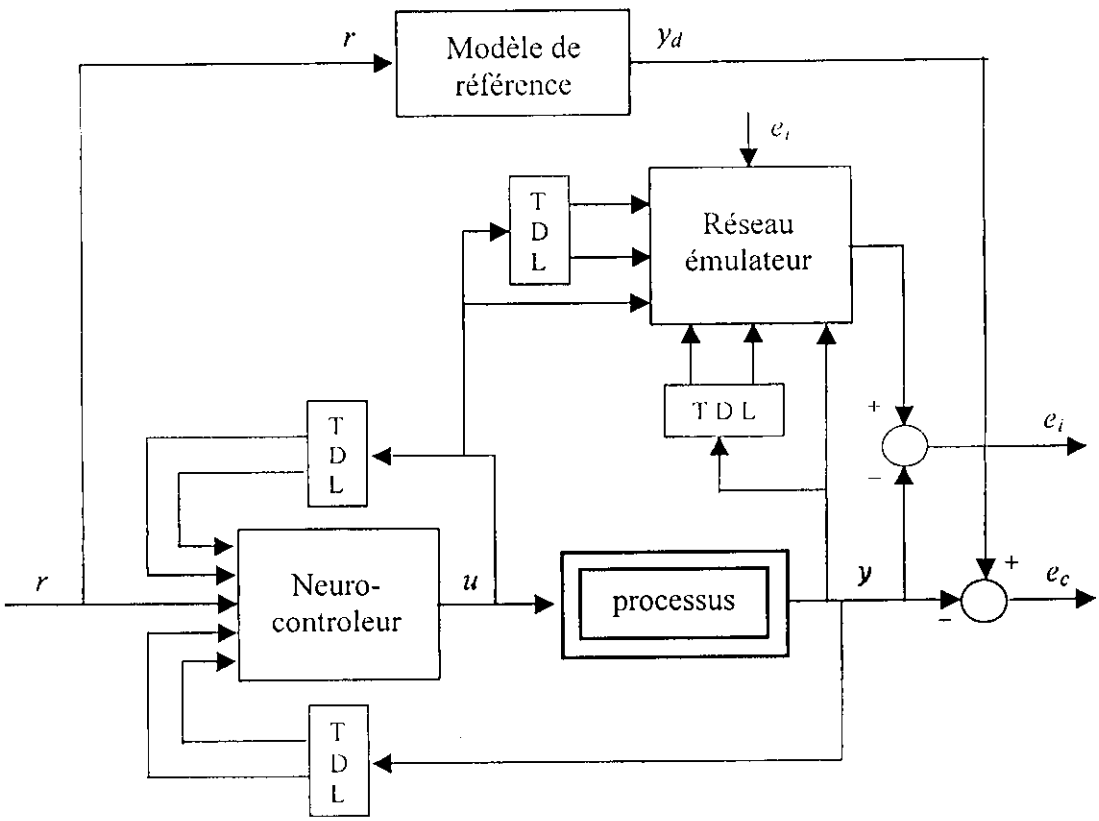


Figure IV.7
Architecture de commande adaptative neuronale indirecte

IV.4.2 Commande adaptative directe :

En commande adaptative directe, les paramètres du réseau contrôleur sont directement adaptés, en minimisant l'erreur en sortie entre le modèle de référence et le système (figure IV.8).

La vérification de la stabilité de cette structure est basée, comme dans le cas de la commande adaptative directe classique, sur la théorie d'Hyperstabilité et le formalisme de la fonction de Lyapunov. Cependant, dans le contexte neuronal, deux conditions importantes doivent être vérifiées :

- Les poids initiaux du réseau ne doivent pas être éloignés des valeurs qui donnent lieu à une poursuite parfaite du modèle de référence.

- La vitesse d'adaptation des poids du réseau, qui dépend du taux d'apprentissage, doit être suffisamment faible, afin de permettre au réseau de pouvoir séparer dans les erreurs mesurées, l'effet des ajustements des poids de l'effet des variations des signaux d'entrée.

Mais le problème qui revient ici, et que nous avons déjà évoqué dans le cas de la commande spécialisée, est les difficultés que peut poser l'entraînement du réseau. En effet

l'erreur à minimiser est à la sortie du système, et les valeurs désirées à la sortie du réseau sont inconnues.

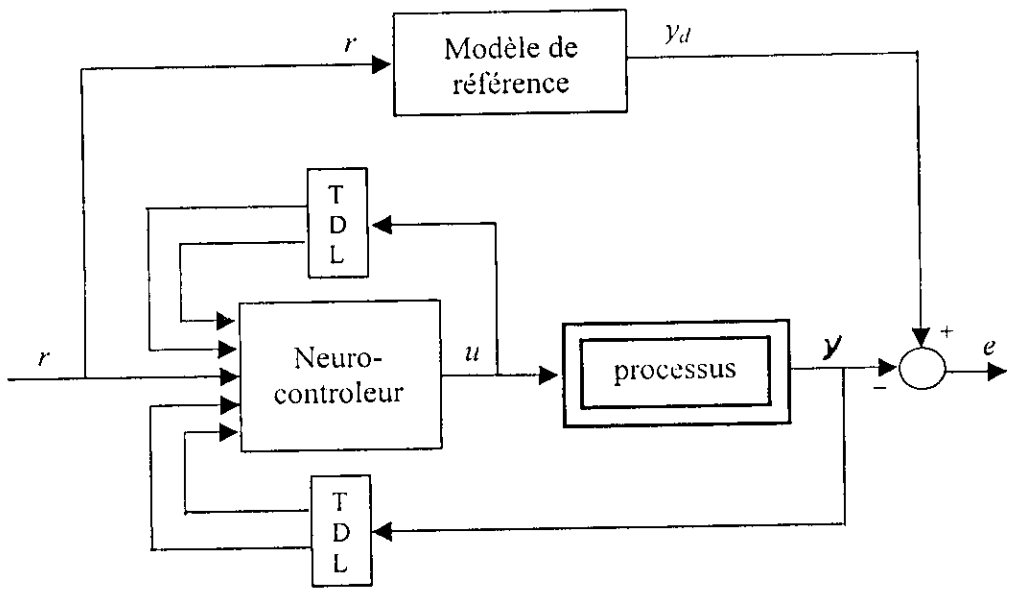


Figure IV.8
Architecture de commande adaptative
avec modèle de référence et régulateur neuronal

IV.4.2.1 Apprentissage du réseaux controleurs

Le problème d'apprentissage des réseaux de neurones en commande est lié aux sorties désirées que les réseaux controleurs doivent délivrer.

Si ces sorties sont connues, comme dans le cas supervisé, où la loi de commande est connue, ou dans le cas d'un apprentissage général, où l'entrainement se fait hors ligne, ce problème se trouve résolu. La réadaptation des poids se fait en minimisant l'erreur à la sortie du réseau.

Cependant, dans la plus part des cas, les sorties que doit délivrer le réseau sont inconnues. C'est, justement, dans ces cas que l'utilisation des réseaux de neurones est la plus intéressante ; notamment dans celui de la commande adaptative ou, plus généralement encore, la commande à base d'un apprentissage généralisé.

Le problème qui intervient est la rétropropagation de l'erreur. En effet le système, supposé inconnu, est placé entre le réseau de neurone et l'erreur. Celui ci **bloque** l'accès du réseau de neurone vers cette erreur. D'un point de vue théorique, le problème se situe à la réadaptation des poids, qui nécessite le calcul du gradient de la sortie du système par rapport à la matrice des poids.

La solution la plus triviale consiste à calculer la chaine de dérivées partielles jusqu'à aboutissement au réseau. Dans ce cas le calcul du jacobien du système est donc nécessaire. On peut concevoir un réseau avec un algorithme d'apprentissage en utilisant la méthode de la descente de gradient pour ajuster les poids du réseau controleur. Une des solutions possibles est l'utilisation d'un réseau neuronal dont les entrées sont constituées du vecteur :

$H_k=[y(k),y(k-1),\ldots,y(k-p+1),u(k-1),u(k-2),\ldots,u(k-p+1)]$, ainsi que la référence $r(k)$.

A chaque instant k, le régulateur neuronal tentera de réduire en ligne l'erreur $e(k)=1/2(y(k)-y_d(k))^2$ en exécutant une descente de gradient dans l'espace des poids. Nous aurons donc :

$$w_{k+1}=w_k-\eta\frac{\partial e(k)}{\partial w} \tag{IV.29}$$

avec :

$$\frac{\partial e(k)}{\partial w}=(y_r(k)-y(k))\frac{\partial y(k)}{\partial w} \tag{IV.30}$$

et

$$\frac{\partial y(k)}{\partial w}=\frac{\partial y(k)}{\partial u}\frac{\partial u}{\partial w} \tag{IV.31}$$

la quantité $\frac{\partial y(k)}{\partial w}$ est définie comme le gradient de $y(k)$ par rapport aux poids et $\frac{\partial y(k)}{\partial u}$ est le jacobien du processus.

si nos connaissances sur le système le permettent, ce calcul peut être fait. D.Psaltis considère le système comme une couche supplémentaire, faisant partie du réseau, dont les paramètres sont figés, et ne peuvent pas être modifiés. Ainsi, l'algorithme de backpropagation peut être appliqué, sans toutefois réadapter ses poids.

Si nos connaissances sur le système sont insuffisantes, l'obtention d'expression analytique du jacobien est impossible. J.Soquet proposent, dans ce cas, de se contenter du signe du jacobien pour l'entrainement du réseau. En effet, cette information est souvent facile à obtenir à apartir de connaissances qualitatives sur le système, et nous guide sur le sens de variation de la sortie. Mais l'entrainement risque d'être difficile , du fait que le

réseau n'est pas très bien informé sur l'évolution de l'erreur à la sortie du système ; le signe peut, en effet, être insuffisant.

Dans le cas où le système est complètement inconnu, la première solution est apportée par D.Psaltis. Celui ci propose d'approcher ce calcul, en provoquant des petites perturbations locales δu aux entrée u injectées au système, et relever à chaque étape les variations δy à la sortie. Ainsi, on aura dans la chaine de différentiations, qui nous mène vers la sortie du réseau controleur dans l'équation (**) :

$$\frac{\partial y}{\partial u} \approx \frac{y(u+\delta u)-y(u)}{\delta u} \qquad \text{(IV.32)}$$

Enfin, la solution, qui peut faire l'unanimité, est proposée par Nguyen et Widrow. Elle consiste à utiliser un réseau de neurone entrainé à modéliser le système par les procédés d'identification, que nous avons déjà étudiés. Ce réseau, appelé **réseau émulateur**, est utilisé comme un **canal** à travers duquel l'erreur est acheminée, pour parvenir au réseau controleur. Ce réseau sera excité par les mémes entrées que le système et c'est l'erreur en sa sortie qui sera minimisée. Si le réseau identifie le système avec précision, Narendra propose de calculer les dérivées directement à travers ses couches.

Le jacobien est approximé en calculant les dérivées partielles du réseau identificateur, comme dans le cas de la méthode de Rétropropagation classique. La rétro-propagation est appliquée sans, toutefois, modifier les poids du réseau émulateur.

G.Lightbody a utilisé la technique du réseau émulateur pour la fonction de sensibilité système par rapport à l'entrée lui provenant du réseau contrôleur . Pour cela, il propose une méthode numérique, évitant de calculer des dérivées, et permettant une procédure de calcul itératif plus souple.

Toutes les solutions que nous avons rapportées ici reposent sur la nécessité de calculer le jacobien du système du fait de l'utilisation de l'algorithme de Rétropropagation pour l'entraînement du réseau contrôleur.

Appliacation

La commande adaptative a été appliquée sur le bioréacteur, la synthèse de la loi de commande adaptative passe par deux phases principales :

a. Phases d'identification

comme on a expliqué précédemment, la synthèse de la loi de commande nécessite la connaissance du système et plus particulièrement sont jacobien qui est utilisé dans la formule d'adaptation du réseaux contrôleur.

La phase d'identification du système par un émulateur s'effectue Hors-ligne, le réseau utilisé pour cette identification est un réseau à base radiale normalisé généralisé comportant 25 neurones dans sa couche cachée, l'algorithme d'adaptation utilisé est la rétropropagation du gradient appliquée aux réseaux RBF avec des pas variables pour l'adaptation des poids, centres et écart type.

Ce type de réseaux est choisi grâce à sa robustesse dans le calcul du jacobien, qui nécessite le calcul de la dérivée du critère J à minimiser par rapport à l'entrée u du réseau (rétropropagation étendue aux entrées), cette dérivée s'obtient par la relation :

$$\frac{\partial J}{\partial u_k} = \frac{\partial J}{\partial \rho_j} \frac{\partial \rho_j}{\partial u_k}$$

$$\frac{\partial \rho_j}{\partial u_k} = -\rho_j \cdot \frac{u_k - c_{jk}}{\sigma_{jk}^2}$$

les résultats de simulation sont représentés par les figures suivantes :

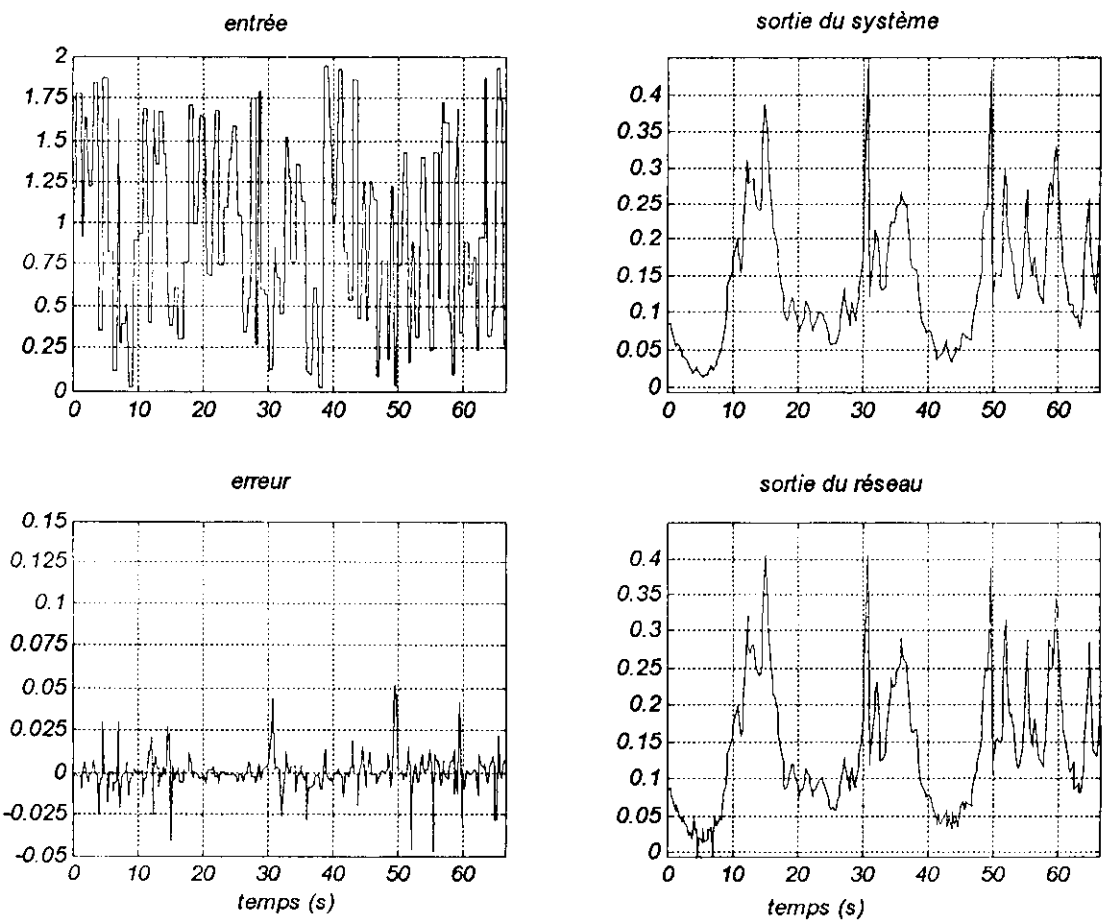


Figure IV.13
Représentation des résultats d'apprentissage de l'émulateur

Un teste de généralisation à été effectué afin de tester la robustesse du réseaux et on a obtenu les résultats suivants :

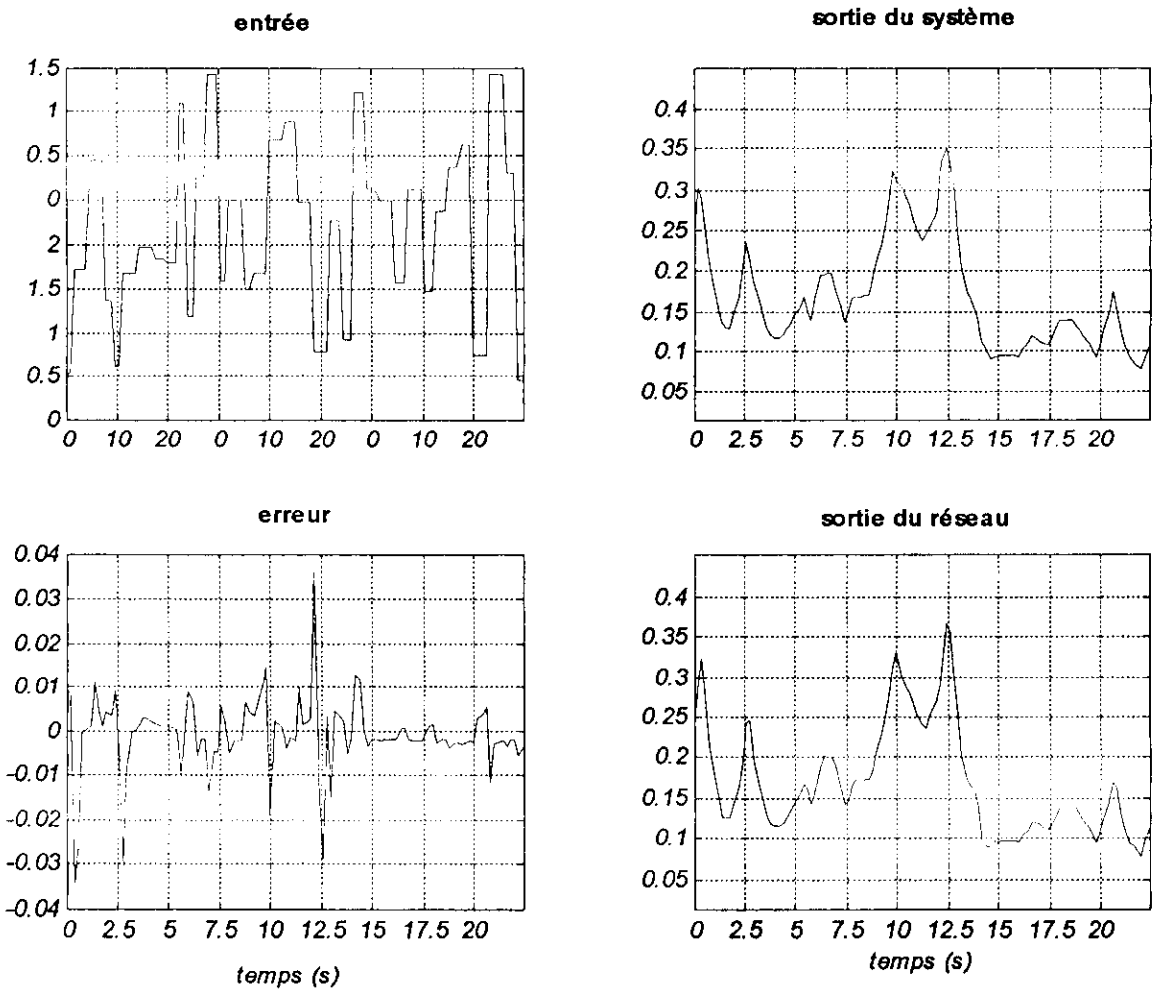


Figure IV.14
Résultats de test (validation du modèle)

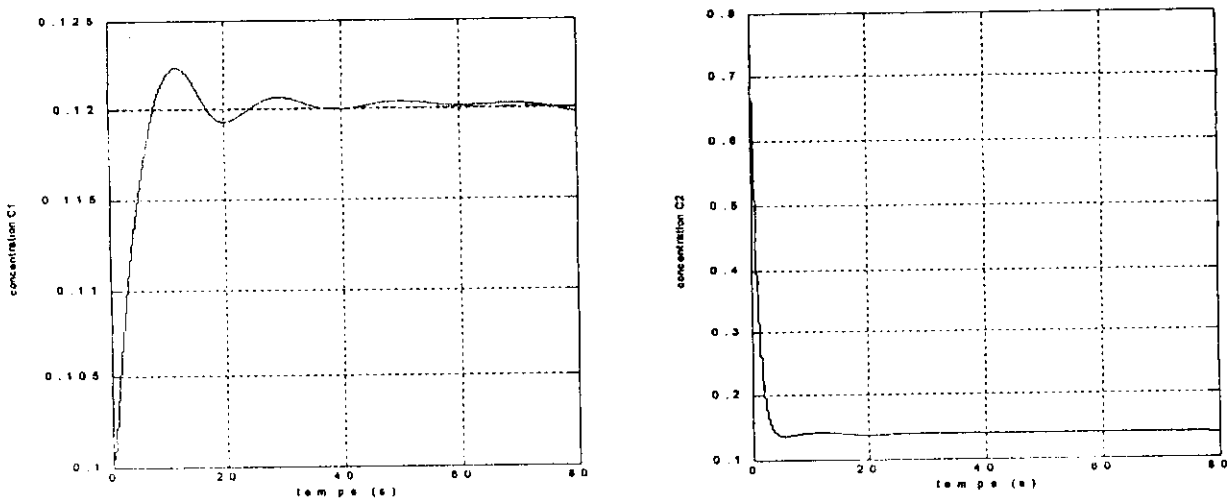


Figure 14
Réponse du système pour C1(0)=0.1, C2(0)=0.8, C1ref = 0.12

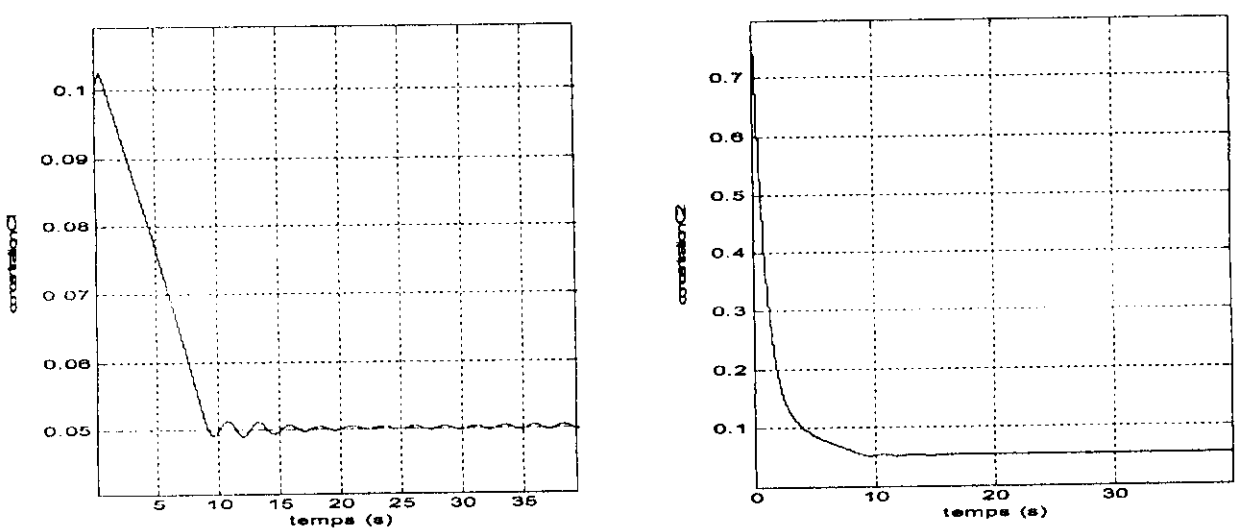


Figure 15
Réponse du système pour C1(0)=0.1, C2(0)=0.8, C1ref=0.05

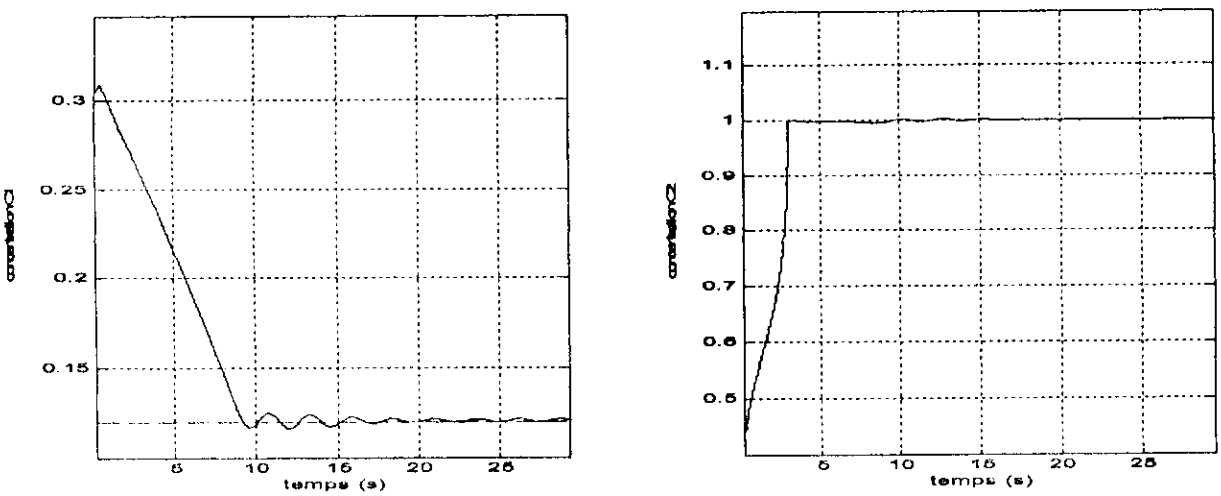


Figure 16
Réponse du système pour C1(0)=0.3, C2(0)=0.45, C1ref=0.12

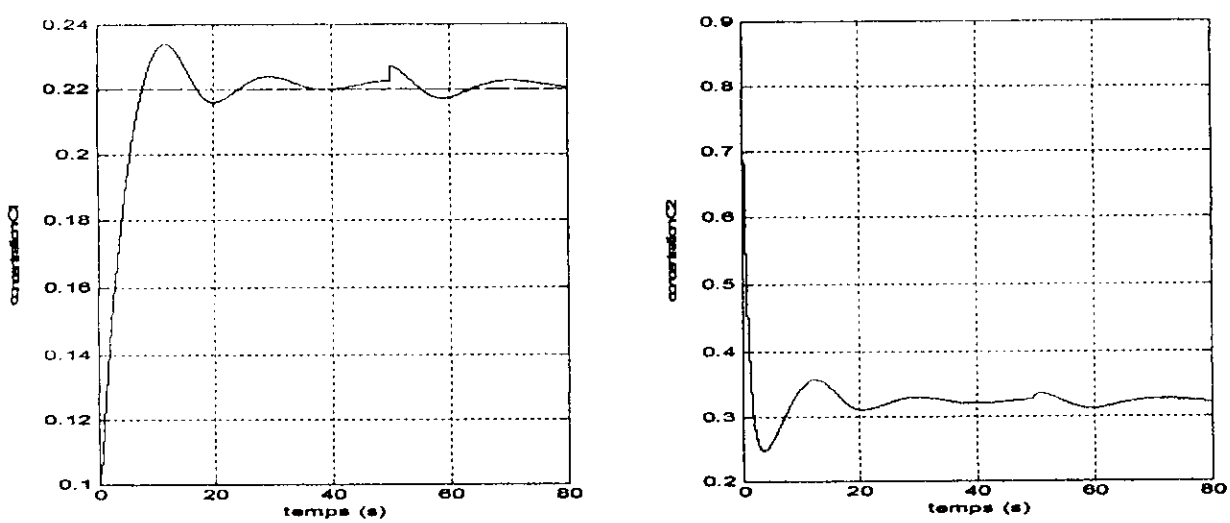


Figure 17
Réponse du système pour C1(0)=0.1, C2(0)=0.7, C1ref=0.22
Avec une perturbation à l'instant t=45s

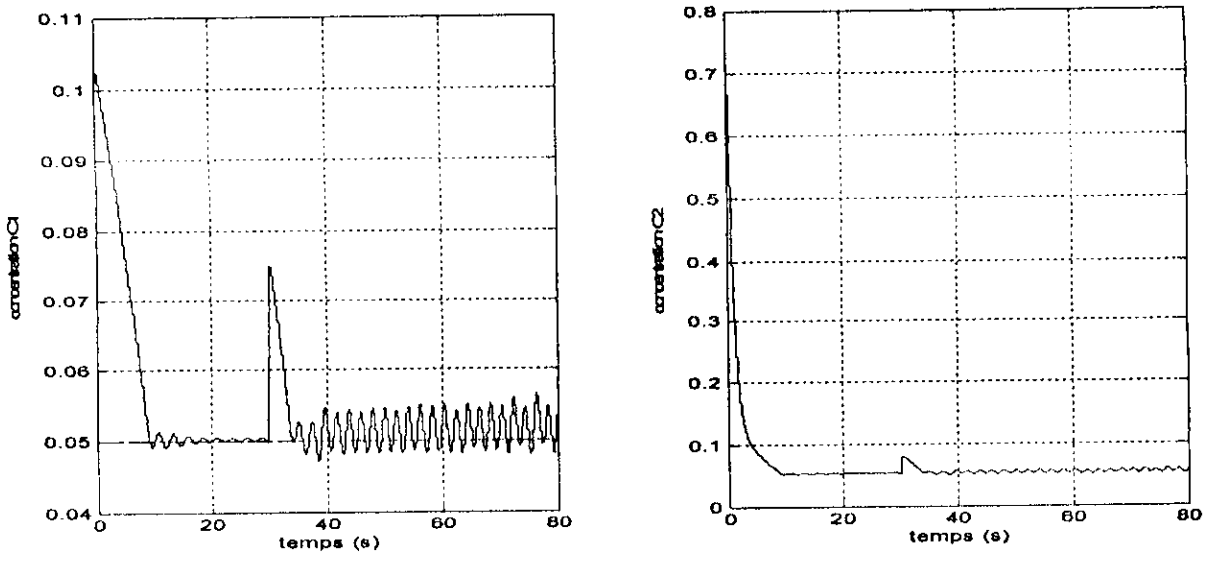


Figure 18
Réponse du système pour C1(0)=0.1, C2(0)=0.8, C1ref=0.05
Avec une perturbation à l'instant t=30s

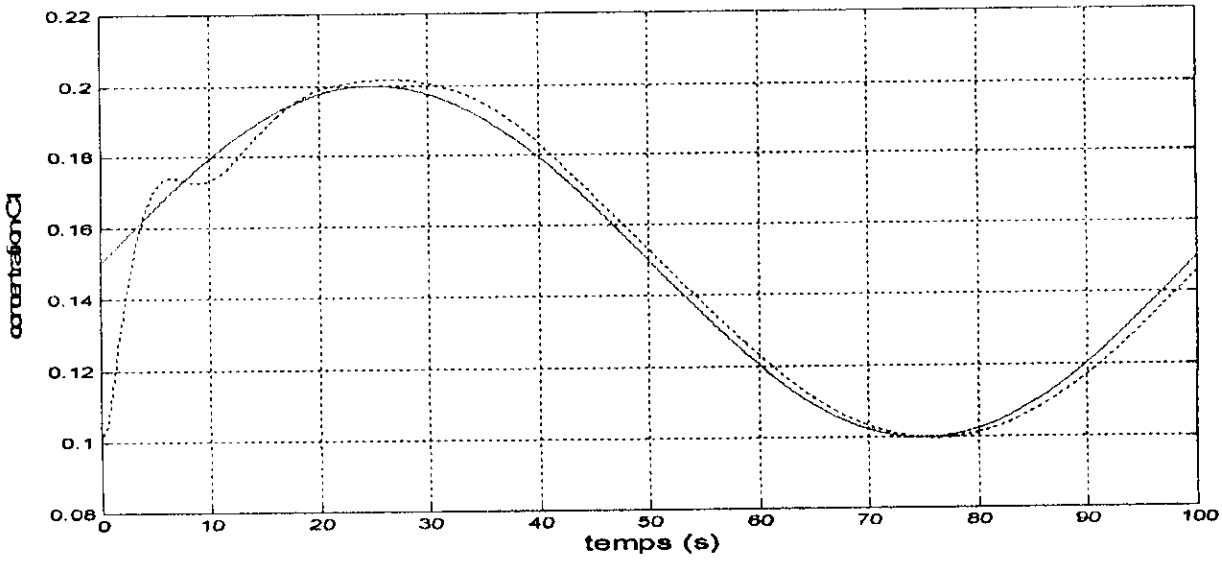


Figure 19
Réponse du système à une référence sinusoidale
avec C1(0)=0.1, C2(0)=0.8,

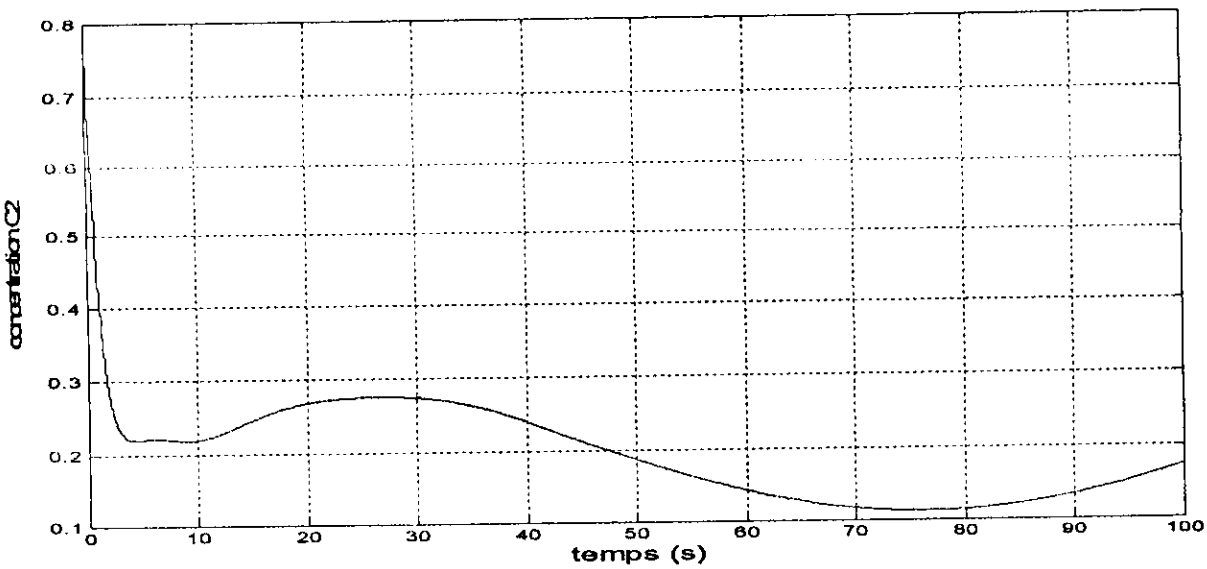


Figure 20
Sortie C2 pour une référence sinusoidale
avec C1(0)=0.1, C2(0)=0.8

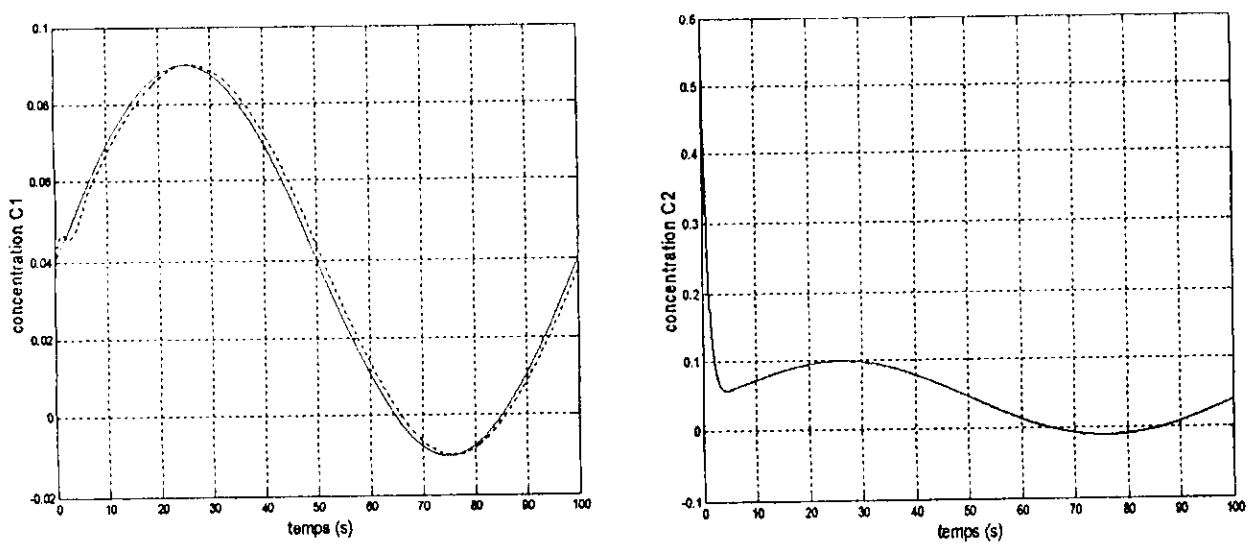


Figure 21
Réponse du système à une référence sinusoïdale
Avec les conditions initiales C1(0)=0.04, C2(0)=0.45.

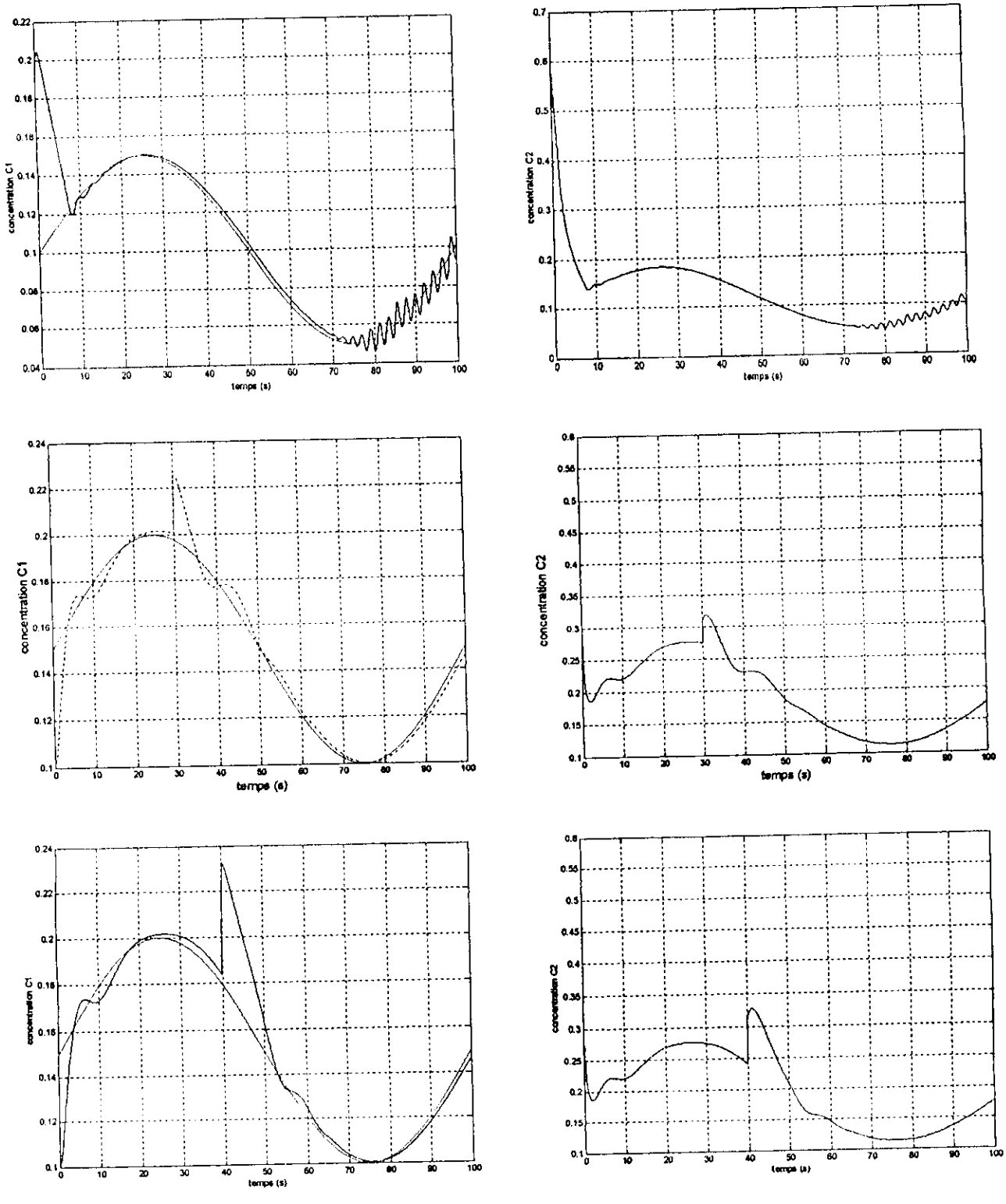


Figure 22

1.Réponse du système à une référence sinusoïdale pour C1(0)=0.2, C2(0)=0.6

2. Réponse du système à une référence sinusoïdale pour C1(0)=0.1, C2(0)=0.4, avec une perturbation de 30% à t=30s.

3. Réponse du système à une référence sinusoïdale pour C1(0)=0.1, C2(0)=0.6, avec une perturbation de 50% à t=40s.

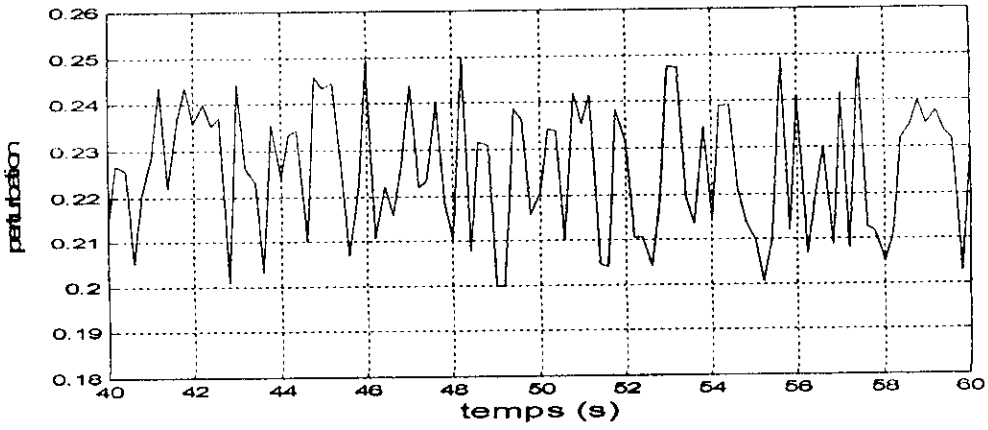


Figure
Perturbation variable (bruit) entre 40 s et 60 s

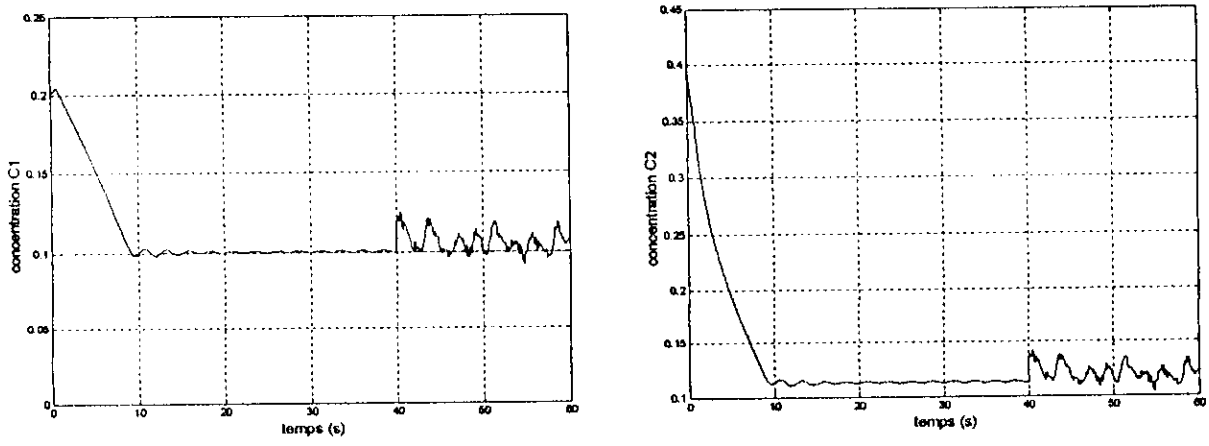


Figure
Réponse du système pour une référence de 0.1 et C1(0)=2, C2(0)=0.4
avec une perturbation aléatoire à t=40 s

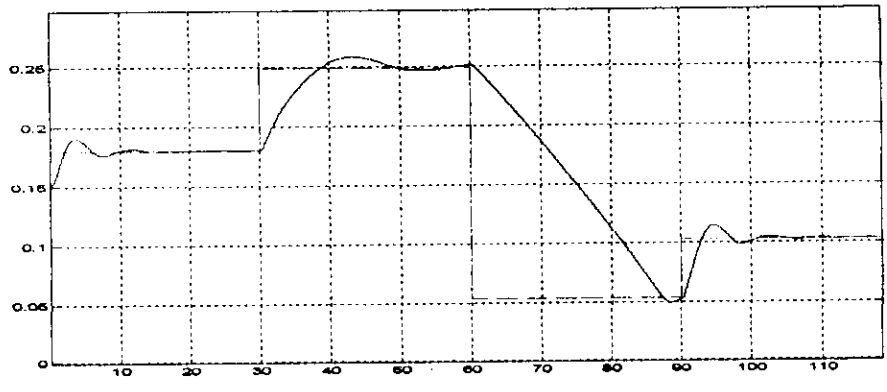


Figure
réponse du système pour une référence échelon variable pour C1(0)=0.15,C2(0)=0.8.

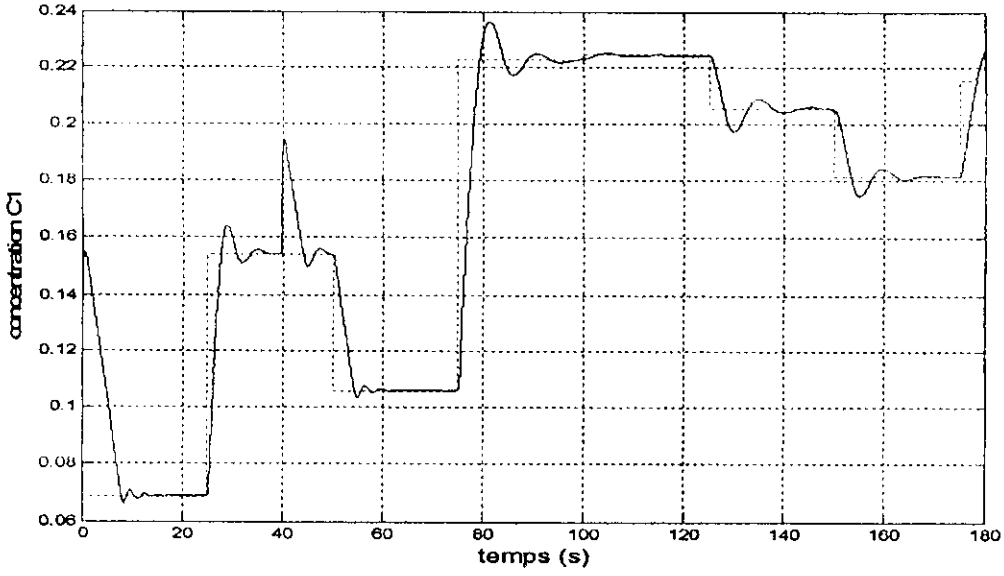


Figure 26
Réponse du système (Concentration C1) pour une référence échelon variable pour C1(0)=0.15,C2(0)=0.4.
et une perturbation à l'instant t=40 s.

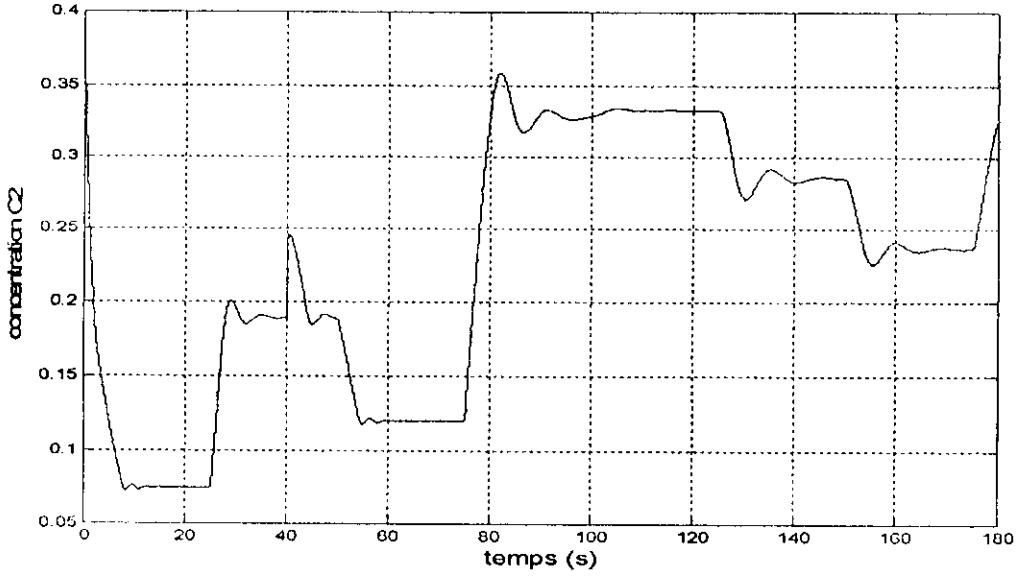


Figure 27
réponse du système (Concentration C2) pour une référence échelon variable pour C1(0)=0.15,C2(0)=0.4.
et une perturbation à l'instant t=40 s.

Conclusion

On remarque que le système suit la référence sous forme échelon et même une référence variable, mais pour les références variables en remarque qu'il y a toujours une petite erreur de poursuite, cette erreur diminue de plus en plus si la référence est un signal lent car le système à commander est lent (caractéristique des processus biotechnologiques), cette commande présente aussi une grande robustesse aux perturbations externe fixes où variables.

Annex A

Considérations pratiques

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

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

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

➢ *Comment traiter les exemples ?*

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

➢ *telles sont les dimensions du réseau .?*

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

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

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

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

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

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

- Le nombre de couches cachées est égal au produit du nombre d'entrées par le nombre de sorties.

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

Cette question est importante, car si on arrête l'apprentissage trop tôt le réseau n'aura rien appris, et si on arrête l'apprentissage trop tard le réseau risque d'apprendre le bruit qui entache les données, ce phénomène s'appelle *sur-spécialisation,* pour mieux comprendre ce phénomène, considérons le problème de l'approximation de la fonction $y = f(x)$, supposons que ces données soient entachées de bruit; Ainsi, au fur et à mesure de l'apprentissage, les réseau améliore ses performances jusqu'à réaliser une fonction proche de la fonction désirée. En poussant l'apprentissage, le réseau va apprendre les données bruitées, car il est capable d'approcher des fonctions plus complexes. D'après [Gér92], pour éviter la sur-spécialisation, il faudrait que le réseau possède une architecture adaptée aux données, ce réseau doit posséder la Propriété que le rapport entre les poids et le nombre d'exemples soit de 1/10. D'autre Part, afin d'éviter la sur-spécialisation, on conseille des réseaux de neurones qui Possèdent une architecture dynamique, qui convergent vers un nombre minimal de neurones tels que les "Gaussien Potential Function Network " GPFN (dans le cas d'une seule sortie) .

REFERENCES BIBLIOGRAPHIQUES

[Bar89] Bark, A. Pearlmuter. Learning State space Trajectories in Reccurent Neural Networks. Neural Computation 1, 1989, 263-266.

[Blo00] Bloch, G., Les réseaux de neurones pour le contrôle et l'optimisation des processus : deux applications industrielles. Centre de Recherche en Automatique de Nancy (CRAN), 2000.

[Cor98] Courriou, J. P., Commande des Procédés. Ed Londres New Yourk, 1998.

[Cha95] Chao-Che, K., and Wang, K., Diagonal Reccurent Networks for Dynamic Systems Control. IEE Transaction on Neural Networks, 1995, Vol 6, 144-156.

[Dre01] Dreyfus, G.,Neural Networks for Process Modeling and Control.École Supérieure de Physique et de Chimie Industrielles, 2001.

[Dre98] Dreyfus, G., les Réseaux de Neurones. Mécanique industrielle et matériaux, 1998, No 51.

[Hér94] Hérault, J.,Réseaux Neuronaux et Traitement du Signal. Traité des Nouvelles Technologies, ed HERMES, 1994.

[Hun92] Hunt, K. J. D., and Gawthrop, P. J., Neural Network for control systems. Automatica, 1992, Vol 6, pp1112.

[Hen98] Heniche, M., sur l'utilisation des réseaux de neurones et la logique floue à la commande des processus chimique, Thèse de Magister, ENP, 1998.

[Jer93] Jervis, T., Connectionist Adaptive control. Philosophy Doctorat Theses at The University of Combridge, 1993.

[Kar93] Karayanis, N. B., and Venetsanopoulos, Artificial neural networks, Learning, Performance, Evaluation and Application. Kluwer Acadimic Publishers, 1993.

[Kar93] Karakasoglu , A., and Sudharsanan, S. I., Identification and Decentralised Adaptive Control Using Dynamical Neural Network With Application to Robotic Manipulators. IEEE Transaction on Neural Networks, Novembre 1993, Vol 4, No 6.

[Kha96] Khalil, K. H., Adaptive Feedback control of Nonlinear Systems Represented by Input-Output models. IEE Transaction on automatic control, February 1996, Vol 41, No2.

[Khe94] Khemaissia, S., and Morris, A. S., Review of Networks and choice of Radial Basic Fonction Networks for Systems Identification. Technologies Avancées, 1994, N 6, pp 55-85.

[Kol89] Kollias, S., and Anastassiou, D., An Adaptiv least squares Algorithm for Effficient Transing of Artificial neural networks. IEEE Transaction on Circuits and systems, August 1989, Vol 36, No 8.

[Kun93] Kung, S. Y., Digital neural networks PTR printice Hall, 1993.

[Lee88] Lee, S., and Kil, M. R., Multiplayer Feedforword Potentiel Function Network. Second International Conference on neural networks, 1988, Vol 1, pp161-171.

[Lee91] Lee, S., and Kil, M. R., A Gausssian Potential Function Network With Hierarchically Self-Organising Learning. Neural networks, 1991, Vol 4, pp 207-224.

[Lin99] Linko, S., Yi-Hong, Z., and Linko, P., Applying Neural Networks as Software Sensors for Enzyme Engineering. Reviews Elsevier Science. 1999, pp155-163.

[Len00] Lennox, B. Montague, G.A. Hiden, H.G. Kornfeld,G. and Goulding P.R., Process Monitoring of an Industrial Fed-batch Fermentation, Control Technology Centre, Dept. of Chemical and Process Engineering, University of Newcastle, England.2000.

[Mul91] Muler, B., and Reinhart, J., Neural Networks. Pergamon Press 1991.

[Ner92] Nerrand, O., Roussel, R. P., Personnaz, L., and Dreyfus, G., Neural Networks and Nonlinear Adaptive filtrening : Unifying Concept and New Algorithms. Neural Computation, 1992, Vol 5, No 2, pp 165-199.

[Ner93] Nerrand, A., Personnaz, L.,Dreyfus, G., Non-Linear Recurssive Identification and control by Neural Networks. Uropean Control Conference, 1993.

[Phi98] Philippe, L., Aprrentissage et Diagnostic de Systèmes Complexes : Application à la Gestion en Temps Réel du Trafic Téléphonique Français. Thèse de Doctorat en Informatique à l'Université de Paris 6, 1998.

[Pat99] Patnaik, P.R., Coupling of a Neural Filter and a Neural Controller for Improvement of Fermentation Performance. Biotechnology Techniques 13, 1999, pp 735–738.

[Riv95] Rivals, I., Modélisation de Processus par réseaux de Neurones: Application au pilotage d'un Véhicule Autonome. Thèse de Doctorat en Physique à l'Université de Pierre et Marie Curie. Paris 6, 1995.

[Rum86] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., Learning Internal Representation by Error Propagation in Parallel Distributed Processing. Combrige, M.A, MST press 1986, Vol 8, pp 223-236.

[Ran95] Randers, J.M., Algorithmes Génétiques et Réseaux de Neurones, Application à la Commande des Processus. ed HERMES 1995.

[Riv94] Rivals, I., Canas, D., Personnaz, L., and Dreyfus, G., Modeling and control of Mobile Robots and Intelligent Vehicles. IEEE Conference on Intelligent Vehicles, 1994.

[Riv96] Nerrand, A., Personnaz, L.,Dreyfus, G., Modélisation, classification et Commande par Réseaux de Neurones : Principes Fondamentaux Méthodologie de Conception et Illustrations Industrielles. Ecole Supérieure de Physique et de Chimie industrielles de la Ville de Paris, Laboratoire d'Electronique, 1996.

[Rou01] Rousu, J., and Aarts, R.J., An integrated approach to bioprocess recipe design. Integrated Computer Aided Engineering, 8 (2001) .

[Rou99] Rousu, J., Elomaa, T., and Aarts, R.J., Predicting the Speed of Beer Fermentation in Laboratory and Industrial Scale. Engineering Applications of Bio-Inspired Artificial Neural Networks, Proc. 5th IWANN. Lecture Notes in Computer Science 1607, 1999, 893–901

[Urb95] Urbani, D., Méthodes statistiques de sélection d'architectures neuronales : application à la conception de modèles de processus dynamiques. thèse de Doctorat de l'université de Paris 6, 1995.

[Van95] Van der Walt, T.,Bernard, E., and Van Deventer, J., Process Modeling With the Regression Network. IEEE Transaction on neural networks, January 1995, Vol 6, No 1.

[Wil89] Williams, R. J., and Zipser, D., Experimental Analysis of Real Time Reccurent Learning Algorithme. Connection Science, 1989, Vol 1, No 1.

[Yed98] Yeddou, Y.M., Etude de Synthèse sur les Réseaux de Neurones et Leurs Applications, Thèse de Magister, ENP, 1998.

[Yua99]Yuan, J.Q., Vanrolleghem, P.A., Rolling Learning-Prediction of Product Formation in Bioprocesses, Journal of Biotechnology 69,1999, pp 47–62.

La commande des systèmes chimiques présentent une trés grandes difficultés en vue de leurs commandes à cause des non linéarités, les réseaux de neurones présentent l'une des solutions qui permettent de remédier à ces problèmes à cause de leur grande capacité d'approximation des fonctions non linéaires et leur apprentissage qui est assuré par l'application de l'algorithme de

Rétropropagation de l'erreur en sortie.

Mots clés :

Rétropropagation, algorithme, identification, commande, réseaux de neurones.

Absract

The chimecal system's control present a very big difficulties because there are very high non linearities, the neural networks control are the most technic used in this domain, thanks to their big capacity to approximate non linear functions and their learning wich is based on the backpropagation of the output error.

Key words :

Backpropagation, algorithme, identification, control, neural network.

ملخص :

في هذا البحث قمنا بدراسة للتحكم في المفاعلات الكيميائية و هذا لما لهامن صعوبة من ناحية التحكم فيها، و قد استعملنا التحكم عن طريق الخلايا العصبية الاصطناعية التي تعتمد أساسا في تعلمها على خوازمية الانتشار الخلفي للخطأ التحكمي .

كلمات مفتاحية:

تراجع خلفي, خوارزمية, مطابقة, تحكم, شبكة خلايا عصبية