REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE
D.E.R. DE GENIE ELECTRIQUE ET INFORMATIQUE
SPECIALITE: AUTOMATIQUE

THESE

Présentée Par : M. Yazid M'hamed YEDDOU
Ingénieur d'Etat en Automatique à l'Ecole Nationale Polytechnique

Pour l'Obtention du Diplôme de Magister en Automatique à l'Ecole Nationale Polytechnique

Thème

Etude de Synthèse sur les Réseaux de Neurones et leurs Applications

Soutenue le 09/06/1998 devant le jury composé de :

M. D. Berkani (Maitre de Conférence à l'ENP)	Président du jury.
M. M.S. Boucherit (Maitre de Conférence à l'ENP)	Rapporteur
M. M. Attari (Maitre de Conférence à l'USTHB)	Examinateur
M. F. Boudjema (Maitre de Conférence à l'ENP)	Examinateur
M. H. Chekireb (Chargé de Cours à l'ENP)	Examinateur

Juin 98

ملخص- كانت الشبكات العصبية في بداية ظهورها، محاولة لتجسيد خصائص العقل البشري بغرض الإستفادة منها. حاليا، أصبحت هذه الأخيرة تمثل مجموعة من العوامل الغير خطية. تجميع هذه العوامل في شكل شبكات يعطي لها فعالية عند استعمالها في مختلف الميادين. نذكر على سبيل المثال ميدان معالجة الإشارات، التصميم، والتحكم الآلي للجمل.

هذا العمل يمثل دراسة شاملة لمختلف الشبكات العصبية، والخوارزميات المتعلقة بتمرينها، من جهة. ومن جهة أخرى تطبيقات متعلقة بعملية التعرف على الأشكال (الأحرف اللاتينية، الأعداد العربية)، والتصميم والتحكم الآلي في الجمل المتمثلة في مفاعل كيميائي، ويد آلية. استعمالنا للشبكات العصبية، ذات تمرن بدون إشراف للتعرف على الأشكال، سمح لنا بالكشف عن إمكانياتها في ترشيح الشوائب، و إتمام الإشارات. أثناء عملية تصميم، تمكنا من مقارنة ومناقشة مختلف النتائج المتحصل عليها من خلال استعمالنا لمختلف الشبكات العصبية ذات التمرن المشرف عليه، كوسيلة لتقريب الدوال الغير خطية. في عملية التحكم الآلي اقترحنا عدة هياكل تحتوي على شبكات عصبية ذات تمرن مشرف عليه، و غير مشرف عليه. تمثيل النتائج المتحصل عليها في مختلف التطبيقات التي قمنا بها، سمح لنا باستنتاج مدى قدرات مختلف الشبكات العصبية المستعملة وحدودها.

كلمات مفتاحيه *- الشبكات العصبية، تصميم الأنظمة، التحكم الآلي، الترشيح، الإتمام والتعرّف.*

Abstract– Neural networks constituted, at first, a tentative aiming to mimic the biologic brain in order to benefit of its very interesting features. Currently, These ones designates a whole of no linear operators, who gathered in networks, could be effectively used in several operation of different disciplines. Among these disciplines, we can find signal processing, identification control of systems.

In this work, an extended study of Artificial neural networks and their training algorithms, with their application in different fields according to the model used is presented. Our applications are done in recognition of shapes (Latin characters and Arab numbers), and in identification and command of systems, where a chemical reactor and a robot arm are used. Our use of the unsupervised learning neural networks for the recognition of shapes, allowed us to insist on their faculties of noise filtering and signal completion. In identification, the results obtained by the different types of supervised training neural networks, as approximators of no linear functions, are discussed and compared. In control, some structures, containing supervised training networks and unsupervised training ones are proposed. The presentation of the different results obtained in every application and their discussion allowed us to conclude on the faculties of the different neural networks used and their limits.

Key words– **Neural Networks, System Identification and Control, Filtring, Completion and Recognition.**

Résumé –Les réseaux de neurones constituaient, lors de leur création, une tentative visant à mimer le cerveau biologique dans son fonctionnement afin de bénéficier de ses caractéristiques très intéressantes. Actuellement, Ces derniers désignent un ensemble d'opérateurs non-linéaires, qui rassemblés en réseaux, peuvent être efficacement utilisés dans plusieurs opérations relevant de différentes disciplines. Nous citons le traitement de signal, la modélisation et la commande de systèmes.

Ce travail porte une étude détaillée sur les réseaux de neurones et leurs algorithmes d'apprentissage. Nos applications se sont effectuées en reconnaissance de formes (caractère latin et chiffres arabes) et en identification et commande de systèmes, à savoir un réacteur chimique et un bras de robot. Notre utilisation des réseaux de neurones à apprentissage non-supervisé pour la reconnaissance de formes, nous a permis de mettre l'accent sur leurs aptitudes en filtrage de bruits et en complétion des signaux. En identification les résultats obtenus par les différents types de réseaux à apprentissage supervisé, en tant qu'approximateurs de fonctions non-linéaires, sont discutés et comparés. En commande, des structures, contenant les réseaux à apprentissage supervisé et ceux à apprentissage non-supervisé, sont proposées. La présentation des différents résultats obtenus dans chaque application et leur discussion nous ont permis de conclure sur les aptitudes des différents réseaux utilisés et leurs limites.

Mots-clés– **Réseaux de Neurones, Identification et commende des Systèmes, Reconnaissance, Filtrage et Complétion.**

TABLE DES MATIERES

IV. RESEAUX DE NEURONES A APPRENTISSAGE SUPERVISE

V. IDENTIFICATION DES SYSTEMES PAR RESEAUX DE NEURONES

VI. COMMANDE DE SYSTEMES PAR RESEAUX DE NEURONES

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Malgré la constante augmentation de la puissance des calculateurs, et les approches théoriques de plus en plus developpées, plusieurs opérations restent toujours difficiles à effectuer, et se trouvent heurtées à de sérieuses difficultés. Le grand nombre de données, leur variabilité, la nécessité de traitement en temps réel, et le fait que les problèmes posés ne répondent à aucun modèle physique clair, laissent parfois l'ingénieur démuni devant des tâches de reconnaissance, de caractérisation, ou de prise de décisions. Parmi ces opérations on trouve la modélisation de systèmes, le contrôle de processus, le filtrage ou la reconnaissance de signaux.

Face au développement récent de la biologie moderne et des neurosciences, la recherche ne peut rester indifférente aux multiples retombées que ces développements ont engendrées. En effet, plusieurs domaines ont connu l'éclosion de nouveaux concepts, qui en sont directement issus. L'une des évolutions les plus marquées, qui a été engendrée, est celle de l'introduction des réseaux de neurones dans le domaine des sciences de l'ingénieur.

Tout a commencé en 1943, lorsque *Mc culloch* et *Pitts* ont inventé le premier *"neurone artificiel"*, qui n'était qu'un produit scalaire d'un vecteur d'entrées, et un vecteur poids, suivi d'un élément à seuil [Lip87, Hus93]. Depuis, les recherches n'ont pas cessé, et "la ruée vers le neurone" était longue et pleine de difficultés.

Au début, pendant plus de deux décennies, tous les travaux avaient comme principal objectif de pouvoir s'approcher, un jour, du fonctionnement du cerveau biologique, afin de bénéficier de plusieurs de ses fascinantes caractéristiques. En effet, l'origine de plusieurs travaux actuels peut être trouvé dans les recherches faites pendant cette époque, qui visaient la métaphore biologique [Koh 77, Ant90, Fre92].
Ainsi, le principe de l'apprentissage par adaptation des poids synaptiques des réseaux, n'est pas sans lien avec la théorie de *"plasticité synaptique"* de *S. Freud*. Théorie selon laquelle, les neurones préforment leur apprentissage à travers les excitations répétées appliquées à leurs entrées [Hér93, Puz95].
Plus significatif encore, tous les algorithmes d'apprentissage non supervisé connus à nos jours, découlent de la théorie d'apprentissage de, cet autre psychanalyste, *J. Hebb*; Qui elle, à son tour, repose sur la fameuse expérience de *J. Pavlov* [TRC89].

Cette idée a donné ses fruits jusqu'à la fin des années 60. En 1969, *M.Minsky* et *S.Papert* ont publié une étude prouvant l'inefficacité des réseaux, développés jusque là, devant les problèmes de non-linéarité [Kos92]. Quelques années plus tard, plusieurs modèles de réseaux ont été mise en oeuvre mais leurs aptitudes sont restées limitées. Ainsi, les réseaux de neurones ont traversé une période difficile pendant toute la décennie des années 70. Une période qui était, cependant, riche en travaux. D'illustre chercheurs dans ce domaines sont apparues pendant cette époque. Nous citons T.Kohonen, J. Andeson, S. Amari, S. Grossberg et B. Widrow qui ont été à l'origine de plusieurs modèle de réseaux, développés plus tard et utilisés à nos jours [Koh77, Coh83, Fuk83, Per86, Koh88]. Mais tous les travaux effectués pendant cette période étaient souvent liés à la métaphore biologique.

Vient par la suite l'inévitable question *"des avions qui volent mais qui ne battent pas des ailes"*. Ainsi, l'idée de pouvoir, un jour, approcher le fonctionnement du cerveau biologique fut

abandonnée au profil d'une autre démarche, plus sûre et plus réaliste. Les mathématiques et la physique théorique sont venus prendre le relais, et ont, enfin, permis aux réseaux de neurones de dépasser cette période de "blocage" qui a duré pendant plusieurs années. C'est J. Hopfield puis J. Grossberg qui, au début de la décennie suivante, ont résolu, pour la première fois, le problème de stabilité des réseaux dynamiques, en se basant sur la théorie de l'évolution des systèmes physiques vers l'équilibre.
Cette nouvelle démarche a conduit, par la suite, à la mise en oeuvre d'un nouvel algorithme d'apprentissage, appelé Backpropagation, qui est venu ouvrir une nouvelle ère pour les réseaux de neurones [Rum86].C'est surtout à cet algorithme, que revient le mérite, si les réseaux à couches sont couramment utilisés aujourd'hui, et ont conquis plusieurs domaines [Rum90, Fre92, Sim90].

Actuellement, les réseaux de neurones bénéficient de fondements théoriques solides. Ces derniers constituent un ensemble d'opérateurs non linéaires, qui permettent de constituer, grâce à leur adaptabilité, une large gamme d'outils différents, pouvant être utilisées dans divers problèmes, suivant leurs spécificités [Sim90, Kos92, Kun93].
De plus, le développement d'algorithmes, performant l'apprentissage de ces réseaux, leur a ouvert de nouvelles perspectives d'utilisation [Kol89, Bab89, Bil90, Hun90, Hér93, Van95].

Les réseaux de neurones se sont avérés particulièrement adaptés, dans le domaine de l'automatique. Leurs capacités d'approximation sont mise à profil pour la modélisation des systèmes [Dem91, Che 90, Bur93]. Leur adaptabilité leur permet d'être utilisés, pour construire des régulateurs pour la commande de processus [Bar87, Has92, Ngu90, Kho96, Pol96].
D'un autre coté, les caractéristiques dont jouissent ces réseaux, en tant que filtres adaptatifs, leur permettent d'être utilisés pour les opérations de reconnaissance [Eber95] et, plus généralement, en traitement de signal [Ner92, Hus93, Hér93].

Ainsi de la classification au filtrage, de la modélisation au contrôle et à l'optimisation, les applications où les réseaux de neurones ont prouvé leur efficacité ne cessent de s'élargir.

Le présent travail porte sur les réseaux de neurones et leurs applications. Il sera présenté une synthèse de nos recherches bibliographiques sur les différents modèles de réseaux de neurones, leurs différentes structures et les algorithmes de leur apprentissage. Dans nos applications, nous mettons en évidence les avantages et les inconvénients des modèles utilisés et présentons les solutions que nous avons pu dégager au vu de certains problèmes.

Cette étude comprend six chapitres.

Dans le chapitre I, nous introduisons les concepts fondamentaux sur les réseaux de neurones. Nous étudierons les différentes classes de réseaux de neurones ainsi que les notions importantes liées à leurs fonctionnements.

Le chapitre II sera consacré aux réseaux à apprentissage non supervisé. Nous présenterons les modèles les plus importants de cette famille, et nous expliquerons leurs modes de fonctionnement.

Dans le chapitre III, nous utiliserons les réseaux à apprentissage non supervisé en reconnaissance de formes. Dans nos applications, nous effectuerons la reconnaissance de caractères latins puis celle de chiffres arabes. Nous utiliserons quatre types de réseaux différents,

afin de constater les résultats et conclure sur les aptitudes et les limites des différents réseaux utilisés dans ces opérations.

Dans le chapitre IV, nous nous intéresserons à la classe des réseaux à apprentissage supervisé. Une partie importante sera consacrée aux différentes structures de ces réseaux et leurs algorithmes d'apprentissage.

Le chapitre V de cette étude, portera sur l'identification de systèmes par réseaux de neurones. Des applications sur quelques modèles seront présentées afin de pouvoir conclure sur les aptitudes des différents réseaux, ainsi que leurs méthodes d'apprentissage, dans ce domaine. Nous ferons, par la suite, l'identification d'un réacteur chimique, afin d'effectuer sa commande dans le chapitre suivant.

Le chapitre VI sera consacré à la commande neurale. Après une étude sur les différentes stratégies de commande possibles, nous utiliserons les réseaux de neurones étudiés pour mettre en oeuvre plusieurs techniques de commande de systèmes. Nos applications se feront sur un bras de robots puis sur le réacteur chimique identifié dans le chapitre IV.

A la fin, et à la lumière des résultats obtenus dans nos différentes applications, nous présenterons les conclusions.

CHAPITRE I:

INTRODUCTION AUX RESEAUX DE NEURONES

CHAPITRE I
INTRODUCTION AUX RESEAUX DE NEURONES

A première vue, la biologie et la science de l'ingénieur n'ont rien en commun. Pendant que le biologiste tente de percer les mystères de la vie en essayant de comprendre ses phénomènes et ses mécanismes, l'ingénieur travaille à créer d'autres mécanismes tendant à diminuer ou pallier l'intervention et l'effort humain. Ainsi ces deux spécialistes semblent se tourner le dos en regardant dans des directions opposées. Et pourtant nous pouvons au moins affirmer que les connaissances en biologie constituent un terrain d'inspiration de certains domaines de la science d'ingénieur.

Contrôler un système complexe, Modéliser un processus, détecter un objet mobile, reconnaître une forme ou une voix au milieu d'un bruit ambiant, sont autant de domaines de recherche pour l'ingénieur.

Malgré la puissance grandissante des calculateurs et la mise en oeuvre d'approches théoriques de plus en plus sophistiquées, plusieurs des problèmes posés dans le cadre de ces recherches résistent encore aux algorithmes et aux méthodes classiques. Ces problèmes sont souvent dépendant de l'environnement et ne répondent pas à des modèles physiques connus.

Chez les être vivants, la reconnaissance rapide d'un visage, Le contrôle d'une centaine de muscles du corps et la génération de trajectoire suivant la survenue d'obstacles, par exemple, sont autant de tâche remplis quotidiennement de manière naturelle. Faire réaliser ces opérations par une machine "c'est là le défi". Il est donc aisé de comprendre l'intérêt marqué par l'ingénieur au neurone biologique. Sans prétendre copier le cerveau, il entend s'inspirer des architectures et des fonctions du système nerveux.
Le développement continuel des connaissances en biologie, l'apparition de nouvelles méthodes théoriques et l'incessante montée en puissance des outils de simulation autorisent les meilleurs espoirs.

C'est en 1943, que Mc Culloch et Pitts ont inventé le premier " Neurone Artificiel " de l'histoire. Cette idée a vite fleuri et fait son chemin.
Ainsi après la publication de la théorie du psychanalyste J.Hebb sur la plasticité synaptique (basée sur la fameuse expérience de Pavlov), dans les années 1950, Rosenblat présenta le *"Perceptron"* qui n'avait de différent par rapport au modèle de Mc Culloch et Pitts, que sa faculté d'apprentissage basé sur la règle de Hebb; ce qui constitua une nouveauté à l'époque. Ce réseau qui était capable de prendre de bonnes décisions en classification a vite montré son inaptitude devant les problèmes non linéaires. En 1969 Papert et Minsky ont démontré clairement les limites du Perceptron, et la nécessité de plusieurs couches identiques à celui ci, pour résoudre ces problèmes.
N'ayant aucun moyen pour faire l'apprentissage de ces couches de neurones nécessaires, cette étude plongea l'idée de réseaux de neurones dans l'ombre. [Sim90].

Durant les années 1970, les travaux se sont orientés vers les mémoires associatives, qui sont des réseaux à apprentissage non supervisé à architecture généralement interconnecté. Nous citons parmi ces travaux, ceux de S.Amari [Ant90], C.Anderson et T.Kohonen [Koh77] qui ont apporté des modèles très intéressants. Mais ces derniers se sont heurtés aux problèmes d'instabilité et de concepts liés aux systèmes dynamiques, que ni la théorie mathématique ni les moyens de calcul disponibles à l'époque, n'ont pu prendre en charge. Les réseaux de neurones ont ainsi connu ce que nous pouvons appeler leur "traversée du désert."

Il a fallu attendre 1982 pour voir, avec le travail de J.Hopfield [Koh87] puis celui de M.A Cohen & S.Grossberg[Coh83], définitivement réglé le problème de stabilité des réseaux dynamiques. Ce premier obstacle franchi, plusieurs travaux ont pu voir le jour tels que Les BAM de B. Kosko [Kos92], les OLAM de L. Personaz [Per86] et bien d'autres.

Le plus grand événement qui est venu propulser les réseaux de neurones par la suite, est sans doute le résultat du travail d'un groupe de chercheur de l'université de Stanford Rumelhart, Hinton, and Williams en 1986 [Wid90]. Ce travail a permis la résolution du vieux problème posé par Minsky et Papert, en mettant en oeuvre l'algorithme d'apprentissage de Backpropagation. Désormais, il est possible d'effectuer l'apprentissage des réseaux de neurones statiques à plusieurs couches et de ce fait de s'attaquer aux problèmes non linéaires.

A partir de là, les réseaux de neurones ont connu un essor continu. Ainsi à cet algorithme d'apprentissage, sont venus s'ajouter d'autres, qui lui ont apporté des améliorations [Lip87, Bab89, Wer90, Tri93, Hus95]. Les architectures de ces réseaux ont eux aussi connu une très grande évolution, donnant naissance à de nouveaux types de réseaux, parfois plus complexes, mais souvent plus adaptés à certains problèmes, tels les DBNN (*Data Based Neural Networks*), [Kun93], Les RBF (*Radial Based Networks*) [Khe94, Ren95], les Réseaux Gaussiens ou GPFN (*Gaussian Potential Fonction Networks)* [Sca92] et les *Réseaux Dynamiques* [Kar93, Ku95].

Les réseaux de neurones, avec leurs aptitudes en classification, mémorisation, filtrage et d'approximation, sont devenus un moyen très efficaces et ont conquis plusieurs domaines. Nous citons le contrôle et modélisation des système [Nar92, Hun90, Lev94], le traitement de signal [Bel96] et notamment de l'image [Yed95].

Dans ce chapitre, nous introduisons les principales notions relatives aux réseaux de neurones, telles que leur architecture, fonctionnement, apprentissage et stabilité, qui nous permettrons d'entamer notre étude

I.1 Le neurone biologique

La connaissance de la structure interne du cerveau biologique, et la compréhension de son fonctionnement a été d'une extrême importance, pour l'émergence des réseaux de neurones artificiels, à la fin de la première moitié de ce siècle. Nous présentons, dans ce qui suit, un bref aperçu sur le neurone biologique et son fonctionnement.

I.1.1 Définition

Le Neurone est l'unité fonctionnelle de base du système nerveux. D'une espèce à une autre, celui ci peut présenter des différences du point de vu fonctionnel ou anatomique. Cependant divers points communs subsistent, et sont à la base de la cellule nerveuse que nous présentons ici.

I.1.2 Anatomie du neurone

Le neurone est une cellule constituée principalement de trois parties (figure I.1) qui, au regard du transfert d'informations, ont un rôle fonctionnel bien défini. Ce sont les dendrites, le Soma et l'Axone.

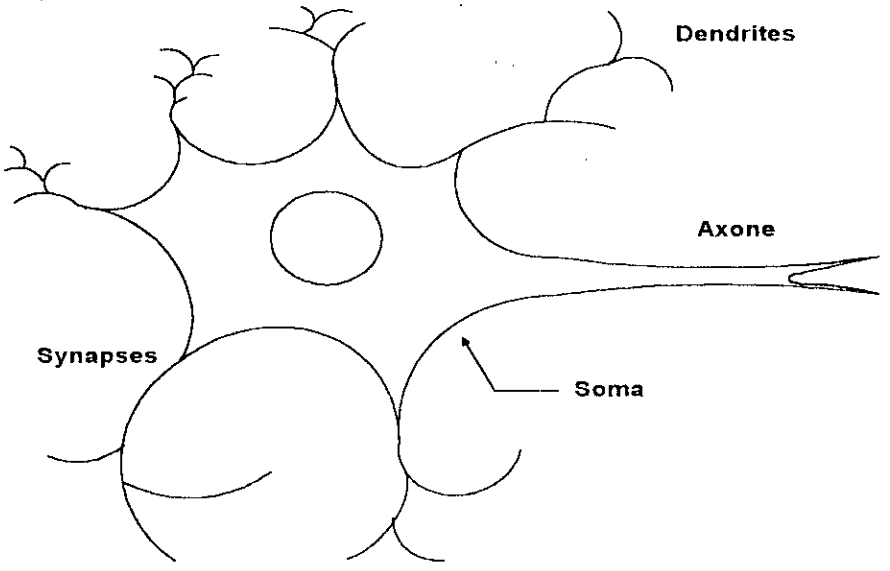


Fig. I.1 Le neurone biologique.

a. La Dendrite

Cette partie qui présente une sorte de ramifications appelées *arbre dendritique*, collecte les signaux venant d'autres cellules ou de l'extérieur. La réception des signaux en provenance des autres neurones, se fait par des points de contacts appelés *Synapses*. Certaines cellules peuvent compter jusqu'à 100 000 synapses [Hér94]. Ces informations, qui ne sont rien d'autre que des impulsions électriques, sont par la suite acheminées vers le corps cellulaire ou le Soma.

b. Le Soma

L'arbre dendritique fait converger vers le Soma des influx nerveux venus d'une très large étendue autour du neurone. Le Soma, outre son rôle concernant le métabolisme de la cellule, recueille et concentre les informations reçues et en fait une sommation dite "spatio-temporelle". Si le potentiel somatique dépasse un certain seuil, il y a émission d'un

potentiel d'action, appelé " Spike ", qui correspond à une oscillation électrique très brève (1ms) [Mul91].

c. L'Axone

L'information traitée est transmise vers l'extérieur, le long de l'axone et repartie sur les synapses des neurones cibles grâce à l'arborescence terminale que possède l'axone.

I.2 Le neurone formel

En 1943, Mc Culloch et Pitts Inventèrent le premier modèle de neurone artificiel (Fig.I.2). Inspiré du neurone biologique, ce modèle n'était rien d'autre qu'un produit scalaire entre un vecteur d'entrée et un vecteur poids suivi d'une fonction seuil[Dav90]. Depuis, plusieurs types de réseaux avec des modèles de neurones inspirés du premier modèle ont été élaborés.

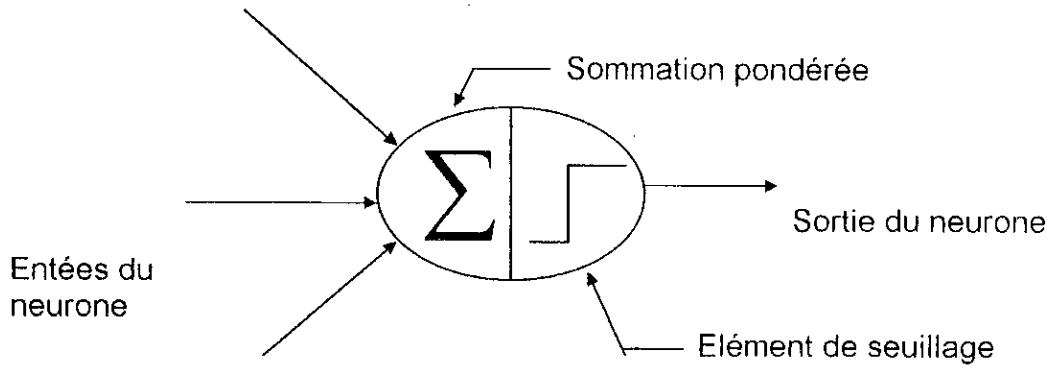


Fig. I.2 Le modèle de Mc Culloch et Pitts

Au début le souci était de se rapprocher le plus possible du modèle biologique dans l'espoir de pouvoir synthétiser, un jour un réseau qui peut ressembler au cerveau humain dans son fonctionnement. Plus tard, les chercheurs se sont plutôt occupés de l'efficacité, et de concepts théoriques et outils mathématiques nécessaires pour cela, entraînant ainsi des modifications sur le premier modèle[Gil87].
Nous présentons ci dessous un modèle théorique général pour le neurone.

Modèle mathématique général du neurone

La figure I.3 présente un modèle d'un neurone formel

Fig. I.3 Schéma synaptique des différentes fonctions mathématiques à l'intérieur d'un neurone formel

I.2.1 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. Cette fonction est depuis longtemps la plus usitée. C'est le cas pour les réseaux multicouches [Rum90] et d'une manière générale les réseaux **LBF** (Linear Basis Fonction).
Afin d'améliorer les performances des réseaux, d'autres formes de fonctions *discriminant* non linéaires ont été élaborées ces dernières années [Hus95], offrant ainsi aux réseaux de meilleures possibilités dans certains domaines et un large champ d'application [Kun93]. Ces Techniques nécessitent néanmoins des calculs plus compliqués pour leur élaboration. Nous citons ci dessous les fonctions de bases utilisées

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

Cette fonction est une sommation pondérée des entrées vers le neurone. Sa forme est en général définie par:

$$\Phi(z,w)=z^T w \tag{I.1}$$

Où W représente la matrice des poids et Z les entrées qui viennent de l'extérieur ou de la part des autres neurones vers le neurone en question. En plus de son efficacité, cette fonction est la plus simple à implémenter.

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

La forme de cette fonction est inspirée des réseaux de neurones utilisés pour la classification notamment les ART de Cohen et Grossberg [Kun94].

$$\Phi(x,w)=-\frac{\|x-w\|^2}{2}. \tag{I.2}$$

Dans les réseaux 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 entrées x et le centre de chaque classe. En effet cette fonction sert à effectuer un échantillonnage de l'espace des entrées, où chaque groupe de poids synaptiques représente une concentration de données.

c. La fonction de base Elliptique EBF (elliptic Basis Fonctions)

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

$$\Phi(x,w)=\sum_{k=1}^{n}\alpha_k(x-w)^2+\theta \tag{I.3}$$

Celle ci a une forme modulée par des paramètres α_k et dotée d'un Biais θ.
Ces paramètres font qu'elle ne soit pas forcément symétrique. Cette fonction conserve cependant les mêmes caractéristiques que la précédente, mais elle est très dépendante des paramètres qui déterminent les formes de zones de regroupement de données dans l'espace des entrées.

I.2.2 Fonction dynamique linéaire

Ce bloc détermine la dynamique du réseau. Dans le cas général, cette fonction est régie par l'équation différentielle de premier ordre suivante:

$$\alpha_0 u_i(t) + \alpha_1 u_i(t) = v_i(t). \qquad (I.4)$$

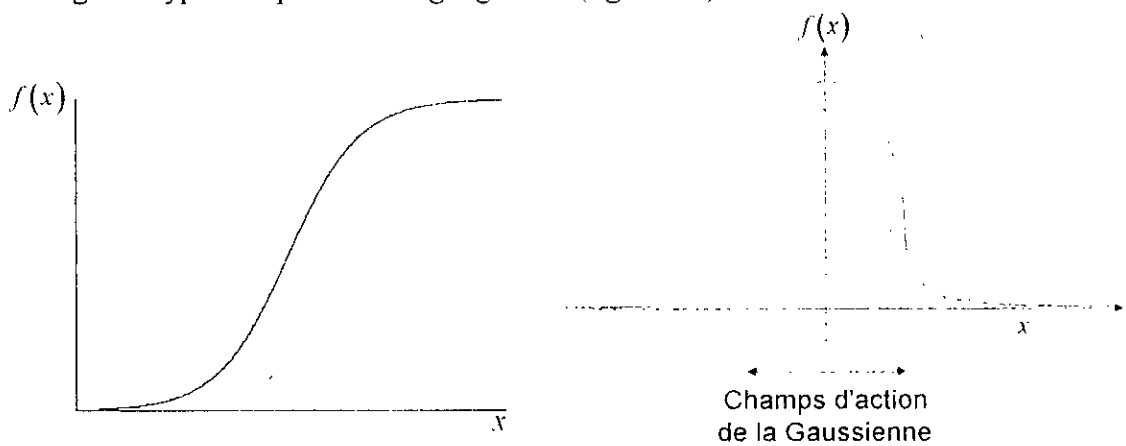
où u_i représente l'activité du *i éme* neurone. et v_i représente l'entrée du systeme dynamique décrit par l'équation (I.4).

Cette équation reflète l'activité électrique réelle du neurone, à savoir la charge et décharge de potentiel. C'est J. Hopfield qui, pour la première fois [Ant90], en 1982 a modélisé un réseau de neurones par un circuit électrique régi par cette équation, Traduisant son comportement dynamique.

Un réseau statique, représente un cas particulier de l'équation (I.4) où la première dérivé est tout simplement nulle. Celle-ci ne devient donc qu'une simple fonction linéaire.

I.2.3 Fonction non-dynamique non linéaire

C'est la fonction dite d'activation. Celle-ci a pour objectif de rendre l'activité du neurone bornée. Pour ce faire, une non-linéarité est donc nécessaire. La première fonction qui a été proposée est la fonction *Seuil (Thresholding)*. Or cette fonction qui délivre une sortie binaire n'est mathématiquement pas adaptée à certaines opérations notamment la différentiation qui, comme on le verra, est nécessaire pour l'amélioration des paramètres internes du réseau dans la plupart des cas. Pour cela, la fonction d'activation doit être continue, dérivable et monotone. Ainsi d'autres fonctions qui effectuent elles aussi le seuillage mais d'une manière moins abrupte sont utilisées. Ces fonction sont dite de *Forme-S(a cause de leur allure)*. Parmi ces types de fonctions on retrouve la sigmoïde ou la tangente hyperbolique ou la Log-sigmoïde.(figure I.4).



a.) **b.)**

Fig.I.4 Allure de fonctions d'activation
a. Fonction Sigmoïde; **b.** Fonction Gaussienne: (caractéristique locale)

Parmi les fonctions d'activation utilisées, il existe cependant une qui viole la condition de monotonie, c'est la fonction Gaussienne(fig.I.4.b). Cette fonction qui se distingue des autres, a une non linéarité plus significative et un comportement impulsionel. Mais là ne sont pas les principales raisons pour lesquelles la gaussienne a connu un engouement ces dernières années. C'est en effet, le comportement local de cette fonction qui la rend très efficace. Utilisée avec un discriminant de fonction de base radiale qui partage l'espace

d'entrée en différentes régions, cette fonction a l'aptitude de répondre d'une manière différente dans chaque zone (fig.I.4.b). La caractéristique de comportement local, rend cette fonction très puissante en approximation de fonction et en classification.
Toutes les fonctions citées ont généralement un fondement biologique [kos92]. Les neurones du système visuel sont par exemple connus ayant un comportement Gaussien [Fre92]. La recherche de l'efficacité a permis à d'autres fonctions, qui ont plutôt montré leurs performances mathématiques, de voir le jour ces dernières années. Nous citons parmi ces dernières les fonctions Logarithmiques, Cubiques, ou celles Exponentielles Logarithmiques[Hér94].
Enfin, la sortie du neurone peut aussi être linéaire. C'est le cas de la sortie des réseaux multicouches utilisés pour l'approximation de fonctions

I.3 Champs de neurones

Le neurone lui même, en tant qu'unité autonome élémentaire n'a aucun pouvoir. La force et l'efficacité du cerveau résident en effet, dans le regroupement de ces neurones et le partage des tâches entre eux. Ce regroupement est appelé un *champ de neurones* où chaque élément reçoit et envoie de l'information. L'organisation de plusieurs champs entre eux, constitue un réseaux de neurone, dont les unités à l'intérieurs doivent travailler ensemble pour remplir une certaine tâche bien déterminée. Ces neurones sont reliés entre eux par des connexions. Les mécanismes de cette organisation déterminent l'architecture du réseau.

I.4 Classification des réseaux de neurones

La figure I.5 résume les trois différentes possibilités suivant lesquelles on peut classifier les réseaux de neurones. Dans ce qui suit, nous nous serviront de ces différentes classifications pour faire l'étude de plusieurs points liés aux réseaux de neurones.

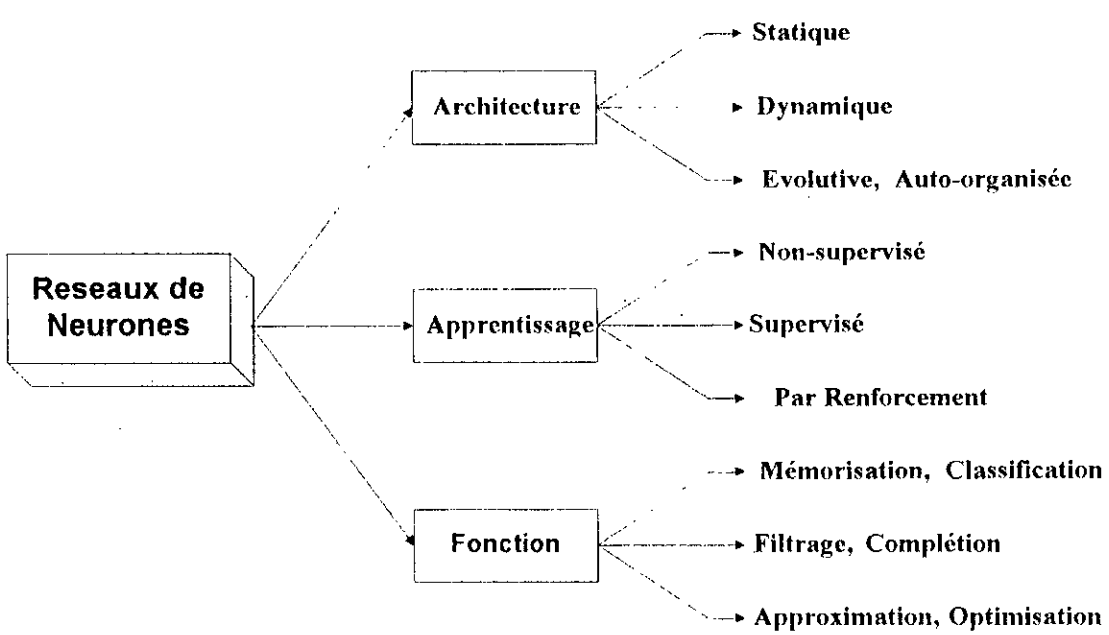


Fig.I.5 Les différentes possibilités de classification des réseaux de neurones.

I.5 Architecture des réseaux de neurones

L' architecture défini le fonctionnement du réseau. Il existe trois types de réseaux à architectures différentes.

I.5.1 Réseaux statiques

Ce type de réseaux est organisé généralement en couches de neurones. Chaque neurone d'une couche reçoit ces entrées à partir des neurones de la couche précédente ou tout simplement de l'entrée du réseau. Dans de tels réseaux il n'existe pas de "feed-back" (retour) d'information. Ces réseaux peuvent être utilisés pour les problèmes de classification ou d'approximation de fonctions non-linéaires complexes.

I.5 2 Réseaux dynamiques

L'introduction de feed-back entre les neurones rend le réseau dynamique (Fig.I.6). Cette structure dynamique est modélisée par le deuxième bloc de la figure 1.3 et est gouvernée par l'équation (I.4).
L'introduction d'une telle structure qui rend le réseau dynamique nécessite l'étude de la stabilité de ce dernier. Au début, ces réseaux étaient souvent utilisés pour les problèmes de classification, et de mémorisation. Actuellement, dans plusieurs travaux la structure interne dynamique de ces réseaux est exploitée pour l'identification ou la commande de systèmes dynamiques [Pea89] [Wer90] [Kar93] [Char95].

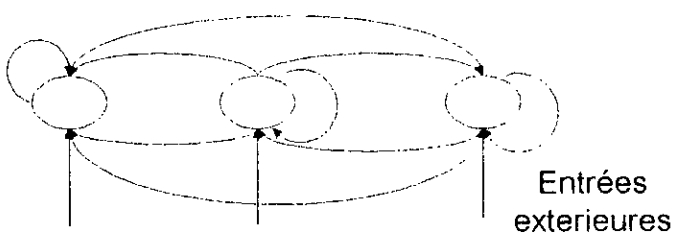


Fig.I.6 Architecture générale d'un réseau dynamique.

I.5.3 Réseaux à architecture évolutive et réseaux Auto-Organisés

Les réseaux Auto-organisés sont des réseaux de neurones qui changent leurs structures interne pendant l'utilisation. Ainsi les neurones se regroupent topologiquement suivant la représentation des exemples issus de l'espace d'entrée. Ces réseaux sont généralement des dérivées des modèles de Kohonen, dont le quantificateur de vecteurs LVQ (*Linear Vector Quantiser)*[Koh87] .
Les réseaux sont dits évolutifs au vu de leur méthode d'apprentissage; c'est la dimension du réseau qui change pendant l'entrainnement. Ainsi, le nombre de neurones augmente ou diminue. La structure interne (la nature du réseau) n'est cependant pas bouleversée par ce changement. Parmi ces réseaux on trouve les GPFN (*Gaussien Potentiel Fonction Neural Networks*) entrainnés par la méthode d'apprentissage hiérarchique de S. Lee et R.M Kill [Lee90]. Ces techniques se sont étendues maintenant aux réseaux à couches LBF [Van95].

I.6 Stabilité des réseaux de neurones

Il est connu que tout système dynamique est confronté au problème de stabilité. Dans ce sens, les réseaux de neurones notamment ceux d'architecture entièrement connectée, doivent vérifier la condition de stabilité. En effet, dans ce genre d'architecture, une fois stimulé, le réseau de neurones entre dans une phase de circulation massive d'informations entre les neurones. D'une manière générale, à chaque instant les neurones peuvent calculer leurs activations et envoyer leurs sorties vers les autres neurones ou vers l'extérieur. Il est donc important, pour que le réseau fonctionne correctement de vérifier que ce régime de circulation d'informations s'arrête en atteignant enfin la stabilité et prévoir où il s'arrêtera.

I.6.1 Fonction de Lyapunov

Vérifier la stabilité d'un réseau de neurones, c'est établir que, écarté de son état initial, celui ci évolue vers un autre état d'équilibre. Pour ce faire, un théorème utilisant le formalisme de ***Fonction de Lyapunov*** est utilisé.
Nous expliquons ci dessous le contenu de ce théorème.
Si on peut trouver une fonction $L(x_1,\ldots,x_n)$, de l'espace des états d'un système dynamique quelconque vers $\Re$, tel que n'importe quelle variation des états de ce système conduit à la décroissance de cette fonction, ce système a donc une solution stable. [Pat96].
La fonction L vérifiant ces conditions, est appelée *Fonction de Lyapunov* ou *Fonction d'Energie*. Cette fonction doit donc vérifier que $\dot{L} < 0$.

Il est à noter que trouver une telle fonction pour un système dynamique constitue une condition suffisante mais pas nécessaire pour établir sa stabilité. [Kos 92].

I.6.2 Stabilité du réseau de Grossberg

M. Cohen et S. Groosberg ont rigoureusement démontré qu'il existe une fonction globale d'énergie pour un réseau de neurones dynamique général dont l'équation décrivant son fonctionnement englobe tous les types de réseaux de neurones.
Le modèle du réseau de neurone qu'ils ont établi pour cela, est décrit par Le système suivant d'équations différentielles non linéaires continues entièrement connectées [Kar93]:

$$\frac{d u_i}{dt} = a_i(u_i)\left[b_i(u_i) - \sum_{j=1}^{n} c_{ij} d_j(u_j) \right] \qquad (1.6)$$

Où u_i représente l'activité du i éme neurone, $d(.)$ la sortie des neurones qui, par ailleurs, doit être monotone, $C = \left[c_{ij}\right]$ la matrice des poids synaptique, et $a(.)$ fonction définie positive. La somme $\sum_{j=1}^{n} c_{ij} d_j(u_j)$ représente l'entrée du *i éme* neurone de la part des neurones constituant ce réseau. La décroissance de l'activité de ce neurone dépend de la fonction $b_i(u_i)$ qui doit être, dans ce cas, inférieur à l'entrée du réseau. Le modèle de réseaux défini par l'équation (I.6) présente un cas général d'un réseau entièrement interconnecté. Les autres réseaux peuvent être identifiés comme des cas particuliers de cette équation.
pour démontrer la stabilité de ce modèle, la fonction d'énergie proposée est de la forme[sim90]:

$$V = -\sum_{i=1}^{n} \int_{0}^{u_i} b_i(\zeta_i) d_i'(\zeta_i) d\zeta_i + \frac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{jk} d_j(u_j) d_k(u_k) \qquad (I.7)$$

Pour vérifier les conditions de la fonction de Lyapunov, on a:

$$\dot{V} = \frac{dV}{dt} = -\sum_{i=1}^{n} a_i(u_i) d_i'(u_i) \left[b_i(u_i) - \sum_{k=1}^{n} c_{ik} d_k(u_k) \right]^2 \qquad (I.8)$$

Ainsi sous condition que
$c_{ij} = c_{ji}$ Ce qui revient à une matrice C des poids synaptique symétrique.
Et sachant que $a_i(u_i) \geq 0$ (les fonction $a(.)$ sont definies positives) et $d_i'(u_i) \geq 0$ (la fonction d'activation $d(.)$ est monotone), on obtient $\frac{dV}{dt} \leq 0$.
Ainsi la stabilité de cette couche de neurones entièrement connectée est démontrée
Les autres modèles de réseaux étant en général de structures relativement plus simples peuvent établir de même leur stabilité. Les fonctions de d'énergie d'autres réseaux, comme celui de Hopfield, par exemple, ne sont que des cas particuliers du modèle de Grossberg.

I.7 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.

$$\text{Apprentissage} \equiv \frac{dW}{dt} \neq 0 \qquad (I.8)$$

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 (figure I.7): l'apprentissage supervisé et l'apprentissage non supervisé. L'apprentissage par renforcement est un troisième type que nous citons qui a des points communs avec chacun des deux premiers.

I.7.1 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'intérmédiaire d'un critère à optimiser définissant la performance du réseau à chaque étape.

I.7.2 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.

I.7.3 Apprentissage par renforcement

Ce type d'apprentissage est moins "classique" que les deux premiers qui sont les principaux. Il a 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 ce type d'apprentissage, le réseau rajuste ses poids synaptiques suivant un critère de performance. Celui-ci renforce les poids du réseau si le critère y est favorable et les punit dans le cas contraire.

Au court du présent travail, nous reviendrons vers ces méthodes pour les détailler et les utiliser par la suite, dans nos applications.

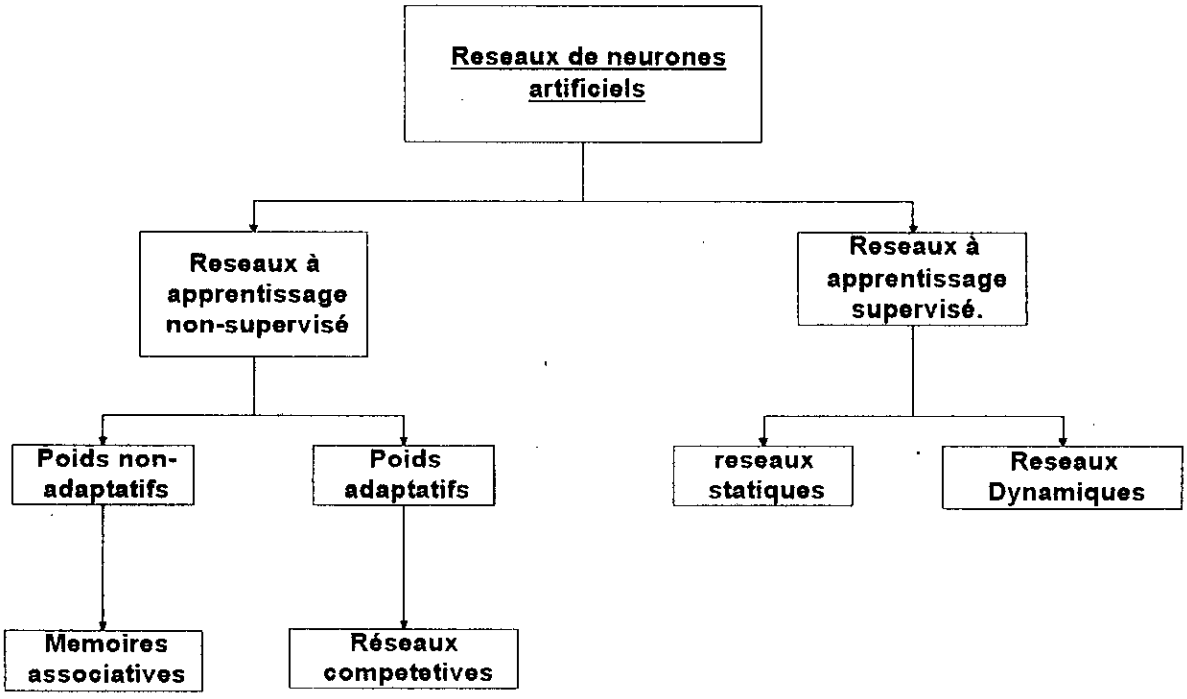


Fig.I.7 Classification des réseaux de neurones suivant leur apprentissage.

Conclusion

Dans ce chapitre, nous avons effectué une "visite guidée" dans l'univers des réseaux de neurones. Nous avons introduit des définitions et des notions, qui seront utilisées tout au long de notre étude. Plusieurs points abordés dans ce chapitre, tel que l'apprentissage ou les architectures des réseaux, seront détaillés dans les chapitres suivants.

Les réseaux de neurone constituent un véritable moyen pour la résolution de plusieurs problèmes, ou les méthodes classiques ont montré leurs limites. Leur utilisation s'est avérée efficace dans les processus qui nécessitent une interaction avec l'environnement, et ce par leur pouvoir d'adaptation souvent appelé *"Plasticité synaptique*.

De plus, l'information à l'intérieur des réseaux de neurones, étant traitée en parallèle et d'une manière distribuée, fait que le temps de calcul des opérations s'en trouve réduit. Ces caractéristiques les rendent candidats même là où une solution classique existe déjà.

La rapidité et la plasticité permettent, d'autre part, de les utiliser dans les applications qui se font en temps réel.

A partir de notions théoriques classiques en mathématique ou en traitement de signal, plusieurs améliorations ont été apportées aux réseaux de neurones. L'évolution que ces réseaux ont connues fait que l'approche visant à mimer le modèle biologique, à partir de laquelle l'idée originale des réseaux de neurones a démarré, soit abandonnée au profil de fondements théoriques solides et d'outils mathématiques, ne visant que l'efficacité et la souplesse d'implémentation [Ant90, Pat 96, Kos92].

CHAPITRE II:

RESEAUX DE NEURONES A APPRENTISSAGE NON-SUPERVISE

CHAPITRE II
RESEAUX DE NEURONES A APPRENTISSAGE NON-SUPERVISE

Introduction

Les réseaux à apprentissage non supervisé traitent des échantillons en entrée sans avoir d'informations sur les sorties que ceux ci doivent générer. Ces réseaux présentent moins de complexités de calculs, mais sont moins précis que ceux à apprentissage supervisé, que nous étudierons plus loin. L'utilisation de ces réseaux est pratique dans les problèmes qui affrontent l'environnement et nécessitent un apprentissage rapide en temps réel.
L'objectif des réseaux à apprentissage non supervisé n'est pas, en effet, de faire une approximation. Le rôle de ces réseaux est, en général, basé sur la mémorisation et la classification de leurs entrées.

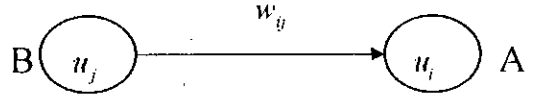
Ces types de réseaux sont les premiers à avoir bénéficié de plusieurs travaux de recherche. Après l'échec du perceptron linéaire, et avant l'apparition d'algorithmes assurant l'apprentissage supervisé des réseaux multicouches, c'est vers ces réseaux que beaucoup de travaux se sont orientés [Ant90].
Dans une première étape, qui s'étale des années 1960 jusqu'à la fin des années 1970, ces travaux ont posé les premières bases pour ce type de réseaux. Ceux ci ont cependant, été souvent limités en efficacité. En effet, l'absence d'une théorie mathématique concise sur le fonctionnement dynamique de ces réseaux posait souvent problème et limitait la portée de ces travaux. Parmi les réseaux les plus importants développés, nous citons le *Learning Matrix (LM)*. de Steinbuch en 1963, et le Linear Associative Memory (L.A.M.) de S. Anderson en 1977, qui est un réseau linéaire [Sim90] et par conséquent très sensible au bruit. Les travaux de T.Kohonen, qui est un pionnier dans ce domaine, s'étalent pratiquement sur toute cette période, et se sont révélées très utiles pour la suite de la recherche [Koh 77].
La seconde étape constitue un aboutissement qui a permis de surmonter la barrière que posait la compréhension du fonctionnement interne de ces réseaux. En effet Grâce aux travaux de Hopfield en 1982, 1984 et ceux de M.Cohen et S. Grossberg en 1983, 1988, une Théorie mathématique claire sur l'évolution interne des réseaux de neurones a été introduite. C'est, justement, ces travaux[Coh83] qui ont permis de jeter la lumière sur le comportement dynamique interne des réseaux de neurones et, de ce fait, ont donné un nouvel élan à la recherche dans ce domaine.

Ce chapitre comporte deux parties principales et une conclusion. La première traitera des mémoires associatives. Ce type de réseaux est, comme on le verra, à poids fixes. Sa classification avec les réseaux à apprentissage non supervisé n'est accréditée que par le fait que le calcul des poids de ces réseaux est basé sur les principes de l'apprentissage non supervisé; à savoir la règle de Hebb, et la théorie de corrélation.
La seconde partie sera consacrée aux réseaux compétitifs.

II.1 Fondements de l'apprentissage non-supérvisé

L'apprentissage non supervisé a, généralement, comme objectif de partager l'espace d'entrée en plusieurs classes, représentant des régions de décision différentes. ce type d'apprentissage peut être trouvé sous deux aspects. Le premier aspect représente l'idée principale sur laquelle ce type d'apprentissage est basée. C'est la **loi de Hebb**. Cette règle représentait initialement une méthode qualitative d'adaptation des poids synaptiques. Nous présentons ci dessous son contenu tel qu'énoncé par J. Hebb en 1949 [TRC89]:

Si un neurone A est stimulé à maintes reprises par un autre neurone B pendant qu'il est actif, Le neurone A devient plus sensible aux stimulations en provenance de B; Le lien synaptique de B vers A devient plus significatif. Ainsi B pourra stimuler A plus facilement à l'avenir.



Cette règle a été développée sous des aspects mathématiques qui traduisent le principe initial. La formulation mathématique la plus usitée repose sur la méthode de corrélation[Sim90].
Ainsi si w_{ij} est le poids synaptique reliant le neurone B au neurone A, La réadaptation de ce dernier peut s'écrire sous la forme discrète suivante:

$$\Delta w_{ij} = s(u_i)s(u_j) \tag{II.1}$$

où u_i représente l'activité du neurone et $s(.)$ est sa sortie, représentée par une fonction non linéaire appliquée à son activité.

Cet apprentissage peut être trouvé sous un deuxième aspect. Celui ci est lié au fonctionnement Compétitif-Coopératif des réseaux de neurones. Le principe de cet apprentissage, que nous étudierons en détail dans la deuxième partie de ce chapitre, repose sur la mise en compétition entre les neurones du réseau, afin de trouver le neurone gagnant qui représente le groupe *(Cluster)* dans lequel l'entrée sera classée.

II.2 Mémoires Associatives

II.2.1 Notions fondamentales

La mémorisation est une fonction importante dans tout système de traitement d'information.
L'idée de base des mémoires associatives tourne autour de concepts mathématiques classiques en traitement de signal. L'introduction des réseaux de neurones dans ce domaine vise à exploiter plusieurs de leurs caractéristiques, afin de rendre les opérations plus souples, plus efficaces. et plus simples à implémenter. En effet, dans ce contexte, les réseaux de neurones peuvent être perçus comme des systèmes adaptatifs pouvant remplir des fonctions clés en traitement de signal comme **la complétion** et **le filtrage**.

De nos jours le mot *"mémoire* "rime avec cette fonction que peut remplir n'importe quel calculateur. Il serait donc logique de se demander pourquoi l'utilisation des réseaux de

neurones. La réponse est que la fonction que doivent remplir ces réseaux est différente. Contrairement aux calculateurs, qui stockent des informations dans des endroits précis appelés adresses, une mémoire associative doit en plus de la mémorisation, effectuer plusieurs autres opérations de traitement.
Si une information a été correctement mémorisée par le réseau, lors de la présentation de cette information une nouvelle fois, le réseau doit la reconnaître. Plus, si cette information a été modifiée à cause de problèmes d'acquisition ou de réception, par exemple, ce réseaux doit également la reconnaître et la restituer en son état initial. Il s'agit là d'un problème de réduction de bruit ou de **filtrage**.
D'autre part, si cette même information a été amputée de certaines composantes, donnant ainsi une représentation non complète, dont la cause peut être l'interruption dans une transmission, par exemple, le réseau est appelé, encore une fois à la reconnaître et lui rendre son état initial. On parle alors de problème de **complétion**.

La mise en mémoires des informations à l'intérieur de ces réseaux se fait grâce à un apprentissage non supervisé. Pendant cette étape, appelée **Encoding Step**, le réseaux calcule les poids synaptiques W, représentant un codage de l'information à stocker.
Après cela, des exemples parfois incomplets ou modifiés, sont présentés au réseaux. Grâce à la représentation interne déduite de l'apprentissage, les neurones calculent leurs activations. A la sortie le réseaux doit se rappeler de ce qu'il a appris. Cette étape constitue ce qui est appelé **Recal Step**.

Du fait que la mémorisation s'effectue à l'intérieur des liaisons synaptiques, où l'information doit y être"gravée", ceux-ci sont souvent appelés **Long Term Memory (L.T.M.)**. L'activation des neurones, qui n'est en réalité qu'un signal éphémère ne durant pas longtemps dans la représentation interne du réseaux, est appelé **Short Term Memory (S.T.M.)**[Kos92].

Suivant le mode de fonctionnement, on distingue deux types de mémoires associatives.

Les **Mémoires Auto-Associatives** sont destinées à stocker des vecteurs prototypes $P^k = p_1^k, p_2^k, \ldots, p_n^k$ tel que $k=1,\ldots M$. représente les différentes classes.

Les **Mémoires Heteroassociatives,** par contre stockent les couples prototypes $\left(X^k, Y^k\right)$, où:

$$Y^k = y_1^k, y_2^k, \ldots, y_m^k \quad \text{avec } k=1,\ldots M.$$

A chaque vecteur X^k en entrée , cette mémoire associe le vecteur Y^k en sortuie et vice versa.

Dans ce qui suit, nous allons étudier les principaux types de mémoires associatives qui éxistent avec leur algorithmes d'apprentissage. Ceci en vu de les utiliser, par la suite, dans nos applications.

II.2.2 Réseau de Hopfield

En 1982, Hopfield a été le premier a avoir étudié la dynamique des réseaux de neurones et leur évolution vers l'équilibre. Ceci en y introduisant la notion d'énergie, en analogie avec le système magnétique de spin. Pour ce faire il a utilisé la fonction d'énergie de Lyapunov (qui est un moyen pour l'étude de la dynamique globale de systèmes d'équations non linéaires). Cette fonction a été introduite afin de décrire le processus à l'intérieur du réseau et son évolution vers l'équilibre (cf. I.6.1).
En démontrant qu'une couche de neurones entièrement connectés peut être décrite par une fonction d'énergie, le réseau est traité comme un système physique qui ne peut évoluer vers un équilibre (un point d'énergie minimale) qu'en dissipant de l'énergie. D'ailleurs une analogie peut être faite avec les systèmes mécaniques, électriques, ou thermodynamique.
Le modèle entrepris par Hopfield pour ce réseau est, en réalité, une reprise du modèle de Amari de 1977 qui était connu sous le nom de *"Digital Autocorrelator"* (D.A). Certains ouvrages désignent d'ailleurs ce réseau sous le vocable de modèle de Hopfield-Amari [Sim90].
Le **Digital Autocorelator D.A** est une mémoire auto-Associative constituée par une couche de neurones entièrement interconnectés(figure II.1.).

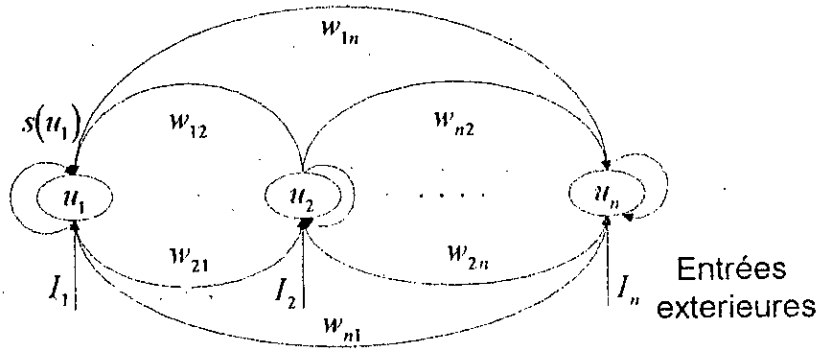


Fig II.1 Architecture d'un Digital Autocorrelator (Réseau de Hopfield).

L'originalité du réseau de Hopfield réside notamment dans son modèle électrique très connu[Fre90],. Ce modèle traduit le comportement dynamique intégrateur des neurones.
Celui ci est défini par le système d'équations suivant:

$$\dot{u}_i = \sum_{j=1}^{n} w_{ij} s_j(u_j) - b_i u_i + I_i \qquad i=1,...n \qquad \text{(II.2)}$$

où I représentent les entrées extérieures, w_{ij} des paramètres définissant la forces des liens synaptiques représentées par la conductances de ces connections, et b_i représente la conductance équivalente à l'entrée du neurone.[Ant90].
La sortie du neurone est, comme son entrée, binaire. La fonction de sortie est donc la fonction seuil définie par l'équation (II.3). Néanmoins, des modèles de ce réseaux avec une sortie continue définie par la fonction Sigmoïdes existent. Ces réseaux constituent le modèle analogique du réseau de Hopfield [Pat96].

une fonction d'énergie vérifiant les conditions de Lyapunov est définie pour la stabilité du réseau de Hopfield. Pour cela la matrice des poids W doit être symétrique et définie positive [Fre92].

S. Grossberg a établi que ce modèle n'est qu'un cas particulier de son modèle général Défini par l'équation (I.6) [Coh83].

II.2.2.1 Codage de l'information

La stabilité du réseau étant établie, il est important de définir les points où le système d'équations (II.2) va se stabiliser. La décroissance de la fonction de Lyapunov fait que ces points soient des minimums d'énergie. L'apprentissage doit donc, stocker les exemples à mémoriser dans ces minimums .
Pour cela, la matrice des poids synaptiques, qui est déduite à partir de la fonction d'énergie définie pour ce réseau utilise la règle de Hebb sous sa forme mathématique générale discrète. Ainsi l'équation (II.1) donne:

$$W = \sum_{k=1}^{M} \left(P^k\right)^T P^k \qquad \text{(II.3)}$$

où P^k, $k = 1,...M$ représentent les exemples que le réseau doit mémoriser, appelés prototypes.

Afin d'améliorer la capacité du réseaux, l'utilisation de corrélations d'ordre supérieur qui peuvent regrouper trois exemples ou plus ont été proposés, mais restent encore rarement utilisées [Gil88].
D'autre part, il est recommandé, d'utiliser les exemples binaires sous leurs formes bipolaires (0,-1). En effet, sous cette forme, le rappel se fait d'une manière plus précise et les corrélations, sont plus équilibrées. Dans la représentation binaire, par contre, les corrélations de l'équation (II.3) sont souvent biaisées et favorisent l'état haut de l'activation des neurones.
Par l'apprentissage à travers l'équation (II.3), chaque état stocké présentera une corrélation maximale avec lui même, constituant ainsi un minimum dans la fonction d'énergie appelé bassin d'attraction. Ce bassin défini, en fait, autour de lui un champ d'attraction. N'importe quel exemple qui est très proche de ce prototype, est plus fortement corrélé avec celui ci et sera vite attiré par son bassin d'attraction [Kos92].

II.2.2.2 Apprentissage et implémentation du réseau

Pour le fonctionnement de ce réseau, deux cas différents peuvent être utilisés.

Le modèle séquentiel (Simply Assynchronous Model)

Dans ce cas, qui se veut ressembler le plus au modèle biologique, un seul neurone peut calculer sa sortie à chaque itération. Afin d'éviter le phénomène d'instabilité unidirectionnelle, où le réseau pivote continuellement à l'équilibre entre deux états d'équilibre différents, Le feed-back de chaque neurone avec lui méme est supprimé [Kos92]. La matrice des poids est donc à diagonale nulle. Les poids de ce réseau sont calculés par:

$$w_{ij} = \begin{cases} \sum_{k=1}^{M} P_i^k P_j^k & i \neq j \\ 0 & \text{i = j} \end{cases} \qquad \text{(II.4a)}$$

Avec $i=1,...n$; $j=1,...n$; $k=1,..M$.
où les prototype sont utilisés sous leurs états bipolaires[Kun93].

Les étapes d'implémentation de ce réseau sont identiques au modèle synchrone, que nous présentons ci dessous, avec la seule différence qu'à chaque itération, il n'y a qu'un seul neurone qui peut changer d'état. Ainsi l'équilibre dans ce cas est plus long à obtenir. De plus, cette technique peut conduire à des états d'équilibre différents suivant l'ordonnancement des neurones qui changent d'états à chaque itération.

Le modèle synchrone(Synchronuos Model)
Ce modèle a été introduit afin d'exploiter les possibilités de calculs parallèles, notamment dans l'implémentation analogique, où tous les neurones peuvent changer leurs états en même temps. Les feedbaks entre chaque neurone et lui méme ne sont plus supprimés.
Le calcul des poids pour ces réseaux est donc défini par:

$$w_{ij} = \sum_{k=1}^{M} P_i^k P_j^k \qquad \text{(II.4b)}$$

où les prototype P_i^k, P_j^k sont utilisés sous leurs états bipolaires[Kun93].

les étapes d'implémentation de ce réseau sont les suivantes:

1. Injecter une entrée X^k au réseau.
2. Calculer l'activation de tous les neurones de la couche dynamique en parallèle:

$$u_i(t+1) = \sum_{j=1}^{n} w_{ij}\, s_j\left(u_j(t)\right) + \theta_i \qquad \text{(II.5)}$$

où θ_i représente le seuil du neurone i $(i=1,...n)$, dont les choix des valeurs offrent plus de souplesse dans l'apprentissage, comme nous l'avons constaté dans notre application présentée au chapitre III.

3. Remise à jour des états des neurones qui se fait, toujours en parallèle:

$$s_i\left(u(t+1)\right) = \begin{cases} 1 & u_i(t+1) > 0 \\ 0 & u_i(t+1) < 0 \\ s\left(u_i(t)\right) & u_i(t+1) = 0 \end{cases} \qquad i=1,...n. \qquad \text{(II.6)}$$

la fonction Sigmoïde avec un coefficient de l'exponentiel important, peut être utilisée pour un modèle analogique de ce réseau.

4. Répéter les étapes **2.** et **3.** jusqu'à ce que aucun neurone ne change d'état. C'est l'équilibre.

Pour fonctionner d'une manière adéquate, le réseau de Hopfield exige des exemples à stocker d'être orthogonaux. Dans le cas contraire, à l'introduction d'entrées bruitées, le réseau aura du mal à filtrer le signal et peut présenter des résultats erronés.
D'autre part, ce réseau est limité en nombre d'exemples qu'il peut stocker. Une augmentation de ce nombre entraîne un rapprochement des bassins d'attraction dans la fonction d'énergie; ce qui provoquera des interférences entre les zones d'influence de ces minimums et conduira à de mauvaises classifications. Il a été établi que pour garantir le bon fonctionnement du réseau, le nombre de prototypes maximal M qu'un réseau contenant n neurones peut stocker, doit nécessairement vérifier la relation[Lip87]:

$$M \leq 0.15\, n. \qquad \text{(II.7)}$$

Notre application, a montré que plus on se rapproche de ce seuil plus les résultats se dégradent. L'utilisation du modèle synchrone qui exploite les aptitudes de traitement parallèle des réseaux de neurones est préférable, et présente plus de précision dans ces résultats.

Une augmentation des capacités du réseau de Hopfield est possible par l'utilisation de neurones d'ordre supérieur, dont l'activation est une somme de produits de poids avec plusieurs sorties en même temps[Gil87]. Ce type de réseaux introduit de fortes non linéarités qui ont pour utilité le filtrage de bruit [Sim90]. Néanmoins, les propriétés de stabilité et les aptitudes de tels réseaux ne sont pas encore maîtrisées et leur utilisation qui rend les calculs plus complexes, n'est pas encore fréquente [Kar93].

II.2.3 Mémoires Associatives Bidirectionnelles (BAM)

Les BAM représentent une extension du réseau de Hopfield vers une mémoire Hétéroassociative.
Introduits par Kosko en 1988 [Kos90], ce réseau est composé de deux couches de neurones F_x et F_y. (fig. II.2). L'information à l'intérieur de ce réseau peut circuler aussi bien de F_x vers F_y. que dans le sens inverse (en feed-back).

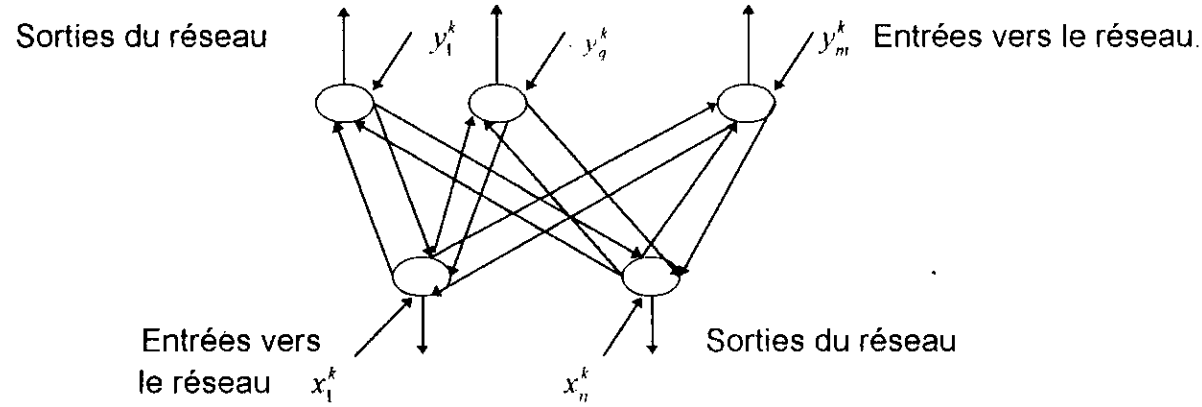


Fig. II.2 Architecture du BAM.

En mémorisant les couples d'exemples (X^k, Y^k), $k=1,...,M$, ce réseau associe aux vecteurs $X^k = [x_1^k, x_2^k, ..., x_n^k]$ les vecteurs $Y^k = [y_1^k, y_2^k, ..., y_m^k]$ et vice-versa.

Comme dans le cas du réseau de Hopfield, le codage de l'information à l'intérieur de ce réseau est basé sur la règle de corrélation.

$$W_{ij} = \sum_{k=1}^{M} x_i^k y_j^k . \qquad \text{avec: } i=1,...,n \text{ et } j=1,...,m \tag{II.8}$$

Ce réseau est, en outre, basé sur une fonction d'énergie qui garantie, sous condition que la matrice des poids synaptiques W soit symétrique, que les prototypes stockés de part et d'autre des deux couches constituent des points d'équilibres stables dans l'espace des états.

Le fonctionnement de ce réseau peut être résumé à travers les étapes suivantes :

1. Présenter une entrée à l'une des deux couches: X à la couche F_x ou Y à la couche F_y.

2. Envoyer cette entrée vers l'autre couche à travers les poids synaptiques W.
3. Calculer l'activation des neurones de la couche déstinataire et leurs sorties.
Ainsi, pour la couche F_x, on a:

$$u_i^x(t+1) = \sum_{j=1}^{m} w_{ji} s_j^y(t) + \theta_i^x \qquad \text{(II.9a)}$$

$$s_i^x(t+1) = \begin{cases} 1 & u_i^x(t+1) > 0 \\ s_i^x(t) & u_i^x(t+1) = 0 \\ -1 & u_i^x(t+1) < 0 \end{cases} \qquad \text{(II.9b)}$$

où $s_i^x(t)$ et $s_i^y(t)$ représentent les sorties des couches, respectivement, F_x et F_y; $u_i^x(t)$ et $u_j^y(t)$ représentent les activités des neurones des couches, respectivement, F_x et F_y; θ_i^x et θ_j^y représentent les seuils des neurones des couches, respectivement, F_x et F_y.

4. Renvoyer en Feed-back ces sorties vers la couche de provenance a travers W^T qui est la transposée de la matrice mémoire W (equ.II.8).
5. Calculer l'activation de chaque neurone de la couche déstinataire et sa sortie par les équation (II.9a) (II.9b) (selon la couche).
6. Répéter les étapes (3) à (5) jusqu'à ce que les neurones ne change plus d'état. C'est l'équilibre.

Les sorties des neurones sont représentées par des fonctions binaires de préférence sous leurs formes Bipolaires, afin d'obtenir une meilleure représentativité des exemples.
D'autre par, grâce à la théorie de Lyapunov, ce réseau réussit après quelques itérations à se stabiliser dans un point d'équilibre d'énergie minimale. Cet état d'équilibre est représenté par l'association (X^k, Y^k) la plus convenable et constitue le point d'équilibre le plus proche du vecteur d'entrée.

II.2.4 Analyse du fonctionnement du BAM et du réseau de Hopfield

Les BAM n'étant qu'une généralisation du réseau de Hopfield. Nous présentons, ici une analyse générale qui peut être appliquée aussi bien pour une mémoire Hétéroassociative que pour une mémoire Auto-Associative
Soit une entrée X proche d'un exemple prototype X^k de la base d'apprentissage qui est présentée au réseau. A partir de l'équation (II.8), on peut tirer l'équation décrivant la propagation de ce vecteur dans le réseau à travers le filtre synaptique W :

$$XW = C_k Y^k + \sum_{j \neq k} C_j Y^j \qquad \text{(II.10)}$$

Le cofficien C_k du premier terme du coté droit de l'équation (II.10) constitue la corrélation de l'entrée avec l'exemple qui lui est le plus proche. Ceux du second terme représente la somme des corrélations avec les autres prototypes
Du fait que l'entrée X soit plus fortement corrélée à X^k, on a $C_k > C_j$, pour $j=1,...,M$ et $j \neq k$.
Ainsi, le premier terme à droite de l'équation (II.10) constitue le signal utile représenté par Y^k.
Le second terme représente une somme de signaux constituant des bruits dont le signal d'entrée peut être entachés. Le seuilage à la sortie de chaque neurone présente une non-linéarité, qui

réduira l'effet du bruit. Plus le nombre d'exemples prototypes stockés dans le réseau est grand, plus ce terme de bruit prend de l'importance dans l'équation (II.10).
Si les exemples prototypes sont mutuellement orthogonaux, il est clair que les intercorrélations du second terme du coté droit de l'équation (II.10) aurons tendance à disparaître. D'où la convergence est assurée en une seule itération.

comme pour le réseau de Hopfield, La capacité des BAM est limitée. Une augmentation du nombre des prototypes à mémoriser amplifie les bruits (équation II.10) et rapproche les bassin d'attraction créés par ces prototypes dans la surface de la fonction d'énergie.
Kosko [Kos92] a établie qu'en général la capacité du réseau est limitée par sa dimension:

$$M \leq \min(m,n)$$

où P représente le nombre maximal de prototypes que le réseau peut stocker.
D'autre part, d'autre travaux ont montré qu'avec un choix judicieux des valeurs de seuilage θ_i cette limite peut aller jusqu'à:

$$M \leq \min\left(2^n, 2^m\right)$$

Afin d'améliorer les performances, les neurones d'ordre supérieur, avec des intercorélations d'ordre superieur pour leur apprentissage, peuvent être utilisés.[Sim90].

II.2.5 Mémoires Associatives linéaires Optimisées OLAM.

Nous avons vu plus haut que pour un bon fonctionnement d'une mémoire associative, chaque prototype stocké doit constituer un point d'équilibre qui doit, à son tour, être attractif dans l'espace de la fonction d'énergie (équ. I.7). Or le réseau de Hopfield ne Garantit cela que si les exemples sont mutuellement orthogonaux, ce qui n'est pas toujours vérifié en pratique.

Les Mémoires Associatives Optimisés constituent une alternative pour le problème d'orthogonalité. Ce réseau utilise la règle de projection pour son apprentissage.
Cette règle a été introduite la première fois par T. Kohonen [Koh77] en 1977. Elle a été ensuite reprise et développée par Personnaz et al, en 1986 pour être adaptée aux réseaux de neurones[Per86].
Le réseau de neurones basé sur cette méthode est appelé **OLAM** (*Optimized Linear Associative Memory*). Son architecture est la même que celle de Hopfield. c'est son apprentissage qui diffère.

En injectant une entrée égale à un des prototypes stockés ou proche de lui on doit retrouver ce prototype en sortie. Ceci revient à résoudre l'équation suivante:

$$WX^k = X^k \qquad k=1,..M \qquad \text{(II.11)}$$

Une solution de cette équation est

$$W = XX^{\dagger} \qquad \text{(II.12)}$$

où X est une matrice regroupant tous les vecteur prototypes à mémoriser et. $X^{\dagger}$ représente la pseudo-inverse de cette matrice.

La matrice définie par l'équation (II.12) consiste en un opérateur qui effectue la projection des entrées de l'espace $\Re^n$ vers le sous espace $\Im$ généré par les prototypes à mémoriser:

$$\Im = spann\left[P^1, P^2, \ldots, P^M\right] \tag{II.13}$$

Cette méthode est une extension du théorème de Pythagore dans l'espace[Kos92]. Selon cette méthode l'espace est découpé en deux sous espaces, dont les opérateurs de projection vers chacun d'eux sont, respectivement, $W = XX^{\dagger}$ pour l'espace $\Im$ généré par les prototypes, et $\left(I - XX^{\dagger}\right)$ pour le second sous espace $\Im^{\perp}$ qui est orthogonal au premier [Koh77].

Ainsi, n'importe quel vecteur v dans l'espace peut se décomposer en deux vecteurs orthogonaux $\hat{v}$ et $\tilde{v}$, composant les cotés du triangle de Pythagore.
Par cette méthode, la matrice des poids effectue la projection des entrées dans le sous espace $\Im$. Ce qui donne sa composante $\hat{v}$ constituant le signal. La seconde composante $\tilde{v}$ de ce vecteur, ne constitue, dans ce cas, que du bruit appartenant au second sous espace.

La matrice des poids peut être calculée par le procédé de pseudo-inversion matricielle de Moon-Penrose qui donne [Kos92]

$$X^{\dagger} = \left(X^T X\right)^{-1} X^T \tag{II.14}$$

d'où l'on a:

$$W = X\left(X^T X\right)^{-1} X^T \tag{II.15}$$

Le calcul de cette pseudo-inverse peut aussi être effectué par le procédé itératif de Greville[Per86].
Notons que dans le cas particulier, où les prototypes sont orthogonaux nous obtenons aisément à partir de l'équation (II.15): $W = XX^T$ Ce qui revient aux Mémoires Associatives Linéaires.

La règle d'apprentissage basée sur la projection constitue une amélioration par rapport au réseaux de Hopfield, dont l'apprentissage est basé sur la technique de corrélation. Concernant la capacité du réseau, il a été établi par Personaz que l'attractivité d'un vecteur prototype diminue considérablement, en général, quand le rapport M / n (Le nombre de prototypes / le nombre de neurones), est de l'ordre de 0.5. Ceci constitue une grande amélioration par rapport au réseau de Hopfield dont la limite de capacité de stockage définie par la relation (II.7) est largement inférieur [Ant90].

II.2.6. Reseau de Hamming

Le réseau de Hamming est une mémoire Auto-Associative qui est particulièrement efficace pour les problèmes de bruit, qui sont en fait, très fréquent en pratique (communication , transmission, ...). Ce problème a en effet, causé un sérieux handicap pour plusieurs types de mémoires associatives, notamment les réseaux linéaires [Kun 93] et sa résolution est d'une très grande importance.

L'introduction de ce réseau a été faite à la fin de la dernière décennie et au début des années 90. Lippman, qui a beaucoup donné dans le domaine des algorithmes d'apprentissage, a publié un algorithme pour l'entraînement de ce réseaux en 1987 [Lip87, Hus95].

Le concept de base de ce réseau s'articule autour du principe de distance de Hamming. Appliqué en classification, cette mesure de distances entre vecteurs dans l'espace est exploitée pour donner la méthode connue sous le nom du **Optimum Minimum error Classifier**.
L'idée est d'utiliser la distance de Hamming à chaque fois qu'un exemple est injecté afin de calculer sa distance par rapport aux états mémorisés. C'est l'état le plus proche de cet exemple parmi les prototypes stockés qui sera délivré en sortie.

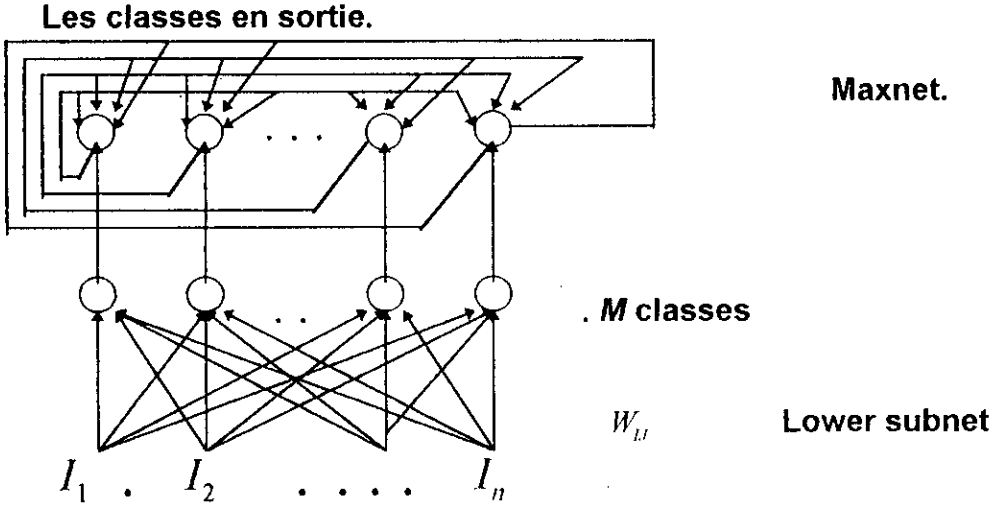


Fig. II.3 Architecture du réseau de Hamming

L' architecture de ce réseau est présentée sur la figure (II.2). Celui ci est constitué de deux sous réseaux (Subnets): Le **Lower Subnet** et le **Maxnet.**

La couche d'entrée est formée de n neurones, constituant la dimension de l'espace d'entrée.
la deuxième couche du Lower Subnet est constituée de m neurones représentant chacun un des états prototypes mémorisés.
Le Lower Subnet constitue la partie dans laquelle la position de l'exemple d'entrée vis à vis de chaque état stocké est calculée .Les sorties des neurones de cette couche sont les corrélations de l'entrée avec chaque classe. Ce Subnet délivre les **Matching Scores**.
Le Maxnet est un réseau qui, a partir de plusieurs entrées, délivre en sortie celle qui est supérieur à toutes les autres. Il est utilisé pour déterminer le neurone qui a le plus fort score. A partir de la détermination de la classe gagnante, l'exemple d'entrée est remplacé en sortie par le prototype qui lui est le plus proche.

II.2.6.1 Apprentissage et fonctionnement du réseau

La mémoire de ce réseau réside dans les poids synaptiques où chaque prototype est mémorisé:

$$w_{ij} = \frac{p_j^i}{2} \qquad \text{(II.16)}$$

où $P^i = \left(p_1^i, p_2^i, \ldots, p_n^i\right)$, $i=1,\ldots,M.$ et $j=1,\ldots,n.$, représentent les prototypes à mémoriser.

De cette manière, la sortie de chaque neurone de cette couche est La corrélations de l'entrée avec le prototype représenté par ce neurone. C'est le score du neurone.

Chaque neurone du Lower Subnet calcule sa sortie qui constitue son *Score (Matching Scor):*

$$S_i(k) = f\left(\sum_{j=1}^{n} w_{ij} I_j - \theta_i\right) \qquad i=1,\ldots M \tag{II.17}$$

où $f(.)$ est la fonction Signum et θ_i le seuil du neurone i de cette couche qui peut être chosi le même pour tous les neurones $\theta_i = \frac{n}{2}$ [Lip87].

Les scores de chaque classe étant disponibles, ces sorties sont transmises vers le Maxnnet qui constitue "le juge de la compétition".
La technique suivant laquelle fonctionne cette couche est propre aux réseaux compétitifs. En effet, ayant plusieurs entrées, cette couche doit déterminer lequel, parmi ces neurones, a la plus forte activité. Le gagnant sera renforcé pendant que les autres seront, au contraire, inhibées. C'est pour cela que cette méthode est souvent appelée *The Winner Take All.*
Cette couche est dynamique et chaque unité représente une classe. Les poids synaptiques de cette couche récurrente sont calculés par la technique D'*Inhibition latérale*. où chaque neurone renforce son activité par un feed-back excitateur et inhibe les autres neurones autour de lui.
Le calcule des poids de cette couche est donc:

$$t_{jl} = \begin{cases} 1 & j = l \\ -\varepsilon & j \neq l \end{cases} \qquad j, l=1,\ldots,M. \tag{II.18}$$

Avec $\varepsilon < \frac{1}{M}$.

La sortie de chaque neurone de cette couche est donc:

$$S_i(t+1) = f\left(S_i(t) - \varepsilon \sum_{j \neq i} S_j(t)\right) \tag{II.19}$$

A l'équilibre, cette couche délivre le gagnant qui est le seul neurone à rester activé.

On résume le fonctionnement de ce réseaux dans les étapes suivantes:

1. Injecter un nouvel exemple dans la couche d'entrée.
2. Chaque neurone du Lower-Subnet calcule sa sortie par l'équation (II.17).
3. Chaque neurone du Maxnnet reçoit la sortie de la classe correspondante.
4. Afin de tirer la classe gagnante, cette couche dynamique évolue suivant l'équation (II.19). jusqu'à l'équilibre où un seul neurone reste activé. C'est le prototype qu'il représente qui a gagner la compétition.

Notre utilisation de ce réseau a prouvé son efficacité et son aptitude à filtrer les exemples du bruit. De plus, sa technique de mémorisation et celle de rappel, qui évitent les interactions entres les états d'équilibres, lui confèrent une meilleure capacité de stockage par rapport aux autres réseaux

II.3 Réseaux Compétitifs

Les réseaux de neurones que nous avons étudiés jusqu'à maintenant, sont à poids fixes. Ils ne peuvent pas s'adapter à un changement d'environnement (création de nouvelles classes). Face à de nouvelles situations, ces réseaux s'avèrent inefficaces. Ainsi, des réseaux compétitifs, qui s'adaptent au fur et à mesure au cours de leur utilisation, ont été élaborés.
Les modelés d'entraînement compétitif, où l'adaptation des poids se fait suivant un apprentissage non supervisé ont été utilisées.
Les travaux les plus importants dans ce domaine, sont ceux de A.Cohen [Coh83] et ceux de S.Grossberg, qui sont les initiateurs de ces types de réseaux et à l'origine de plusieurs algorithmes depuis les années soixante [Sim 90].
Sont encore à citer, les travaux de T. Kohonen qui a développé des réseaux très efficaces tels que le *Self Organization Map (SOM)* et le le *Vector Quatizatiser(VQ)*.[Koh88].

D'une manière élémentaire ces réseaux contiennent, en plus de la couche d'entrée, une couche compétitive, où les neurones entrent en compétition entre eux, afin de déterminer lequel est le plus représentatif de la classe de l'exemple d'entrée. Au bout de la compétition la règle du **Winner Take All** où le neurone gagnant, représentant la classe à laquelle le vecteur d'entrée devrait appartenir, se voit attribuer la valeur maximale "1" comme état et les autres seront inhibés.
Le fonctionnement de tels réseaux est basé sur La division d'un espace d'exemples en plusieurs classes, où les membres d'une même classe montrent un certain degré de similarité entre eux. Parmi les critères utilisés pour mesurer le degré de similarité, on retrouve la **distance euclidienne** et l'**intercorrélation**.

Nous présentons ci dessous le modèle classique de couches de neurones compétitives qui sont utilisées dans ces types de réseaux notamment en implémentation:

II.3.1 Couche de Grossberg

Inspirée du modèle de Grossberg de 1968 cette couche est constituée de neurones entièrement interconnectés (Figure II.1.).

La dynamique des neurones constituant le modèle continu de cette couche est régie par l'équation différentielle suivante [Sim90]

$$\dot{u}_i = -\mu u_i + \delta \sum_{j=1}^{n} s(u_j) w_{ij} + I_i \qquad i=1,\ldots,n \qquad \text{(II.20)}$$

où: u_i représente l'activité du i éme neurone.
I_i représente l'entrée extérieure vers ce neurone.
μ est une Constante positive contrôlant la décroissance de l'activaté du neurone.
δ est une Constante positive contrôlant les interconnections latérales entre les neurones.
S(.) est La sortie du neurone, qui fait une opération de seuillage. D'où le choix de la fonction sigmoïde dans le cas analogique.

Le fonctionnement de cette couche compétitive est basée sur Deux techniques importantes: La **Self-Exitation**, où chaque neurone renforce son activité à travers le feed-back qu'il se renvoie vers son entrée, et la **Neighbor Inhibition**, où chaque neurone inhibe les autres neurones du réseau. Cette dernière opération est, pour cela, appelée la **Lateral Inhibition**.

Lors de la présentation d'une entrée X^k au réseau, La dynamique de l'équation (II.20) fait que le neurone de la classe à laquelle cet exemple est le plus proche est activé au maximum. A l'équilibre, c'est le seul qui reste activé.
Ce neurone gagnant est le seul à se voir réadapter ses poids. L'apprentissage se fait de la manière suivante [sim90]:

$$\dot{w}_{ij} = s(u_i)\left[-\alpha w_{ij} + \beta s(u_i)\right] \qquad \text{(II.21.a)}$$

où:
α: est un paramètre contrôlant le régime transitoire.
β: Le taux d'apprentissage.

Le défaut de ce modèle est qu'il traite le bruit de la même manière que le signal. En effet L'équation (II.20), qui est linéaire, ne permet aucun filtrage du bruit. Ceci conduit le modèle rapidement vers la saturation, dés que les entrées ont des valeurs très importantes. Ce qui constitue le dilemme connue *"Noise - Saturation"* [kos92].
pour cela, S. Grossberg a introduit un autre modèle plus robuste au bruit appelé **Shunting Grossberg.**

L'activité des neurones de cette couche est définie par l'équation suivante [Fre 92]:

$$\dot{u}_i = -\alpha u_i + (\beta - u_i) + \left[s(u_i) + I_i\right] - (u_i + \mu)\left[\sum_{j \neq i} s(u_j) w_{ij} + J_j\right] \qquad \text{(II.21.b)}$$

Où:
β est une constante contrôlant l'entrée excitatrice I_i.
μ constante contrôlant l'entrée inhibitrice J_i. et le feed-back latéral défini par la somme présente dans le dernier terme du coté droit de l'équation (II.21b).
Grossberg a démontré que face à des exemples bruités, ce modèle évite la saturation, tout en restant sensible aux entrées. Ceci en introduisant les deux termes $(\beta - u_i)$ et $(u_i + \mu)$ appelés *Shunting Terms*. Ce processus constitue une "balance" entre entrées et feed-backs et peut être résumé par l'équation : $\dot{U}$ =*decroissance+Exitation-Inhibition*.

L'apprentissage de cette couche se fait de la même manière que l'additive Grossberg.

Enfin, ces deux modelés, définis par les équations (II.21.a) et (II.21.b), constituent un cas particulier du modèle général de Grossberg que nous avons déjà présenté (equ.I.6). Leur stabilité est donc garantie. C'est les modelés continues, sur lesquelles le fonctionnement des réseaux compétitives sont basés.

Dans ce qui suit nous allons faire l'étude des principaux réseaux compétitif, ainsi que leurs algorithmes d'apprentissage.

II.3.2 Adaptive Resonance Theory

Les réseaux dits ART, doivent leur nom à la façon interactive, dont l'apprentissage et l'entraînement interagissent dans leur fonctionnement. Ce type de réseau est basé sur l'idée initiale *Adaptive Resonance theory* introduite par Grossberg en 1976, qui a été reprise par Carpenter et Grossberg en 1987 pour former ce réseau [Ant90].

Le fonctionnement du ART constitue une analogie avec la résonance physique. En effet, a partir d'un faible signal, l'information fait des " Aller - Retour " entre la couche d'entrée et celle de sortie, constituant des *revibrations*, qui continuent jusqu'à ce que celle ci entre en résonance soit en adhérant à une classe déjà existante, soit en créant une nouvelle classe, suivant la finesse de la classification.
Ce réseau est un *Nearest Neighborhood Classifier* à deux couches. Son entraînement se fait En ligne et est basé sur le *clustering* (regroupement).

II.3.2.1 Architecture et principe de fonctionnement

Le réseau initialement proposé par *Carpenter et Grossberg* était constitué de deux couches, dont le fonctionnement est basé sur le Shunting Grossberg. Le traitement de l'information est guidé par un troisième sous système appelé *Gain Control*, dont le rôle est d'inhiber ou de réactiver les neurones et d'organiser la circulation de l'information entre les deux couches. A cela s'ajoute un quatrième sous système appelé Orienting Subsystem qui constitue l'arbitre de la compétition [Fre92].

Beaucoup de modifications ont été apportées à ce modèle. Nous allons, dans ce qui vient, schématiser d'une manière plus simple le fonctionnement du réseau qui est adopté aujourd'hui, ainsi que son mode d'apprentissage.

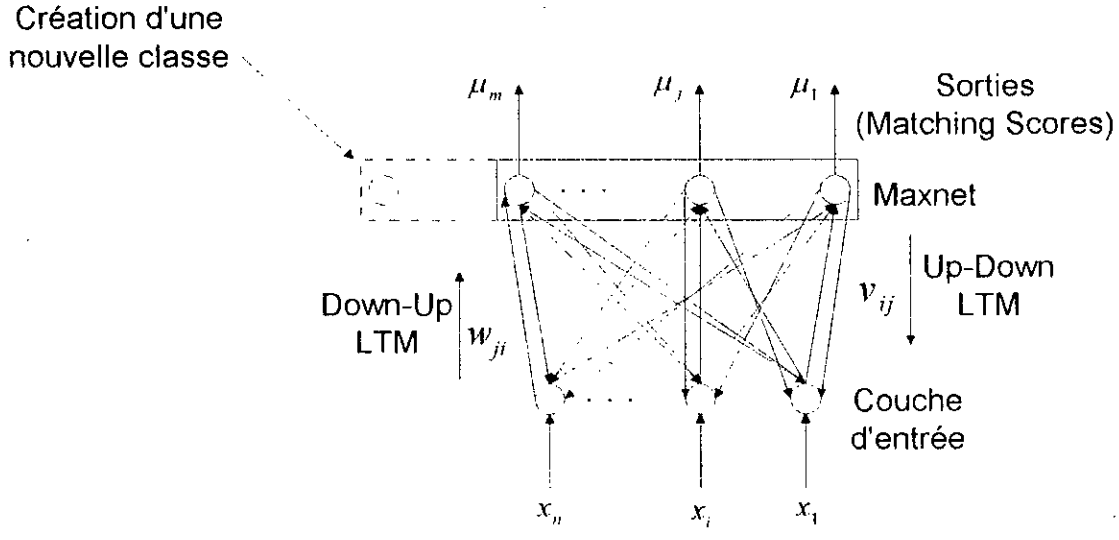


Fig. II.4 .Architecture d'un réseau de neurones ART

La figure II.4 présente l'architecture d'un ART. Celui ci est constitué de deux couches: Une couche d'entrée ordinaire et une couche de sortie appelée Maxnet, dont les neurones sont entièrement interconnectés [Pat96].
Chaque neurone de cette dernière couche représente une classe. Lorsque une entrée X est présentée au réseau, celle ci est transmise vers la couche supérieure via des liens synaptiques allant dans le sens direct appelés **Bottom-up LTM.** Chaque neurone du Maxnet calcule son activation. Le neurone ayant ses poids les plus corrèles avec l'exemple en entrée, présente le plus grands résultat en sortie. Celui ci sera candidat pour être désigné comme étant la classe à laquelle cet exemple appartiendrait. Avant de prendre cette décision, l'exemple X est renvoyé vers la couche d'entrée via les liens synaptiques qui vont dans le sens inverse appelés **Top-Down LTM**. Là, un test, dit *de vigilance*, est appliqué afin d'estimer à quel point cet exemple ressemble à ceux appartenant à la classe gagnante. Si celui ci réussi le test, l'exemple est accepté. Sinon, le neurone gagnant est disqualifié et une autre classe est recherchée. Si cette exemple ne se trouve pas une classe satisfaisant le test de vigilance, il créera sa propre classe qui est indépendante de celles déjà existantes [Kun93].

Pour l'implémentation du réseau, deux cas différents ont été introduits. Le premier, qui est juste utilisé pour des entrées binaires, est appelé **ART1.** Le second, appelé **ART2,** est utilisé pour la classification d'exemples quelconques.

Nous présentons ici les étapes de fonctionnement de chacun de ces deux réseaux, en utilisant le Fast-Learning pour l'adaptation rapide de leurs poids, qui se fait en temps réel.

II.3.2.2 Algorithme de ART1

Ce réseau est destiné à la classification d'exemples binaires seulement[Lip 87].
Ses étapes sont comme suit:

1. Initialiser les Top-Down et les Bottom-up LTM (poids synaptiques).Pour cela, Lippman propose de poser, respectivement [Lip87], $v_{ij}(0) = 1$ et $w_{ji}(0) = \frac{1}{n}$. $\quad 1 \le i \le n \quad et \quad 1 \le j \le m.$

Fixer le seuil de vigilance ρ tel que $0 \le \rho \le 1$.
2. Appliquer l'entrée.
3. Calculer les Matching Scores (le score de chaque neurone):

$$\mu_j = \sum_{i=1}^{n} w_{ji} x_i \quad 1 \le j \le m \tag{II.22}$$

où μ_j est l'activation du neurone j et x_i est le i éme élément du vecteur d'entrée X.
4. Le Maxnet sélectionne le neurone gagnant dans la compétition par le critère:

$$\mu_{j^*} = \max_{j} \{\mu_j\}. \tag{II.23}$$

Ceci peut être fait en utilisant la technique d' *Inhibition Latéral Intensive* que nous avons déjà expliquer avec détail (cf. II.3 et II.2.6.1)
5. Appliquer le test de vigilance:

- Si $\dfrac{\sum_{i=1}^{n} v_{ij^*} x_i}{\sum_{i=1}^{n} x_i} > \rho$, (II.24)

  le neurone j^* gagne la compétition, il est mis à 1.

- Sinon aller à *6.*

6. Le neurone gagnant a échoué au test en *4.*. Ce lui ci est remis à zéro (la mise hors compétition). La compétition est relancée sans lui, afin de chercher un autre gagnant.

- Si aucun des neurones déjà existant ne réussi le test, aller à l'étape *7.*
- sinon, Aller à *4.*

7. Un nouveau neurone représentant la classe de l'exemple injecté est créé. Aller à **2.**.

8. Réadapter les poids:

$$\bullet\ w_{j^*i}(k+1) = \frac{v_{ij^*}(k)x_i}{0.5 + \sum v_{ij^*}(k)x_i} \tag{II.25}$$

$$\bullet\ v_{ij^*}(k+1) = v_{ij^*}(k)x_i \tag{II.26}$$

9. Lever la remise à zéro des neurones concernées et revenir à. **2.**

Pour la classification d'exemples analogiques, nous présentons un algorithme qui est plus général et dont l'implémentation est plus simple.

II.3.2.3 Algorithme du ART2

Pour cet apprentissage on garde la même architecture du réseau. Cependant, les poids synaptiques allant de la couche d'entrée vers la couche supérieur (Bottom-Up LTM) et ceux du sens inverse (Top-Down LTM) sont égaux. De plus, les entrée ne sont pas obligatoirement normées [Kun93]. La sélection du neurone gagnant est en général faite suivant la distance Euclidienne.

Nous présentons ci dessous les étapes de fonctionnement et d'apprentissage du **ART2.**

1. Initialiser les poids synaptiques entre les deux couches.
2. Appliquer une nouvelle entrée X.
3. Calculer l'activation de chaque neurone de la couche supérieur:

$$\mu_i = \|X - W_i\| \tag{II.27}$$

4. Déterminer le neurone gagnant. C'est celui qui a l'activité minimal parmi les autres. La couche supérieure, dans ce cas, est un MINNET(Une couche qui cherche un minimum).
5. Faire passer au neurone gagnant le test de vigilance.

$$\|X - W_{i^*}\| < \rho \tag{II.28}$$

où le paramètre de vigilance ρ détermine le rayon de la classe.

- Si le neurone échoue au test un nouveau neurone représentant la classe de l'exemple X est créé automatiquement. Les valeurs de ses poids sont égales au vecteur d'entrée lui même.

$$W_k = X \tag{II.29}$$

• Si le neurone vérifie la condition du test, l'exemple est rangé dans sa classe. Les poids du neurone sont réajustés de sorte qu'il représentent les coordonnées de l'épicentre de la classe:

$$W_i^{(new)} = \frac{X + W_i^{(old)} \left\| Groupe_i^{(old)} \right\|}{\left\| Groupe_i^{(old)} \right\| + 1} \qquad \text{(II .30)}$$

où $\|Groupe_i\|$ signifie le nombre de membres appartenant déjà à la classe i.

Le fonctionnement de ce réseau et son apprentissage se faisant en même temps, la meilleur manière de procéder est, généralement, de fire démarrer le le réseau avec un seul neurone dont le vecteur poids est égal au premier exemple présenté. L'architecture du réseau croit avec le nombre de classes[Kun93].

II.3.3 Self Organized Feature Map de Kohonen

L'organisation des neurones à l'intérieur du système nerveux reflète généralement, les caractéristiques physiques du signal capté. Dans le système d'audition, par exemple, les neurones sont organisés suivant les fréquences qui génèrent les plus fortes réponses sur chaque groupe de neurones[Fre92].

S'inspirant de cette organisation, l'idée de base de ce réseau est l'introduction dans l'apprentissage compétitif de paramètres, rendant chaque neurone dépendant des neurones qui lui sont voisins. De cette manière, les neurones les plus proches topologiquement, sont sensibles aux mêmes caractéristiques.
Un modèle de ce type a été présenté, pour la première fois, par T. Kohonen en 1982. Dans les années 90, avec le développement des algorithmes de quantification vectorielle[Koh88], ce réseau a connu un regain d'intérêt, et de nombreux chercheurs ont proposé des raffinements pour son apprentissage.[Hus95]

II.3.3.1 Architecture et fonctionnement du réseau

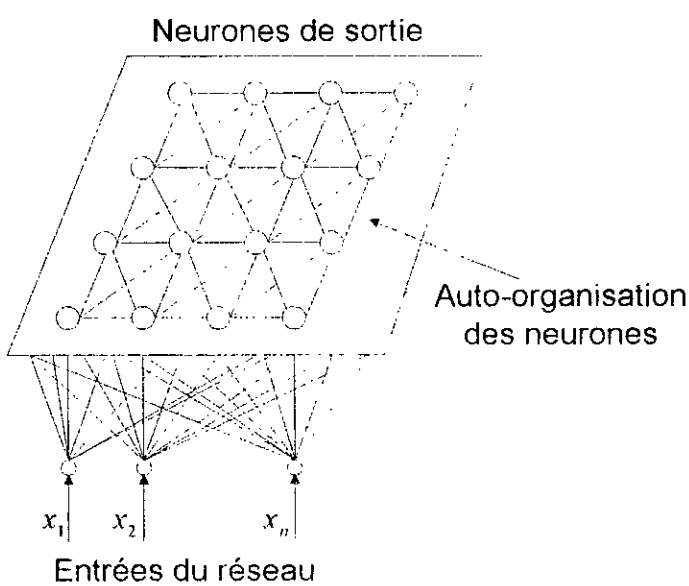


Fig. II.5. Architecture des SOM de Kohonen

L'entrée de ce réseaux est un vecteur à n composants connectés tous aux k neurones de la couche de sortie. L'organisation de cette couche de neurones peut être présenté comme une grille à deux dimensions (ou à une seule dimension) dont tous les noeuds reçoivent les n composants des entrées du réseaux (figure II.5). A l'intérieur de cette grille, chaque neurone a des voisins avec lesquels, il est relié par des connections récurrentes selon un noyau de convolution décroissant de forme qui est, généralement, schématisée par un *"chapeau Mexicain"* (figure II.6). Ainsi, plus le voisin est éloigné, moins les connections sont fortes, jusqu'à devenir inhibitrices.

Les poids reliant les entrées aux sorties, formeront des centres de groupes de vecteurs d'entrée. Ces poids doivent, en fait, échantillonner l'espace d'entrée en plusieurs classes.
Ce réseau se caractérise par rapport aux autres par son aptitude à faire apparaître les caractéristiques liant les exemples en entrée. Cette faculté est due à son mode d'apprentissage compétitif qui est spécifique.
En effet, pendant l'apprentissage, grâce aux paramètres définissant le *neighborhood*, le neurone gagnant dans la compétition n'est cette fois ci, pas le seul à réadapter ses poids. En effet, Ce neurone doit "partager" sa victoire avec les neurones de son voisinage qui bénéficieront eux aussi de l'apprentissage. Ce qui fait qu'à la fin, les neurones du réseau deviennent naturellement ordonnées.

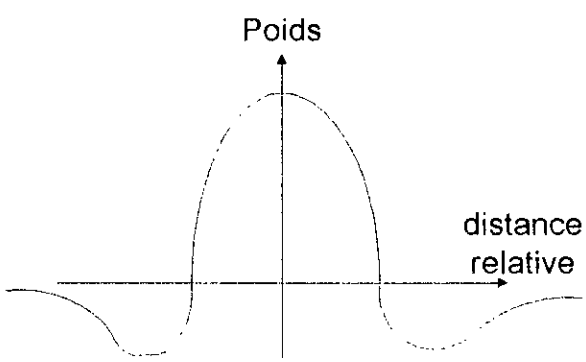


Fig.II.6 Noyau des connections récurrentes: Le chapeau Mexicain
La décroissance des poids avec l'augmentation de la distance spatiale par rapport au neurone

Nous présentons ci dessous les étapes de fonctionnement et d'apprentissage de ce réseau:

1. Initialiser les poids allant des n neurones d'entrée vers les m neurones de sorties à de petites valeurs aléatoires.
2. Présenter au réseau une nouvelle entrée X choisie aléatoirement.
3. Chaque neurone calcule sa sortie. La fonction de sortie est radiale c'est la distance euclidienne de l'exemple d'entrée par rapport à ses poids synaptiques:

$$\mu_i = \sum_{j=1}^{n} \left(x_j - w_{ij}\right)^2 \qquad 1 \le i \le m \tag{II.31}$$

où x_j représente le j éme composant de l'entrée X et w_{ij} est le poids synaptique liant cette entrée au *i éme* neurone.

La sortie peut être calculée par la Sommation pondérée, à condition que les entrées soient normées.

4. sélectionner le neurone à distance minimale par rapport à l'entrée.

Dans le cas de sommation pondérés, on cherchera dans cette étape un maximum. C'est le neurone dont le vecteur poids est le plus corrélé avec cette entrée.

Le neurone gagnant est noté i^*.

5. définir un intervalle de *voisinage* (*Neighborhood*) autour du neurone gagnant. Les poids associés à ce neurone et aux neurones de son voisinage sont réadaptés:

$$w_{kj}(t+1) = w_{kj}(t) + \eta(t)\psi(k,i^*)(x_j(t) - w_{kj}(t)) \qquad \text{(II.32)}$$

où $\eta(t)$ représente le taut d'apprentissage et $\psi(k,i^*)$ représente le voisinage sur lequel s'applique la modification des poids. Dans le cas le plus simple $\psi(k,i^*) = \delta_{ki}$. Avec δ le symbole de Kröneker

6. Réadapter le taux d'apprentissage η en le faisant décroître.

Calculer le nouvelle étendu du voisinage ψ qui doit diminuer, lui aussi, avec l'apprentissage.

7. Retour à l'étape **2.** jusqu'à ce que le test d'arrêt est vérifié.

avec cette manière de procéder, le réseau s'organise topologiquement puis fait converger les poids vers les barycentres des vecteurs de chaque classe.

Pour obtenir une convergence rapide et "presque" sure le pas d'adaptation doit diminué. Typiquement, un choix d'un pas décroissant à raison de $1/t$ est généralement convenable. En outre, la fonctions *Neighborhood* choisie " *Le chapeau Mexicain* " doit faire diminuer l'étendue du voisinage au fur et à mesure que l'apprentissage avance.

A la fin de cette étude sur cette famille de réseaux de neurones, Il est important de noter qu'en plus des réseaux que nous avons étudiés dans ce chapitre, il existe d'autres qui sont moins utilisés:

Le Conterpropagation de Hecht-Nielsen, 1988 est une mémoire Héteroassociative qui combine deux types de réseaux différents; le modèle de *Kohonen* utilisé pour sa couche d'entrée effectue une classification basée sur son fonctionnement compétitif. Le modèle de *Grossberg* est utilisé dans sa couche de sortie. A partir du neurone gagnant de la première couche, ce dernier déduit la sortie du *Conterpropagation*[Fre92].

Le Néocognitron de K. Fukushima, 1983 est un réseau inspiré du système visuel humain. Ce réseau est constitué de plusieurs couches de neurones divisées chacune en plusieurs petites couches. Son rôle est de reconnaître les formes d'objets apprises lorsque ces derniers ont subi des modifications dans leurs dimensions où des translations ou des rotations dans un plan. Pour pouvoir effectuer cette reconnaissance, ce réseaux nécessite un nombre très important de neurones, une seule de ces couches peut en effet, en contenir plusieurs centaines [Fuk 83].

Conclusion

Dans ce chapitre, nous avons procédé à une étude des principaux réseaux à apprentissage non supervisé. Nous avons réparti ces réseaux en deux familles différentes:

Les mémoires associatives qui sont des réseaux dont les poids synaptiques sont calculés une seule fois et ne s'adaptent plus.

Les réseaux compétitifs qui, eux par contre, sont adaptatifs et peuvent effectuer un apprentissage en temps réels.

Les mémoires associatives sont caractérisées par leur simplicité en implémentation. Les réseaux basés sur le produit de corrélation (Hopfield et BAM) sont des réseaux efficaces. Cependant, ils sont limités en capacité notamment lorsque les entrées ne sont pas mutuellement orthogonales ou sont entachées de bruit. Ils ont cependant, l'avantage de pouvoir faire l'apprentissage de nouveaux exemples, par simple rajout de nouveaux termes dans la matrice des poids W, sans avoir à recalculer tous les poids synaptiques.

Les réseaux basés sur la règle de projection OLAM représentent une bonne alternative pour le problèmes d'orthogonalité des entrées et de capacité. Ils sont néanmoins moins souples dans leur apprentissage. En effet, pour accepter de nouveaux prototypes toute la matrice des poids doit être recalculée. Cela signifie qu' un nouveau réseau de neurones doit être reconçu. Cependant la capacité de ces réseaux est plus importante que celle du réseau de Hopfield.

Le réseau de Hamming utilise une autre approche qui est inspirée des réseaux compétitifs. Ce réseaux est plus efficace contre les effet du bruit et son mode de fonctionnement permet de rendre sa capacité plus importante.

Les réseaux compétitifs sont une extension des mémoires associatives vers des réseaux dont l'architecture évolue en fonction des classes en temps réel. Ces réseaux sont, en outre, plus sélectifs en classification, moins sensibles au bruit et. plus adaptés à un environnement changeant.

Pour clore ce chapitre, il est important de mentionner que les familles de réseaux de neurones étudiées, trouvent de larges applications en traitement de signal. Ces réseaux sont bien indiqués pour les problèmes de transmission où il est question de filtrage de bruit, de traitement d'images, et notamment de reconnaissance.

CHAPITRE III:

RECONNAISSANCE DE FORMES PAR RESEAUX DE NEURONES

Chapitre III
RECONNAISSANCE DE FORMES PAR RESEAUX DE NEURONES

Introduction

L'opération de reconnaissance constitue l'une des applications les plus connues en réseaux de neurones. La plus part des premières applications utilisant les réseaux de neurones étaient destinées à la reconnaissance et la classification [Koh77].

A leurs débuts, Ces applications ont connu de sérieuses difficultés devant la non-inéarité des classes, que le perceptron ne pouvait affronter. L'utilisation des réseaux dynamiques en tant que mémoires associatives n'a, non plus, pas résolus la question à cause de phénomènes d'instabilité, qui n'ont été maîtrisés qu' à partir de 1982, après les études de Hopfield et ceux de Grossberg [Ant90]. Malgré cela, la capacité de stockage de ces réseaux est restée limité. En 1984 Hopfield et Tank [Kar93] ont utilisé le réseau de Hopfield pour concevoir leur convertisseur Analogique-Digital. Ce réseau a été également utilisé notamment par Hopfield lui même et d'autres après lui, pour résoudre le fameux problème du vendeur ambulant (*The Traveling Salesman Problem*) [Fre92].

Au fil des années, la question de reconnaissance a pris une autre signification. Celle ci consiste à concevoir des systèmes de mémorisation robustes aux bruits et autres déformations qui peuvent survenir sur les signaux à reconnaître, et de les corriger. Actuellement, encore, ces problèmes demeurent les plus importants et c'est à travers eux que l'on peut juger l'efficacité des réseaux de neurones appliqués en reconnaissance.

L'apparition de l'algorithme d'apprentissage pour le perceptron multicouche a orienté la plupart des chercheurs à la fin des années 1980 vers l'utilisation de ce réseau pour les problèmes de reconnaissance.

Parmi les applications que nous pouvons trouver dans la littérature, J. Burr a conçu en 1988 un réseau à couches entraîné par la Backpropagation, dont le rôle était de classifier les caractères écrits. Mais les entrées du réseau nécessitaient toujours un prétraitement; chaque caractère est d'abord projeté sur un code à 13 barres, similaire au principe du digit à 7 barres, qui présente les chiffres de 0 à 9. C'est cette projection qui est injectée au réseau qui ne pouvait reconnaître que ces formes [Kar93].

Dans les applications de reconnaissance il est question de mémorisation et de classification, plutôt que de généralisation ou approximation. De ce fait, les réseaux à apprentissage non supervisés sont très indiqués pour ces problèmes. K. Fukushima a utilisé son réseau, le Néocognitron, [Kun83] en 1987 afin de concevoir un système de reconnaissance de chiffre, qui était inspiré du système visuel biologique. Ce réseau était par conséquent, constitué de plusieurs couches dont le nombre de neurone d'une seule couche pouvait atteindre plusieurs centaines [Kun87].

Dans ce chapitre, nous allons mettre en oeuvre quelques réseaux étudiés dans le chapitre précèdent pour le problème de reconnaissance de formes. Nous ferrons une application sur la reconnaissance de caractères latins de l'alphabet puis sur les chiffres arabes.

L'objectif que nous visons à travers cela, est surtout de tester ces réseaux dans différentes situations, afin de connaître leurs aptitudes et leurs limites. Ainsi, nous allons faire des comparaisons entre ces réseaux, à la lumière des résultats fournis par chacun d'eux.

III.1 Reconnaissance de Formes et Traitement d'Images

La reconnaissance est l'une des applications les plus importantes en traitement d'images à lesquelles aboutit, généralement, la segmentation. En effet la segmentation sert à subdiviser une image en plusieurs régions où les différentes composantes ont des caractéristiques en commun. Avant cette opération l'image brute doit généralement subir des opérations dites de prétraitement. Ces dernières ont comme objectif d'améliorer l'image et de la préparer pour un traitement ultérieur. Selon l'état de l'image et l'objectif à atteindre, nous citons parmi ces opérations, la réduction de bruit, l'amélioration de contraste, la détection des contours et la binarisation. C'est après ces processus de prétraitement que la segmentation est effectuée. La reconnaissance de formes survient par la suite et a comme objectif de reconnaître les différentes parties de l' image (segments, formes...), les restituer et les isoler du reste de cette image[Pra78].

En traitement d'images Il existe deux approches principales à l'aide de lesquelles la reconnaissance de formes est généralement effectuée [Fu.87]

La première approche, dite *de décisions théoriques,* est basée sur le calcul de la fonction *discriminant.* Celle ci consiste à représenter chaque forme par un vecteur $X = (x_1, x_2, \ldots, x_n)^T$ contenant ses caractéristiques (Périmètre, longueur, intensité moyenne,...). Ayant M classes existantes, des fonctions, appelées *discriminant* ou *fonctions de décision,* $d_1(X), d_2(X), \ldots, d_M(X)$ sont définies pour chacune de ces différentes classes. Ainsi un vecteur X^* est identifié appartenant à la classe j si la valeur de la *jéme* composante de la fonction *discriminant* appliquée pour ce vecteur est la plus importante que toutes les autres c.à.d:

$$d_j(X^*) > d_k(X^*) \qquad k = 1,\ldots M;\ k \neq j$$

Cette fonction de décision peut, par exemple, être représentée par les intercorrélarions de la forme à reconnaître avec le vecteur moyenne de chaque classe. La distance euclidienne entre le vecteur X^* et la moyenne des vecteurs de chaque classe peut aussi constituer une fonction de décision. Dans ce cas c'est la classe à la distance minimale qui est sélectionnée.

La seconde approche est basée sur les méthodes dites *structurelles.* Dans cette méthode il s'agit d'exploiter les caractéristiques géométriques de chaque forme. Ainsi chaque exemple est divisé en composantes géométriques définies par des vecteurs de longueurs et de direction déterminées appelées *éléments Primitifs* (Fig.III.1). En commençant par le sommet gauche de chaque objet, une poursuite de son contour, suivant le sens des aiguilles d'une montre, est effectuée en utilisant ces vecteurs, suivant la forme de la région où l'on se trouve. Le contour est par la suite identifié par un code défini par les différents éléments primitifs (a,b,c,d) utilisés pendant la poursuite du contour (Fig.III.1). La longueur et la direction de chaque élément étant déjà définies, la forme est rapidement reconnue juste à l'identification de son code.

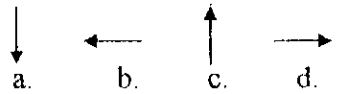


Fig.III.1 Exemple d'éléments primitifs et leurs codes de poursuite.

Cette seconde approche utilisant les *éléments primitifs* diffère par rapport à la première par le fait qu'elle dépend beaucoup des formes géométriques existantes dans l'image. Elle ne peut être directement utilisée sur n'importe quelle image sans examen préalable de ces formes géométriques.

III.2. Reconnaissance de caractères et de chiffres par réseaux de neurones

Les recherches dans le domaine de la reconnaissance de caractères ou de chiffres manuscrits se sont orientés depuis des années vers les possibilités de réalisations pouvant apporter des solutions satisfaisantes. Ces solutions doivent, en effet allier un taux de reconnaissance et une vitesse acceptable, ceci avec un investissement raisonnable en moyens de traitement autonomes.

Nous nous sommes intéressés à ce type de reconnaissance de formes pour deux raisons:
La première est qu'il s'agit d'un problème concret dont le contexte bien défini permet des comparaisons significatives entre différentes techniques utilisées. Ceci contrairement à d'autres qui peuvent être beaucoup plus spécifiques et ne permettent pas de constituer un bon terrain pour la comparaison des résultats fournis par différentes méthodes.
La seconde résulte du fait que la variation du style d'écriture du caractère, la qualité du manuscrit ainsi que la contrainte du temps de reconnaissance en font un domaine d'expérimentation assez adapté aux réseaux de neurones.

Cette opération de reconnaissance consiste à faire l'apprentissage d'un réseau avec des caractères de différentes formes en vue de les lui faire reconnaître en phase de rappel. Outre la reconnaissance de caractères écrits de maniérés différentes, le réseau doit aussi être insensible au bruit et surtout capable de restituer les formes qui ont été déformées ou tronquées. Ces applications trouvent toute leur importance dans la reconnaissance rapide d'écriture ou de codes (code postal, par exemple) et notamment pour la restitution de contenus des documents ayant subi des détériorations.

III.2.1 Reconnaissance de caractères par réseaux de neurones

1. Représentation des formes et réseaux utilisés

Les images que nous allons utiliser seront représentées par des vecteurs à n composants binaires (n étant le nombre de neurones à l'entrée du réseau à utiliser). Nous avons choisi d'utiliser des images à 256 éléments (de dimensions de 16x16 pixels, c'est à dire à la dimension d'un caractère normal écrit à la main). Cette représentation que nous avons choisie n'est pas la plus optimale. Il en existe encore d'autres mais qui nécessitent des prétraitements qui font intervenir d'autres algorithmes. Toutefois, pour résoudre en grande partie le problème de position du caractère, dans notre application, nous avons calé les images en haut et à gauche dans le cadre constitué par la couche d'entrée du réseau (Figure III.2)
La base d'apprentissage est constituée de 20 lettres manuscrites de l'alphabet latin qui sont générées en binaire. Ces exemples constituent en fait, les prototypes qui seront mémorisés sur la base desquels les entrées des réseaux seront classées.

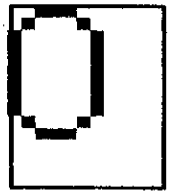


fig.III.2 Représentation du caractère à l'entrée du réseau

Nous utiliserons séparément trois types de réseaux de neurones:

Le réseau de Hopfield

Ce réseau est constitué par une couche dynamique entièrement connectée de 256 neurones répartis en plan représentant l'image. Le mode de fonctionnement que nous avons choisi est basé sur le modèle synchrone dont l'entraînement est fait suivant l'algorithme d'apprentissage non supervisé basé sur la règle de Hebb (cf. II.2.2.2)

Le réseau à mémoire optimisée OLAM

Ce réseau est constitué d'une couche de 256 neurones entièrement connecté identique à celle de Hopfield. Son apprentissage est assuré par la méthode de pseudo-inverse que nous avons présenté dans le chapitre II (cf.II.2.5).

Le réseau de Hamming

L'entrée du réseau est constituée d'un plan de 16x16 neurones, le *Lower-Subnet* est constitué de 20 neurones représentant les différentes classes à mémoriser et reliés au Maxnet Nous avons rajouté une couche qui est relié au *Maxnet* par des poids synaptiques permettant de déduire et délivrer le caractère en sortie, suivant les réponses du *Maxnet* (figure III.6).

2. Présentation des résultats

Après la mémorisation des 20 prototypes par chacun des trois réseaux, nous leur avons présenté 40 caractères inconnus afin d'observer leur comportement.
Nous tenons d'abord à signaler que pour le réseau de Hopfield qui est entraîné par la règle de Hebb, pour la majorité des tests effectués, les sorties ont donné des résultats erronés. Ainsi, on a été obligé de diminuer la base d'apprentissage de ce dernier à 14 exemples. Les résultats présentés pour ce réseaux sont donc pour une base d'apprentissage, qui est réduite par rapport à celle utilisée pour les deux autres réseaux.
Nous signalons, part ailleur, que les images présentées pour illustrer les résultats dans les figures (III.3), (III.4) et (III.5), ont subi un grossissement afin que les résultats paraissent plus clairement.

Pour bien faire apparaître les aptitudes et les handicaps de chacun de ces réseaux, nous organisons la présentation des tests à lesquelles nous avons soumis ces réseaux en trois parties.

Test 1

Le premier groupe d'entrées que nous présentons consiste à des exemples correspondant aux caractères mémorisés, mais dont les formes sont différentes par rapport aux prototypes appris. C'est la capacité de mémorisation et classification de ces réseaux qui est testée par ces exemples.
La figure (III.3) présente une sélection de quelques entrées avec les réponses de chacun des trois réseaux.

Pour ces entrées, nous remarquons que le réseau de Hamming et le OLAM ont présenté de bons résultats. Le réseau de Hopfield avec une base d'apprentissage réduite a parfois fourni des sorties quelque peu déformées qui ne sont dans la plupart des cas pas loin des caractères originaux.

Fig III.3 Réponse des réseaux pour des entrées de forme différente par rapport aux prototypes d'entraînement.

Nous avons remarqué dans ce premier test que le réseau de Hopfield atteignait rapidement l'équilibre. Ceci est dû aux fait que les entrées utilisées représentaient en général des états assez proches, pour chacun d'eux, d'un minimum correspondant à un prototype mémorisé ou à un état intermédiaire dans la surface de la fonction d'énergie. Ceci entraîne leur attraction rapide par les bassins d'attraction.

Test 2

Le second type de test consiste à injecter dans ces réseaux des entrées tronquées ou déformées. Ainsi, les réseaux doivent effectuer la complétion de ces images. Les exemples les plus significatifs et les sorties des réseaux sont présentés sur la figure (III.4).

Nous constatons, que le réseau de Hopfield a montré des résultats médiocres. Nous avons remarqué que plus la base d'apprentissage est importante moins les résultats de ce réseau sont satisfaisants. Ainsi à partir d'un certain seuil de la dimension de la base d'apprentissage, le réseau devient incapable de restituer les parties manquantes dans les exemples.

Le OLAM, pour sa part, a montré des résultats sensiblement meilleurs que ceux du réseau de Hopfield. Les quelques défauts remarqués sur les résultats de la figure (III.4), sont dus au nombre important d'exemples stockés par ce réseau.

Le réseau de Hamming a présenté les meilleurs résultats par rapport aux deux autres. Les cas difficiles pour ce réseaux sont résolus par de mauvaises classifications, mais jamais par des réponses déformées ou bruitées.

Test 3

Dans ce test, nous avons présenté des images bruitées et parfois déformées. Quelques exemples sont repris dans la figure (III.5). Dans ces exemples, le taux de bruit varie entre 15% (III.5.a) à 80% (III.5.f). Les réseaux doivent cette fois ci effectuer le filtrage.

Le réseau de Hopfield a fourni au début des résultats médiocres. Nous avons été amenés à faire plusieurs essais en jouant sur les valeurs des seuils de la fonction de sortie des neurones (équation II.4; II.5) jusqu'à l'obtention de réponses meilleures en moyenne. Nous avons constaté que ce réseau est sensible au bruit et se sature rapidement lorsque le nombre de prototypes stockés est important. Ceci confirme ce que nous avons développé dans le chapitre précèdent (cf.II.2.4). En effet plus le nombre de prototypes est important, plus le bruit présent dans les images en entrée devient influent.
Ceci se traduit sur l'équation (II.10) d'apprentissage par l'augmentation des corrélations, donnant ainsi plus d'ampleur au bruit.

Le OLAM a été moins sensible au bruit et a fourni dans les cas extrêmes (Fig.III.4.f) des réponses inhibant une grande partie du bruit.

Le réseau de Hamming s'est montré très efficace en filtrage. Mais cela n'a pu être obtenu qu'après l'élaboration d'une technique d'adaptation du taux d'inhibition ε afin d'activer la recherche de l'optimum (équation II.19). cette technique consiste à commencer avec un taux d'inhibition très petit. Celui ci est augmenté après un nombre d'itération fixé tans que plusieurs neurones sont toujours actifs. Au fur et à mesure que la compétition avance le taux est augmenté et les neurones s'inhibent progressivement jusqu'à l'apparition de la classe *"gagnante"*.

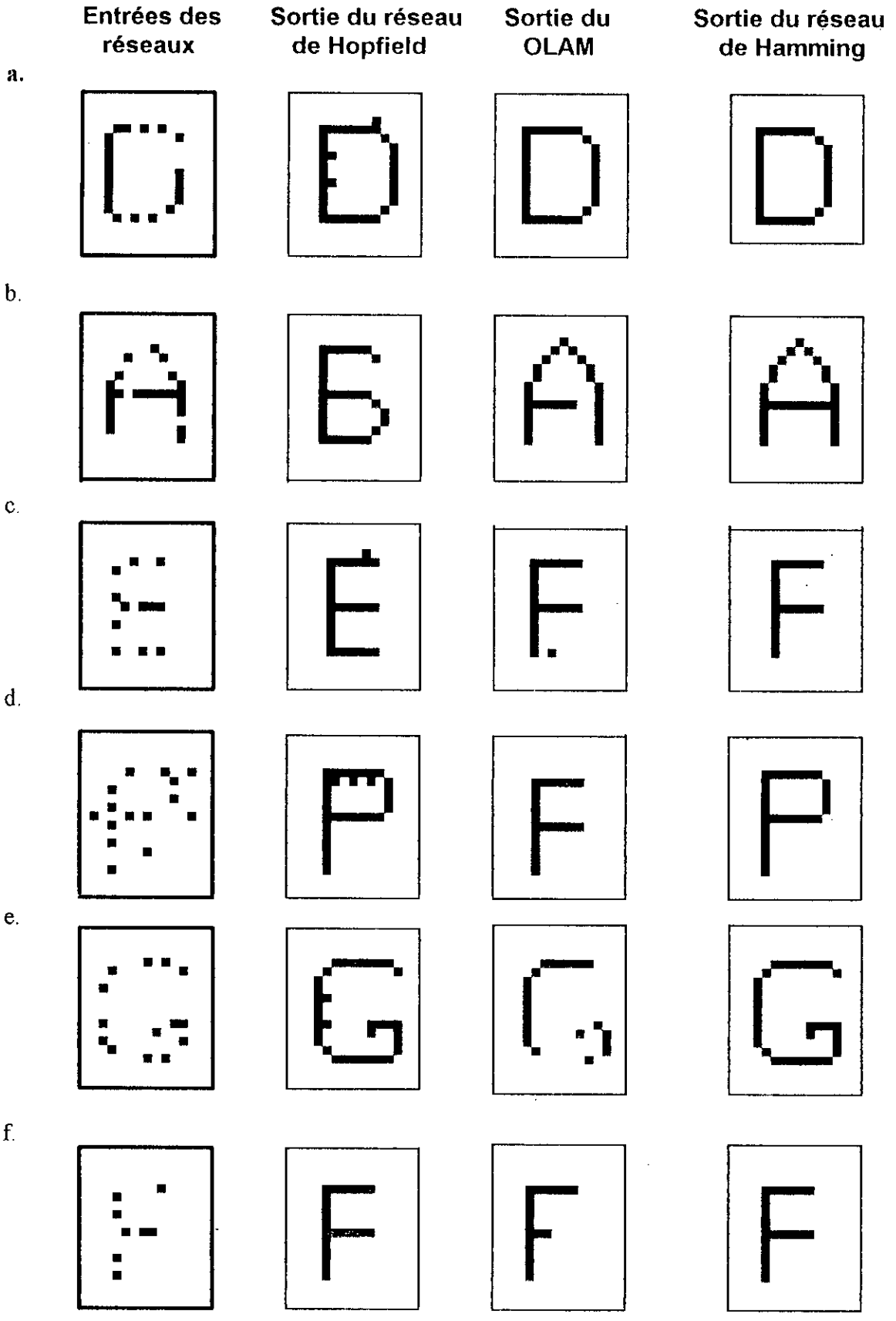


Fig III.4 Réponse des réseaux pour des entrées de formes incomplètes Opération de complétion.

3. Interprétations des résultats et comparaison

Au cours de cette application, nous avons testé les réseaux avec plusieurs cas, afin d'observer leurs aptitudes.

Le OLAM, a montré de bonne performance, en effet sur tous les caractères inconnus qui lui ont été présentés, nous avons obtenu les résultats suivants:

75% des caractères en entrées ont été correctement reconnus; c'est à dire que l'état stable correspond au prototype correct. Voir les figures (III.3), (III.4.a,g,f), et(III.5.a,b,c,d).

15% ont donné des états stables qui ne représentent aucun prototype. L'état stable que nous avons obtenu dans ce cas est une combinaison linéaire seuilée avec prédominance du prototype de la bonne classe. On remarque que c'est le cas pour les exemples (III.5.f), (III.4.c) qui n'aboutissent pas sur un prototype; ces formes sont *"inventées"* par le réseau. Ces états stables sont souvent nommés *Parasites.*

10% sont mal classés. L'état stable correspond à un autre caractère. C'est le cas entre autres des figures (III.4.d), (III.5.d).

Le réseau de Hamming s'est montré le plus performant. Sur toutes les entrées qui lui ont été soumises *95%* ont été correctement classées. Le reste a été mal classé mais jamais des états inconnus n'ont été crées. Ceci revient au mode de fonctionnement de ce réseau qui, à travers la méthode du Minimum Optimal, ne permet aucune interaction entre les états stockés. Dans le cas où l'entrée est très bruitée, ce réseau peut faire une mauvaise classification, qui correspond cependant au prototype le plus proche par rapport à cette entrée, que le réseau a pu détecter. Les figures (III.5.d) et (III.5.e) illustrent ce cas.

Le réseau de Hopfield doté de la règle de Hebb a été, dans la plupart des cas, incapable de classer les caractères qui lui ont été présentés. Il a fallu réduire sa base d'apprentissage et jouer par la suite sur les seuils des neurones. Tous les états ont convergé, au début, vers un seul grand "attracteur". Ce résultat n'est pas surprenant compte tenu des corrélations importantes qui existent entre les prototypes. En réduisant la base d'apprentissage nous avons réduit l'effet de ces corrélations (équation II.10). Nous avons tenu à présenter les résultats que ce réseau a donnés avec cette base d'apprentissage qui est réduite afin d'apprécier ses performances. Nous constatons que ces performances deviennent plus limitées lorsque les prototypes à mémoriser ne sont pas mutuellement orthogonaux. De plus, le réseau fourni parfois, des états qui n'existent pas parmi les prototypes mémorisés, comme dans le cas des figures (III.4.a.), (III.4.b.) et (III.3.d.); ce qui est dû au rapprochement entre bassins d'attractions qui favorise une influence mutuelle entre eux, et qui conduit à des états d'équilibre intermédiaires (cf.II.2.2.2).

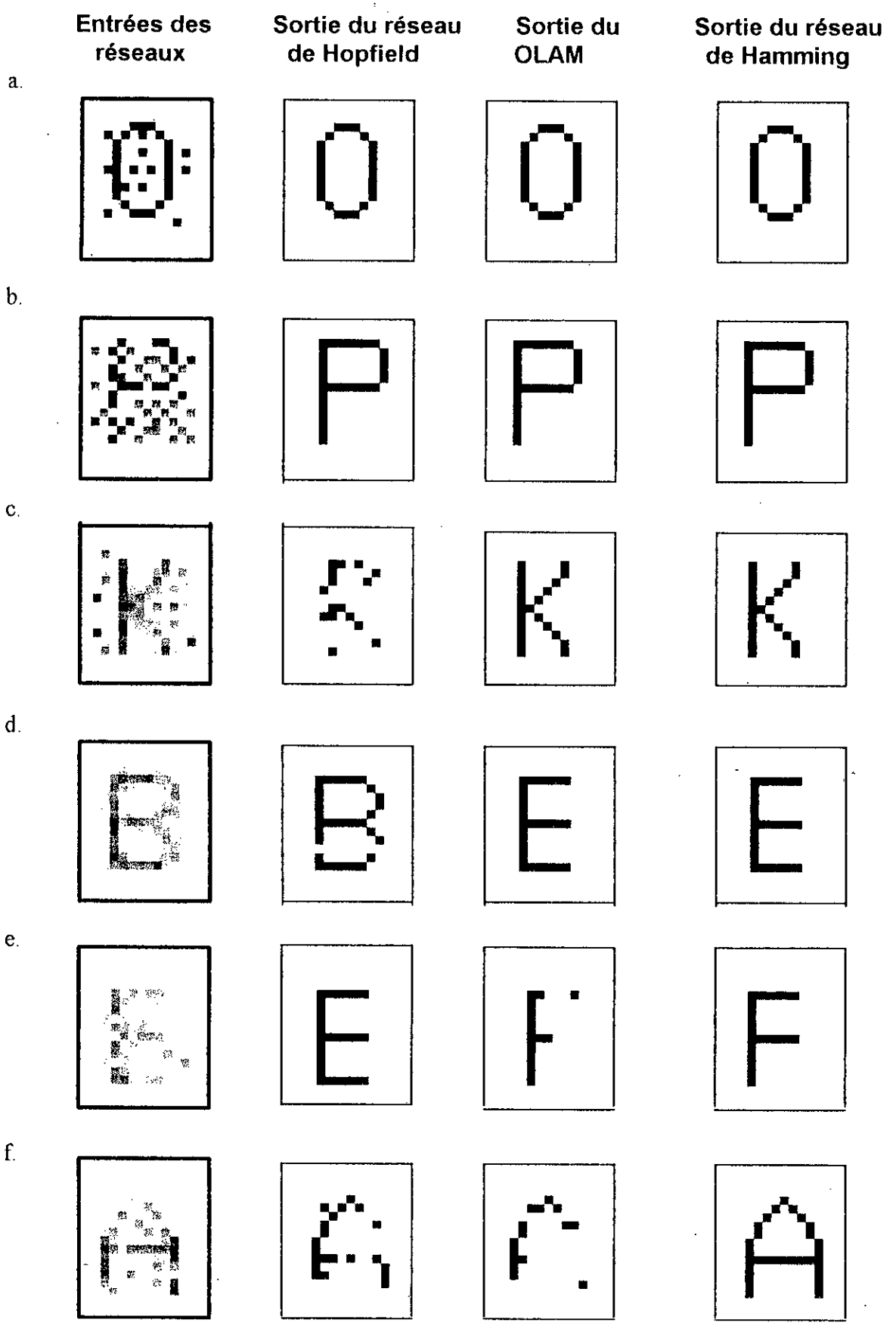


Fig III.5 Réponse des réseaux pour des entrées Bruitées et tronquées.
Filtrage des entrées.

III.2.2 Reconnaissance de chiffres arabes par le réseau de Hamming et par le BAM

Dans cette application nous nous proposons d'utiliser les réseux de neurones pour faire la reconnaissance des chiffres arabes.
Pour cela deux réseaux seront séparement utilisés le *réseau de Hamming* et le *BAM*.

Le réseau de Hamming ayant enregistré la meilleure performance dans l'application précédente constitue selon notre étude un très bon moyen pour les systèmes de reconnaissances nécessitant un filtrage et une correction de signaux.
Nous avons ainsi choisi ce réseau pour l'appliquer en reconnaissance de chiffres arabes, afin d'approfondir son étude.

Nous avons implementé la Mémoire associative Bidirectionnelle *BAM* pour cette application pour deux raisons. La première vise à prouver la possibilité de son utilisation pour les problèmes de mémoires autoassociative pour lesquelles le réseau de Hopfield est généralement choisi. La seconde, est que nous avons remarqué que ce réseau, peut constituer un bon moyen pour la compression de signaux. Ceci, à travers son architecture bidirectionnelle.

L'apprentissage des deux réseaux pour cette application est effectué à l'aide de dix prototypes générés, correspondant au dix différentes formes des chiffres arabes représentant les différentes classes à générer.

Le réseau de Hamming a la même architecture que celui utilisé dans l'application précédente. La figure (III.6) montre l'architecture complète de ce réseau.

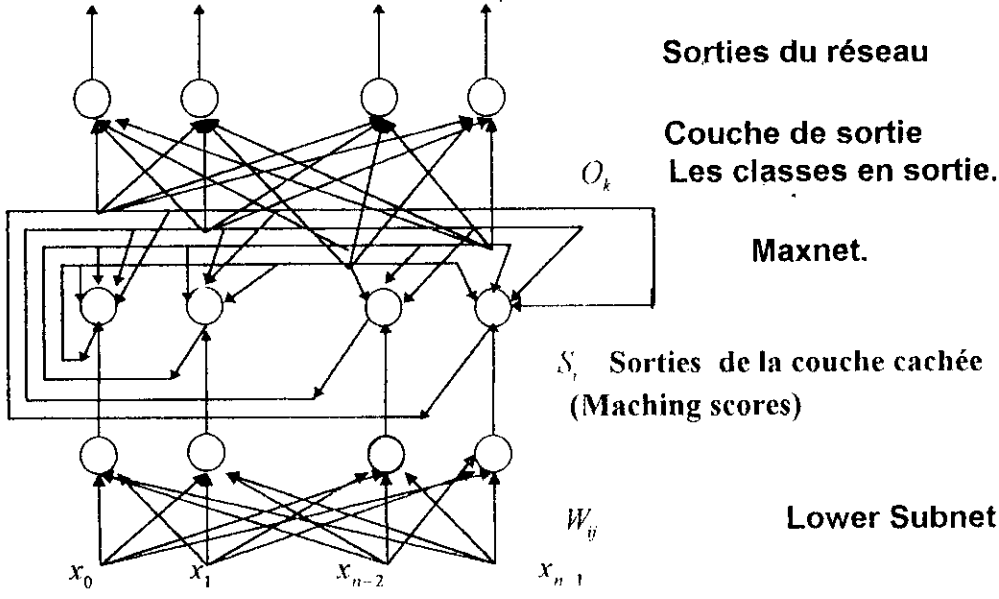
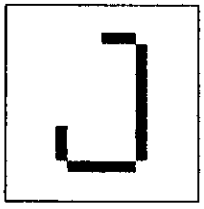
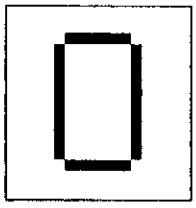
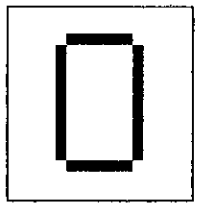


Fig.III.6 Architecture du réseau de Hamming utilisé

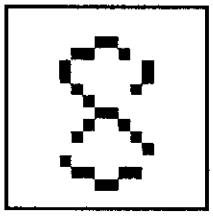
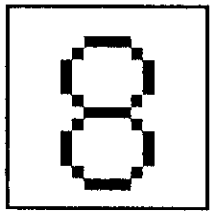
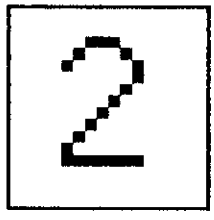
Le BAM est constitué d'une couche d'entrée 16X16 et d'une couche de sortie de 4 neurones. Ces derniers doivent représenter des combinaisons dont chacune d'elle correspond à une classe précise. Ainsi, dés que la forme est reconnue, le chiffre correspondant à la combinaison trouvée est affiché dans la couche d'entrée. Nous visons à travers cette structure à faire apparaître la capacité de ce réseau de faire, en plus de la reconnaissance, la compression des images reconnues. En effet, chaque forme reconnue est représentée en sortie par un vecteur de quatre composants, alors que celle ci est représentée à l'entrée par une image de 16X16 (vecteur de 256 composants). L'apprentissage de ce réseau est effectué à l'aide de la règle des corrélations présentée en (II.2.3).Après l'apprentissage nous avons testé le réseau sur plusieurs exemples. Nous présentons quelques résultats sur la figure (III.6).

Entrées des réseaux: **Sorties du réseau de Hamming** **Sorties du réseau B.A.M**

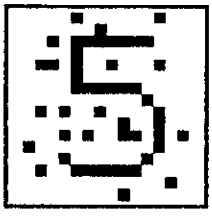
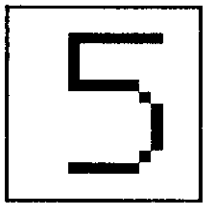
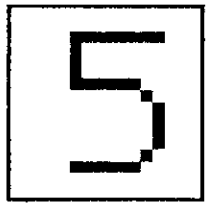
a.

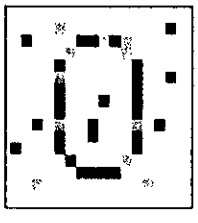
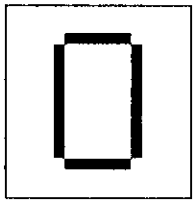
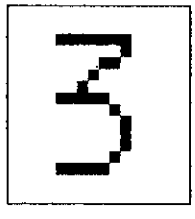
b.

c.

d.

e.

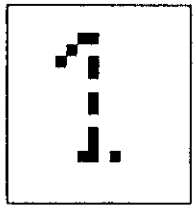
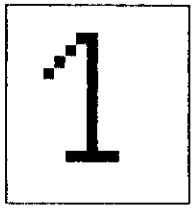
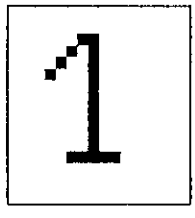




Fig.III.7 Réponse des réseaux pour différentes situations.
Ces images sont sous format TIF 16X16 avec agrandissement pour l'affichage.
Performances des réseaux.

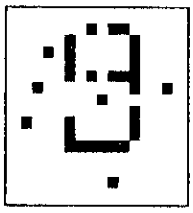
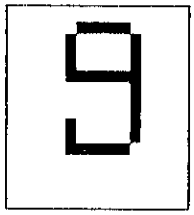
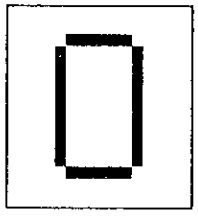
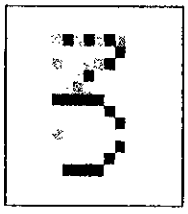
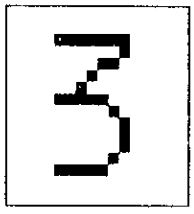
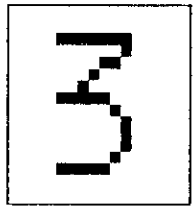
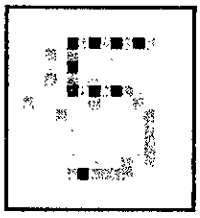
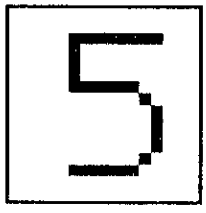
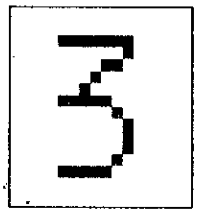
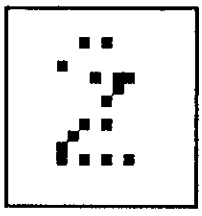
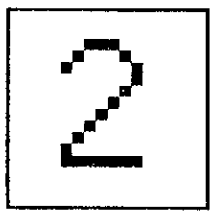
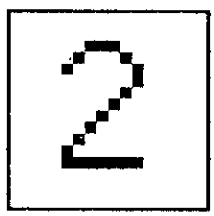
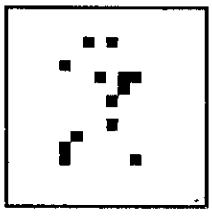
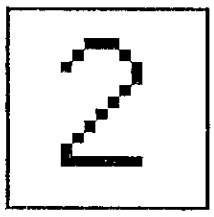
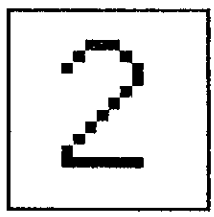
	Entrées des réseaux:	Sorties du réseau de Hamming	Sorties du réseau B.A.M
a.			
b.			
c.			
d.			
e.			

Fig.III.8 Réponse des réseaux pour différentes situations
Performances des réseaux

III.2.2.1. Interprétation des résultats et comparaison

Nous constatons que le BAM a fourni des réponses satisfaisantes pour un peu plus de la moitié (60%) des images que nous lui avons présentées. Néanmoins, il nous a fallu, comme c'était le cas pour le réseau de Hopfield, régler le seuil de la fonction de sortie non linéaire afin d'obtenir les bonnes réponses dans la plus part des cas.

Quant au réseau de Hamming, Nous remarquons, encore une fois, que celui ci est capable de filtrer le bruit et restituer l'image. L'avantage de ce réseau est qu'il fournit toujours l'image la plus proche de l'exemple en entrées. Sur tous les tests effectués le taux de succès de ce réseau est proche de 98%.

Ainsi, nous constatons, à la lecture des résultats obtenus, que le réseau de Hamming est un moyen très efficace de filtrage et de complétion de signaux.

Malgré son haut degré d'absorption des bruits, ce réseau n'est cependant pas à l'abri de mauvaises classifications. Dans le cas où l'entrée est très bruitée, ce réseau fourni la classe qui lui est la plus proche. Ainsi celui-ci a prouvé sa robustesse vis à vis au bruit.

L' inconvénient le plus important que nous avons relevé au cours de notre utilisation de ce réseau, est la durée de recherche du minimum qui peut augmenter parfois et où des oscillations peuvent être générées.
En effet, nous avons constaté que cette recherche qui est basée sur la technique d'inhibition latéral, est tributaire d'un bon choix du taux d'inhibition dans l'équation qui régie l'évolution de cette couche dynamique (équation II.19). Ce paramètre dépend du nombre de classes mémorisées. Un taux d'inhibition très important est très sélectif et risque de faire durer la recherche ou, tout simplement, disqualifier toutes les classes. Un taux très petit peut rendre la compétition très ouverte.
Pour pallier cela, nous avons adopté une technique qui consiste à adapter un taux d'inhibition dynamique. Avec un taux croissant pendant la recherche, le neurone gagnant est rapidement repéré par le Maxnet et l'image initiale est envoyée vers la couche de sortie à travers les poids synaptiques.

Pour le BAM, nous constatons qu'il est possible de l'utiliser pour des problèmes de mémoires auto-associatives. Ce réseau reste, cependant, sensible au bruit, comme c'était le cas pour le réseau de Hopfield dans l'application précédente (Fig.III.8.c). Néanmoins, le BAM s'est montré assez efficace dans certains cas ou l'entrée était bruitée, au cours de la présente application (Fig. III.8.b).

Conclusion

Dans ce chapitre, nous avons exploré quatre des plus importants réseaux à apprentissage non-supérvisé. Nous pouvons tirer d'importantes conclusions qui seront utiles pour des applications ultérieures.

Le réseau de Hopfield a comme principal défaut, sa faible capacité de mémorisation. Le mode de fonctionnement de ce réseau reste, de plus, sensible aux bruits et inefficace lorsque les prototypes à mémoriser ne sont pas mutuellement orthogonaux. Mais nous avons constaté qu'en réduisant, le nombre de prototypes, la non orthogonalité des exemples et la présence de bruit peuvent ne pas avoir d'influence sensible sur les résultats. Le taux de sucés a même avoisiné les 70% sur les cas de déformation ou de présence de bruit sous condition d'adaptation du seuil de la fonction de sortie des neurones. La dimension des états stockés influe grandement sur ce réseau. Ainsi pour une base d'apprentissage modeste qui est très inférieure au seuil que, nous avons présenté dans l'équation (II.7), l'utilisation de ce réseaux fournirait normalement des résultats satisfaisant.

Pour les BAM, bien que leur apprentissage soit basé sur les mêmes règles que le réseau de Hopfield, à savoir le principe de corrélation, il est très important de noter que ces réseaux sont une bonne alternative au modèle de Hopfield. En effet ceux ci peuvent être utilisés dans les problèmes de reconnaissance hétéro-associative ou auto-associative avec un nombre réduit de connexions.
Ainsi, selon notre travail, il est intéressant d'utiliser le réseau BAM en organisant ces sorties, de façon à former une combinaison pour chaque classe, dans les problèmes de mémorisation Autoassociative. Ceci permet notamment d'économiser le nombre de poids synaptiques nécessaires pour cette opération.

Le OLAM constitue selon notre étude et à l'examen des résultats fournis, l'une des mémoires associatives les plus efficaces. Il a d'une part, montré une capacité de stockage supérieure à celle de Hopfield et d'autre part une robustesse appréciable au bruit. Cependant, ce réseau nécessite des calculs importants pour l'apprentissage qui doit passer par le calcul de la Pseudo-Inverse de matrice, le plus souvent en utilisant le procédé de Moor Penrose.

Enfin, Nous enregistrons que le réseau de Hamming donne le meilleur rapport calcul-efficacité. En effet en plus de ses aptitudes qui sont meilleures tant en capacité de stockage qu'en filtrage, celui ci est aussi plus rentable du point de vue implémentation. Le nombre de poids nécessaires à calculer pour ce réseau est, en fait, nettement inférieur à celui nécessaire pour les autres réseaux. Pour notre application la dimension de la matrice poids de ce réseaux a été pratiquement neuf fois inférieures à celles du réseau OLAM.
Cependant, dans la seconde application, qui nous a permis d'approfondir l'étude de ce réseau, nous avons constaté que la recherche de l'optimum peut durer longtemps. Cette recherche en temps réel qu'effectue ce réseau pour tirer la classe gagnante est, en fait, "le prix à payer" pour obtenir ces résultats. Dans certains cas, cette recherche peut se poursuivre indéfiniment sans jamais atteindre l'optimum, si le taux d'inhibition n'est pas convenablement choisi. Un taux très important met tous les neurones hors de la sélection et provoque donc la divergence de la couche dynamique qu'est le Maxnet. Un taux moins important, par contre, peut conduire vers une convergence plus rapide ou au contraire rendre la recherche moins sélective. L'échelle de grandeur de ce paramètre est relative suivant le nombre de classes. Ainsi, un taux important pour un réseau ne l'est pas nécessairement, pour un autre. L' utilisation d'un taux d'inhibition

dynamique croissant, que nous avons présenté dans ce chapitre, nous a permis de résoudre ce problème, en réduisant considérablement le temps de recherche.

Ainsi, selon notre étude, et à la lumière des résultats que nous avons présentés, nous pouvons déduire que pour les problèmes de reconnaissance nécessitant filtrage et correction de signaux, le réseau de Hamming constitue la meilleure alternative.

CHAPITRE IV:

RESEAUX DE NEURONES A APPRENTISSAGE SUPERVISE

Chapitre IV
RESEAUX DE NEURONES A APPRENTISSAGE SUPERVISE

Introduction:

Les réseaux à apprentissage supervisé représentent la classe de réseaux de neurones qui a fait l'objet du plus grand nombre de travaux de recherche, et qui a connu une grande évolution depuis le milieu des années 1980.
Des noms de réseaux comme *le "Perceptron"*, *l'"ADALINE MADALINE"* ou d'une manière générale *les "Réseaux Multicouches"*, constituent maintenant des "classiques " dans le domaine de réseaux de neurones.

En bref, l'histoire de ce type de réseaux commence par le Perceptron de Rosenblat en 1957 [Dav 90]. Puis ce fut Widrow et Hoff avec l'ADALINE en 1960. Après une époque très difficile, c'est Rumelhart, Hinton et Williams qui ont ouvert une nouvelle ère avec l'algorithme d'apprentissage *Backpropagation* et les *Réseaux Multicouches* en 1986 [Rum 86].
C'est à partir de cette date que les réseaux à apprentissage supervisé ont connu un grand essor, qui dure à nos jours. C'est ainsi, que ces réseaux ont été particulièrement utilisés en automatique, où la modélisation de systèmes mal connus et l'élaboration de lois de commande adaptative pour ces systèmes, ont été aux centres des applications [Bav 87, Eil 88].

Dans ce chapitre, nous introduirons d'abord les fondements de cette technique d'apprentissage. Par la suite nous étudierons ces réseaux de neurones, avec un développement sur leurs algorithmes d'apprentissage. Nous commencerons avec les réseaux statiques sous leurs formes LBF et RBF. Puis nous nous introduirons dans les réseaux dynamiques avec les différentes architectures existantes ainsi que leurs techniques d'entraînement.
Une conclusion clôturera notre étude sur ces réseaux qui constituent actuellement la technique connexioniste la plus importante.

IV.1. Apprentissage supervisé

L'apprentissage supervisé a comme objectif l'estimation d'une relation: $X \rightarrow Y$
à partir d'observations, sur des échantillons aléatoires (X_i, Y_i) $i=1,2,...,n.$, faites dans l'espace des entrées-sorties.

Ce type d'apprentissage nécessite la présence d'un superviseur (Teacher) qui présente au réseau ces entrées et leurs sorties désirées.
Il a pour rôle de ramener le réseau vers le comportement désiré imposé par le superviseur; ce qui se fait par la recherche du vecteur synaptique localement optimal W^*, parmi toutes les combinaisons possibles dans l'espace des poids, qui assurerait la meilleure approximation possible [Bau88, Kos90].

Ayant un même objectif, deux approches peuvent être faites pour ce type d'apprentissage. La première est basée seulement sur la validité de la décision du réseau représentée par sa sortie. Ce type de réseaux est alors appelé (DBNN) *Decision Based Neural Networks* ou Réseaux basés sur les décisions [Lip87]. La deuxième catégorie est, par contre, basée sur l'optimisation d'un critère parfois appelé fonction coût, qu'il faut optimiser. La différence capitale existant entre ces deux entraînements, réside dans l'information que le superviseur doit fournir et son utilisation. Dans la première approche, qui est basée sur la

décision prise par le réseau (éventuellement la classification de chaque exemple), le superviseur doit juger si cette décision est correcte ou pas. Dans la seconde, qui est basée sur l'approximation ou l'optimisation, des valeurs exactes de la sortie désirée doivent être mises à la disposition du réseau [Kun93].

Il est nécessaire de déterminer, dans ce genre d'apprentissage, la fonction qui mesure l'écart entre les sorties désirées et celles fournies par le réseau. Cette fonction appelée *Fonction Objectif*, constitue le critère à minimiser en agissant sur les poids synaptiques.
Ce critère est en général une fonction de l'erreur entre la sortie désirée et la réponse du réseau:

$$E = \Psi\left(Y_k - \hat{Y}_k\right) \tag{IV.1}$$

où Y_k est la sortie désirée et $\hat{Y}_k$ représente l'estimation du réseau.

Ainsi l'apprentissage devient une procédure qui cherche à minimiser ce critère à travers la recherche de la séquence de poids synaptiques favorable.

IV.2 Réseaux de Neurones Basés sur la Décision DBNN

Le Perceptron

Introduit pour la première fois par Rosenblat en 1958 [Rum86], c'est le plus ancien des réseaux. Il est basé sur le modèle de Mc Culloch et Pitts. Sa nouveauté par rapport à cet ancien modèle est qu'il est capable de modifier ses poids en fonction des exemples que l'on veut lui faire apprendre.
Le terme Perceptron désigne aujourd'hui les réseaux de neurones statiques constitués d'une couche de neurones appelée, couche d'entrée reliée à un ou plusieurs neurones, appelés neurones de sortie (fig.IV.1). L'information à l'intérieur de ces réseaux, ne circule que dans un seul sens; de l'entrée vers la sortie.

A travers le Perceptron, c'est le principe de fonctionnement de toute cette famille de réseaux de neurones basés sur les décisions qui est étudiée.

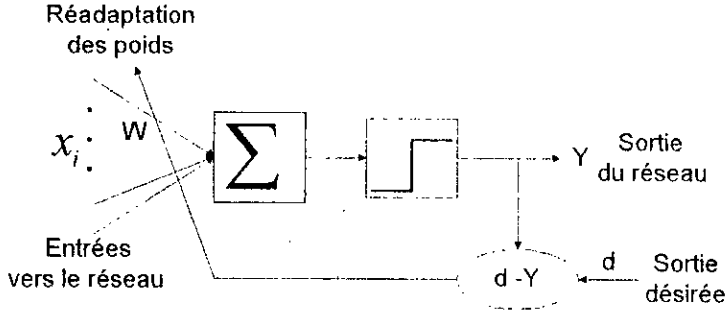


Fig.IV.1 Structure du Perceptron linéaire et son entraînement.

Un réseau basé sur les décisions effectue la séparation entre deux zones de l'espace. La position de l'hyperplan séparant ces deux régions constitue les contours de décision (fig.IV.2). L'équation définissant cet hyperplan est la Fonction appelée *Discriminant* $\Phi(x,y)$ qui, en

fonction des poids synaptiques, positionne cette séparation de sorte que le réseau puisse classifier correctement chaque entrée de part ou d'autre par rapport à cet hyperplan [Sim90].
Dans le cas du Perceptron linéaire, l'activité du neurone (figure IV.2) est définie par l'équation linéaire suivante:

$$net_i = \sum_{j=1}^{n} x_j \, w_{ij} + \theta_i \qquad i=1,...m \qquad \text{(IV.2)}$$

où les x_j représentent les entrées du neurone, w_{ij} le poid synaptique en provenance du j éme neurone de la couche d'entrée vers le i éme neurone de sortie et θ un paramètre qui agit sur le seuil en sortie.

La sortie est définiee par la fonction seuil:

$$s_i(net_i) = \begin{cases} 1 & net_i > 0 \\ 0 & net_i \leq 0 \end{cases} \qquad \text{(IV.3)}$$

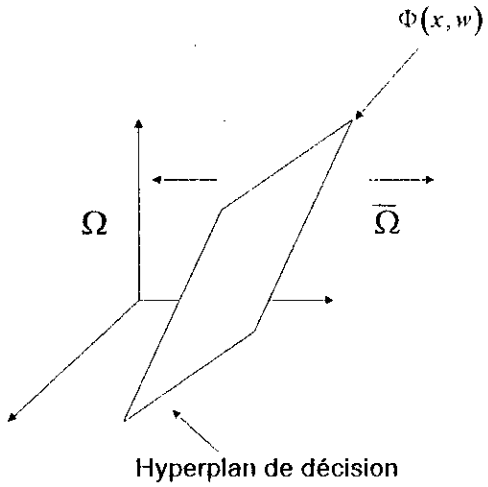


Fig.IV.2 Le positionnement de l'hyperplan de séparation par la réadaptation des poids.

L'adaptation des poids de ce réseau n'est autre qu'un cas particulier de l'application de la méthode de descente du gradient que nous étudierons plus loin.

$$W(t+1) = W(t) + \Delta W(t) \qquad \text{(IV.4)}$$

Dans le cas du Perceptron linéaire, on obtient:

$$\Delta W(t) = \eta \, e(t) X(t) \qquad \text{(IV.5)}$$

$$= \eta (d(t) - Y(t)) X(t) \qquad \text{(IV.6)}$$

Où η est un paramétre qui contrôle le taux d'apprentissage.
$X(t)$ représente le vecteur présenté à l'entrée du réseau, $Y(t)$ la sortie du réseau, $d(t)$ la sortie que nous désirons obtenir et $e(t)$ l'erreur en sortie à l'instant t.

Ainsi, en cas d'erreur deux possibilités peuvent se présenter:

•Si le vecteur d'entrée n'a pas été classé dans la classe Ω où il devait normalement y être, les poids sont renforcés avec une proportion de cet exemple, afin que l'hyperplan soit positionné de sorte à l'admettre à l'intérieur de la classe Ω.

$$W(t+1) = W(t) + \eta X(t) \qquad \text{(IV.7.a)}$$

•Par contre, si le vecteur en entrée a été classé dans la classe Ω, alors qu'il fait réellement partie d'une autre classe, Les poids doivent subir un *Anti-renforcement* qui réajuste la région de décision, pour remettre cet exemple à l'extérieur.

$$W(t+1) = W(t) - \eta X(t) \qquad \text{(IV.7.b)}$$

L'entraînement du réseau se poursuit jusqu'à ce que la position de l'hyperplan de séparation (figure IV.2) permette à chaque exemple d'être correctement classé.

Le principal inconvénient du Perceptron est qu'il n'est efficace que pour les problèmes linéairement séparables. Dans le cas contraire, L'algorithme d'apprentissage ne convergera jamais. Une solution rapide, mais non précise, consiste à minimiser un critère permettant une position optimal de l'hyperplan de séparation. C'est le cas de l'ADALINE (Adaptive Linear Neuron) où la Méthode des Moindres Carrés (LMS) est utilisée pour cela[Wid90].
Dans d'autres travaux, on a pu former des régions de décision complexes à partir de l'intersection de plusieurs régions linéaires de formes simples par une combinaison de plusieurs Perceptrons linéaires [Lip87].
Par ailleurs, Afin de former des espaces de classification non linéaire, il suffit de changer tout simplement la forme de la Fonction *Discriminant* en une fonction non-linéaire. Ainsi, des fonctions de Base Radiale (**RBF**) ou Elliptique (**EBF**) peuvent bien être utilisées. Kung a proposé un réseau où toutes ces fonctions sont utilisées en même temps. chacune étant utilisée pour un petit réseaux (*Subnet*). Ces petits réseaux sont tous reliés à une couche de neurones supérieure (*Maxnet*) qui désigne le réseau fournissant la meilleure classification utilisant ainsi la fonction de base adéquate pour le problème. C'est ce qui est appelé les **HDBNN** (*Hierarchical Decision Based Neural Networks*) [Kun 93].

IV.3 ADALINE, MADALINE

Ce type de réseaux trouve son application en tant qu'approximateur linéaire. Le premier modèle du genre, *ADALINE(ADAptive LINear Element)* est issu des recherches de Widrow et M. Hoff en 1960 sur les filtres adaptatifs [Dav90]. L'architecture de ce réseau est la même que celle du Perceptron, à savoir une couche d'entrée connectée à une ou plusieurs sorties. Dans le cas de plusieurs sorties, il est alors appelé *MADALINE (Multiple ADAptive LINear Elements).*
La différence de ce réseau par rapport au Perceptron classique, est que destiné à l'approximation de fonctions, sa sortie est linéaire (equ.IV.8).

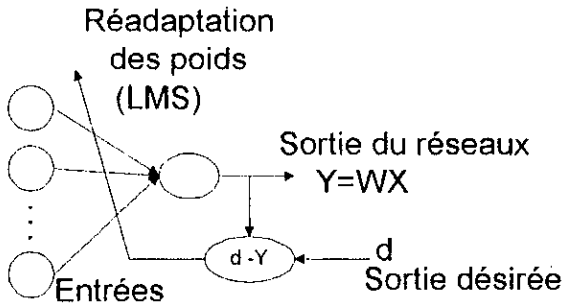


Fig.IV.3. Structure d'un Réseau ADALINE et son principe d'apprentissage.

IV.3.1 Apprentissage

A la présentation d'une entrée X^k, chaque sortie y_i^k est le produit scalaire de cette entrée avec un vecteur poids W_i (Fig IV.3).

$$y_i = \sum_{j=1}^{n} x_j w_{ij} + \theta_i \qquad i=1,...m. \tag{IV.8}$$

Ainsi pour chaque sortie y_j, on définit l'erreur instantanée à la sortie:

$$e_j^k = d_j^k - y_j^k \tag{IV.10}$$

Tel que d_j^k, représente la réponse désirée du j éme élément pour la *k éme* entrée. Le réseau est entraîné avec plusieurs entrées. Le problème est donc d'ajuster les poids W afin que, en moyenne, l'écart entre la sortie du réseau et celle désirée soit nulle pour toutes ces entrées.
La première méthode, qui peut être utilisée, est la méthode des moindres carrés LMS.

IV.3.1.1 Méthode des moindres carrés (LMS)

La LMS (*Least Mean Square error*) repose sur la minimisation d'un terme quadratique. L'erreur à minimiser est donc définie en fonction des poids synaptique:

$$\left(e^k\right)^2 = \left(d^k - W^T X^k\right)^2 \tag{IV.11}$$

Le réseau doit être entraîné sur plusieurs exemples, l'erreur doit donc être minimisée en moyenne sur tous ces exemples. Mais pour cela, il faut que la solution du minimum de la moyenne existe [Dem91].
Afin de minimiser la moyenne de l'erreur sur tous les exemples d'entraînement, on doit donc résoudre l'équation:

$$\frac{\partial}{\partial W} E\left[\left(e^k\right)^2\right] = 0 \tag{IV.12}$$

Or:

$$E\left[\left(e^k\right)^2\right] = E\left[d^k\right]^2 - 2E\left[d^k X^k\right] w^T + WE\left[X^{(k)^T} X^k\right] W^T \tag{IV.13}$$

En posant R la matrice moyenne des autocorrelations entre les exemples d'entrées et P celle des intercorrelations entre les entrées et leurs sorties désirées:

$$\begin{aligned} R &= E\left[X^{(k)^T} X^k\right] \\ P &= E\left[d^k X^k\right] \end{aligned} \tag{IV.14}$$

on obtient :

$$\frac{\partial}{\partial W} E\left[\left(Y^k\right)^2\right] = -2P + 2WR \tag{IV.15}$$

D'où l'on tire le vecteur poids optimal:

$$W^* = PR^{-1} \tag{IV.16}$$

D'où l'existence de cette solution.

Quand la matrice R est définie positive, le calcul du Hessian de l'expression (IV.11), permet de conclure que la solution donnée par l'équation (IV.16) est un minimum. De plus, si la matrice R n'est pas inversible, il est possible d'utiliser sa Pseudo-inverse [Sim90].

Il est intéressant de noter que le résultat obtenu dans l'équation (IV.16) constitue la même forme que l'équation (II.12) que nous avons obtenue pour l'apprentissage des réseaux OLAM, dont l'apprentissage est non supervisé(cf.II.2.5).
La linéarité du système garantit que l'erreur quadratique, définie par l'expression (IV.8), donne une surface moyenne qui est convexe dans l'espace, d'où l'existence de la solution.
Cette méthode est basée sur l'estimation de la moyenne de toutes les entrées pour atteindre la solution optimale satisfaisant tous les exemples présentés au réseau.
B.Widrow a proposé une solution qui évite l'estimation de l'espérance. pour cela l'algorithme agit à la présentation de chaque exemple en entrée. Ainsi il estime la trajectoire à prendre vers le point minimum à chaque présentation d'un vecteur d'entrée; c'est la méthode de Descente de Gradient[Bla91].

IV.3.1.2 Méthode de Descente de Gradient

Cette méthode est utilisée afin de déterminer les adaptations qu'il faut apporter aux vecteurs des poids synaptiques à chaque présentation d'une entrée, afin de faire décroître l'erreur de sortie jusqu'à obtention de la solution optimale.
L'idée consiste à évaluer la valeur du gradient de la Fonction Objectif (équation IV.11), pour quelques points W donnés dans la surface d'erreur. Cette mesure nous informe sur l'augmentation du terme d'erreur avec les poids correspondants. Pour faire diminuer l'erreur, il suffit donc d'aller dans la direction opposée du gradient.
B.Widrow a proposé un algorithme [Wid90] qui estime la trajectoire à prendre vers le point minimum à chaque présentation d'entrée. Ainsi, à chaque vecteur présenté, on engendre une surface d'erreur, dont on estime le gradient en fonction des poids synaptiques, et on modifie ces poids pour faire décroître l'erreur: c'est la méthode du Delta. De cette manière, le Gradient ne pointera que le minimum instantané à chaque étape. La convergence vers le minimum moyen théorique défini dans l'équation (IV.16) est assurée, mais cette manière de procéder fait que la trajectoire devient plus "chaotique".

En utilisant l'équation (IV.11), l'erreur quadratique instantanée à chaque étape devient:

$$\zeta^k = \frac{1}{2}\left(e^k\right)^2 = \frac{1}{2}\left(d^k - y^k\right)^2 \qquad \text{(IV.17)}$$

Cette méthode, dite aussi *de Gradient Stochastique*, donnera une variation des poids après chaque calcul de sortie [Bla91].
Afin de progresser dans le sens opposé du gradient, on a donc:

$$\Delta W^k = -\eta_k \frac{\partial \zeta^k}{\partial W^k} = e^k X^k \qquad \text{(IV.18)}$$

A partir des équations (IV.17) et (IV.18), on obtient:

$$w_{ij}^{k+1} = w_{ij}^k + \eta_k\left(d_i^k - y_i^k\right)x_j^k \qquad \text{(IV.19)}$$

où η_k est un paramètre appelé taux d'apprentissage, dont le rôle est de réguler la vitesse de convergence et contrôler la stabilité de ce processus.
Nous discuterons le rôle et l'influence de ce facteur, ainsi que les valeurs qu'il faut lui attribuer dans l'étude des réseaux multicouches (cf.IV.4).

Ce réseau de neurones, qui est un approximateur linéaire est, évidemment, limité aux fonctions linéaires. En outre, une augmentation du nombre de couches de ce réseau, n'a absolument aucune utilité. En effet, le modèle étant linéaire, n'importe quel nombre de couches sera équivalent à une seule couche dont la matrice des poids est le produit des matrices poids de toutes ces couches.
L'amélioration des performances de ce réseau nécessite donc, l'introduction de non-linéarités; ceci le rendra, entre autres, sensible à l'augmentation du nombre de couches.

IV.4 Réseaux multicouches statiques

Nous avons vu que l'introduction de non-linéarités est nécessaire afin d'améliorer les aptitudes des approximateurs. De plus, contrairement au cas linéaire, il est intéressant pour ces réseaux d'élargir leurs structures vers des architectures multicouches permettant d'obtenir des réseaux plus puissants aptes à traiter des problèmes plus complexes.
C'est en 1969 que Minsky et Papert [Koh77] ont démontré la nécessité de l'introduction de fonctions non linéaires et surtout de multiplier le nombre de couches pour résoudre les problèmes de non-linéarité, devant lesquels toutes les tentatives avaient échoué. Mais Le problème qui se posait, encore est l'entraînement de ces structures de neurones. C'est pourquoi ces réseaux ont été pratiquement délaissés, pendant presque une décennie. C'est en 1986 que les réseaux de neurones multicouches ont enfin pu être utilisés grâce à l'algorithme d'apprentissage *Backprpagation* que nous étudierons dans ce chapitre.

VI.1.1 Architecture des réseaux

L'organisation de ce type de réseaux fait que toute cellule d'entrée soit théoriquement suivie par une succession de couches dites *couches cachées*, pour finalement aboutir à la sortie (figure VI.4). Si un réseau est *"simplement étagé"*, chaque unité d'une certaine couche reçoit son entrée à partir de la couche précédente seulement. Au sens large rien n'empêche cependant, des interactions entre couches éloignées.
La structure en *Feedforword* veut que le signal traverse, à partir de la couche d'entrée, les couches structurées, pour arriver à la sortie, dans le sens direct, et jamais dans l'autre. Ce passage par ces couches contribue à la richesse du traitement de l'information à l'intérieur du réseau afin d'obtenir le résultat désiré en sortie [Fre92].

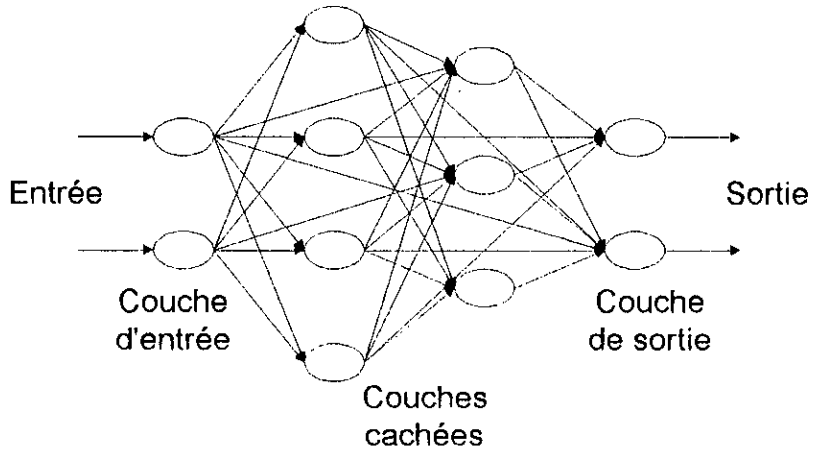


Fig.IV.4 Architecture d'un réseau de neurone statique multicouche à deux couches cachées.

IV.4.2 Réseaux à Fonction de Base Linéaire LBF

Ce type de réseaux est parfois appelé *Backpropagation* [Kos92], du fait que c'est cet algorithme d'apprentissage qui lui a permis de voir le jour.
Un réseau LBF a son discriminant qui est linéaire. Il est régi par l'équation dynamique suivante:

$$u_i^l(t) = \sum_{j=1}^{N_{l-1}} w_{ij}^l(t) y_j^{l-1}(t) + \theta_i^l(t)$$
$$y_i^l(t) = f\left(u_i^l(t)\right) \qquad \text{(IV.20)}$$

où:
$u_i^l(t)$ représente l'activité du i éme neurone dans la l éme couche.
$y_i^l(t)$ la sortie du i éme neurone dans la l éme couche.
L le nombre de couches.
N_l le nombre de neurones dans la l éme couche.
w_{ij}^l représente le poids synaptique du i éme neurone de la l éme couche, en provenance du j éme neurone de la couche précédente.
La fonction d'activation $f(.)$ du neurone est une fonction monotone non décroissante.
Cette fonction, que nous avons déjà présentée dans le premier chapitre (cf.I.2.3), a pour rôle de limiter l'activité du neurone tout en gardant sa continuité. Elle peut être, une fonction *Sigmoïde*, une *Tangente-Hyperbolique*, ou une *Log-Sigmoïde*. En sortie du réseau la fonction linéaire est la plus souvent utilisée. Ceci afin de rendre cette sortie libre de pouvoir prendre toutes les valeurs possibles désirées [Pat 96]. Pour les neurones de la couche d'entrées, cette fonction n'est autre que la fonction identité. Ainsi dans la couche d'entrée, chaque neurone fait passer une composante du vecteure entrées vers sa sortie sans aucun traitement c.à.d $y_i^1(t) = x_i(t)$; où $x_i(t)$ représente la *i éme* entrée du réseau à l'instant t.

Nous remarquons dans l'équation (IV.20) la présence du paramètre θ_i^l appelé *Bias*. l'introduction de ce paramètre, qui est différent pour chaque neurone, a pour objectif de donner plus de souplesse à la fonction de sortie du neurone en approximation des fonctions. L'effet de ce paramètre est très sensible dans l'entraînement. Celui ci est adapté tout comme les poids synaptiques. D'ailleurs en calcul, il peut être considéré comme un poids synaptique relié à une entrée constante.
Les poids et les biais sont théoriquement figés pendant l'utilisation. C'est pendant l'apprentissage qu'ils varient. Ainsi notre notation $w_{ij}^l(t)$ et $\theta_{ij}^l(t)$, dans l'équation (IV.20), qui indique qu'ils sont variables (ce qui n'est valable que si le réseau effectue un apprentissage), est générale.

IV.4.2.1 Apprentissage des réseaux LBF

L'apprentissage de ce réseau consiste à minimiser un critère quadratique définissant l'écart entre la sortie du réseau et celle désirée.

$$E_p = \sum_{i=1}^{n_L} e_i^p = \sum_{i=1}^{n_L} \left(d^p - y_i^L\right)^2 \qquad \text{(IV.21)}$$

où d^p représentent la sortie désirée pour la p *éme* entrée.

Nous pressentons ici les principaux algorithmes d'apprentissage utilisés.

1 Backpropagation

C'est Werbos qui, en 1972, a le premier mis en oeuvre cet algorithme avec notamment, son fondement mathématique. Il avait appelé cette méthode *The Dynamic Algorithm Feed-back* [Wer90].
En 1986, un groupe de chercheur de l'université de Stanford aux Etats Unis, Rumelhart, Hinton et Williams, l'ont développé et l'ont complètement adapté aux réseaux multicouches. Cet algorithme a été publié dans leur ouvrage sur les processus distribués parallèles (*Parallel Distributed Processing)* [Rum86]. Ils ont appelé la méthode Backpropagation en référence à l'erreur qui se "rétro-propage" à travers les couches du réseau (figure IV.5).

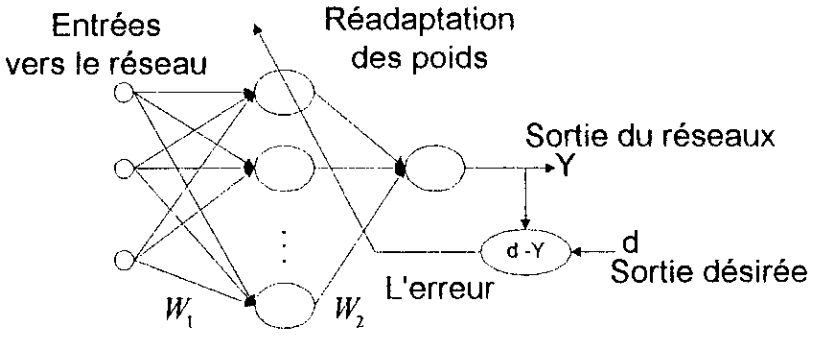


Fig.IV.5. Principe de l'entraînement du réseau par rétro-propagation de l'erreur.

Cette méthode n'est rien d'autre que l'application de l'algorithme de descente de gradient (cf.IV.3.1.2) sur ces réseaux de neurones. Contrairement au MADALINE, pour ce réseaux seuls les poids synaptiques directement connectés aux neurones de sortie ont un lien direct avec l'erreur à minimiser. Le problème qui se pose donc est le calcul du Jacobien de la sortie du réseau (équation IV.18) par rapport aux poids synaptiques des couches de neurones qui ne lui sont pas directement liés.
D.E.Rumelhart et G.E.Hinton ont proposé d'utiliser un enchaînement de différentiations calculant des dérivés partielles successives à partir de la couche de sortie jusqu'aux poids synaptiques en question [Rum86, Wid90]. La solution de P.J.Werbos, Plus rigoureuse mathématiquement, utilise par contre Les Dérivés Ordonnés (Ordered Derivatives) [Wer90] afin de calculer les dérivés de l'erreur par rapport à tous poids synaptiques du réseau.

Nous présentons ci dessous l' algorithme qui y résulte.

Algorithme de Backpropagation

Au début, il faut toujours initialiser les poids synaptiques à de petites valeurs aléatoires comprises en général entre -1 et 1.

1. Présenter un nouvel exemple X^p à l'entrée du réseau:

2. Calculer la sortie du réseau et l'erreur en sortie:

$$E_p = \sum_{i=1}^{n_L} \left(d_i^p - y_i^L\right)^2 \qquad \text{(IV.22)}$$

3. Calculer les valeurs de réadaptation des poids:

$$[\Delta w_{ij}^l]_p = \eta f'\left(u_i^l(t)\right) \delta_i^l y_j^{l-1} \tag{IV.23}$$

où η est le paramètre qui contrôle la vitesse de l'apprentissage appelé *Taux d'apprentissage.*

Avec: $\delta_i^l = \begin{cases} \left(d_i^p - y_i^L\right) & \text{pour les neurones de la couche de sortie.} \\ \sum_{k=1}^{n_{l+1}} \delta_k^{l+1} w_{ki}^{l+1} & \text{pour les neurones des autres couches.} \end{cases}$ (IV.24)

4.

– S'il s'agit d'un Data Learning (FFN pattern), réadapter les poids:

- poser : $\Delta w_{ij}^l(t) = [\Delta w_{ij}^l(t)]_p$
- Aller à 5.

– S il s'agit d'un Block Learning (FFN Batch):

- Si $p \neq M$ (M étant le nombre d'exemple d'entraînement), retourner à **1.**
- Sinon: $\Delta w_{ij}^l(t) = \sum_{p=1}^{p=M} [\Delta w_{ij}^l(t)]_p$ (IV.25)

5. Réadapter les poids

$$w_{ij}^l(t+1) = w_{ij}^l(t) + \Delta w_{ij}^l(t) \tag{IV.26}$$

6. Répéter de **1.** à **5.** autant de fois que nécessaire jusqu'à convergence vers le seuil d'erreur fixé.

Comme le montre l'algorithme ci-dessus, on peut opérer avec cette méthode de deux maniérés différente: *Data Adaptive Learning* ou *Block Adaptive Learning*, que nous examinerons successivement.

a. Data Adaptive Learning(FFN Pattern)

En utilisant cette technique, les poids sont réadaptés au passage de chaque exemple. De cette manière, le processus devient sensible à chaque exemple individuellement, ce qui le rend donc, facilement influençable par les bruits que peuvent contenir ces entrées durant l'entraînement. Cette technique n'est donc utilisée que pour un apprentissage en temps réel - qui ne nous donne d'ailleurs, pas le choix-; C'est pour cela qu'elle est parfois appelée *"On Line Optimisation méthode"*[Jer93, Kun94].

b. Block Adaptive Learning(Batch Learning)

Dans ce cas, on ne réadapte les poids qu'après passage de tous les exemples d'entraînement. La réadaptation est donc plus "prudente". Le réajustement se fait suivant la moyenne de tous les exemples et la méthode est donc beaucoup moins sensible au bruit que peuvent contenir des exemples singuliers pouvant se présenter. C'est donc une méthode plus robuste et si l'application ne nécessite pas un apprentissage en temps réel cette technique est préférable à la première[Qin92].

Etude du Taux d'apprentissage

La version théorique de la Backpropagation 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 changements plus importants dans les valeurs 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 capital 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: 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 optimale de ce paramètre qui peut nous rapprocher le plus des valeurs désirées en sortie est calculée [Kun93]. Nous expliquons ci dessous cette méthode.

Soit $\hat{Y}\left(X^p,W^p\right)$, le vecteur de sorties 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 Backpropagation à cette étape, on a:

$$\hat{Y}\left(X^p,W^{(p+1)}\right) \approx \hat{Y}\left(X^p,W^p\right) + \left[\frac{\partial \hat{Y}\left(X^p,W^p\right)}{\partial W}\right]^T \Delta W^p \qquad \text{(IV.27)}$$

En égalisant cette sortie à la sortie désirée (ce que nous désirons obtenir), et en remplaçant les réadaptations des poids reliés à la sortie issus de l'algorithme de Backpropagation (équations (IV.23), (IV.24) et (IV.25)), dans la relation (IV.27), On obtient aisément une valeur du taux pour cette étape:

$$\eta_p = \frac{1}{\left\| \frac{\partial \hat{Y}\left(X^p,W^p\right)}{\partial W} \right\|^2} \qquad \text{(IV.28)}$$

Dans le présent travail, afin d'augmenter la rapidité de convergence, nous avons utilisé une autre technique plus simple pour la modification du taux d'apprentissage. Cette technique consiste à mettre en oeuvre un algorithme qui a pour rôle de contrôler l'erreur d'entraînement à chaque étape. Ainsi pour un *Block Learning*, à chaque fois que l'erreur

actuelle dépasse la précédente par un certain seuil précédemment fixé - ce qui veut dire une augmentation des oscillations d'où risque de divergence -, on rejette les poids générés, on revient au point précèdent, et on diminue le taux d'apprentissage. Si par contre l'erreur diminue, les poids génèrès 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 plus rapide avec une bonne précision d'entraînement.(Figure IV.6).

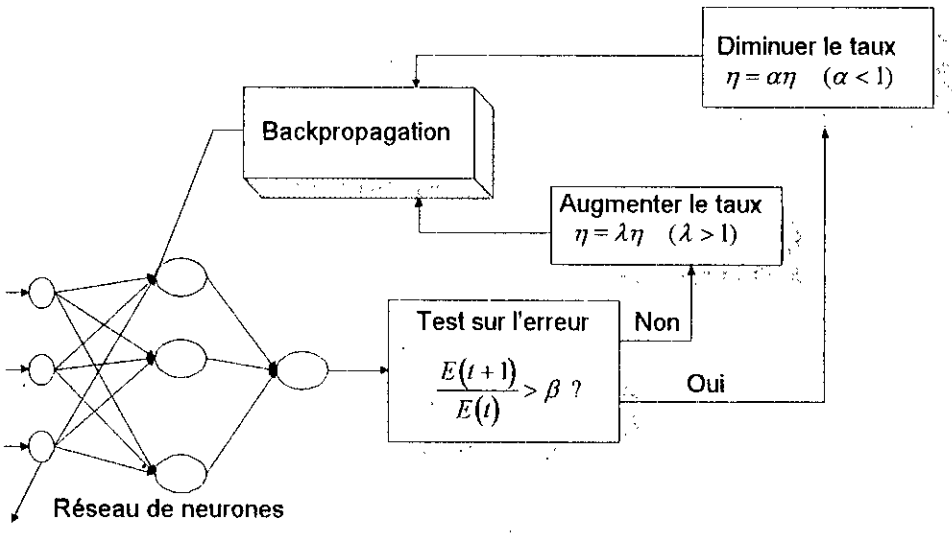


Fig. IV.6 Schéma de principe d'adaptation du taux d'apprentissage

2. Variantes de la Backpropagation

Ce qui est recherché dans un algorithme d'entraînement, c'est la stabilité du processus et la rapidité de convergence avec une bonne précision.
Depuis l'apparition de la Backpropagation, quelques variantes visant à améliorer les performances de cet algorithme ont été mises en oeuvre.
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 Backpropagation avec une Fonction Objective dite généralisée.
Cette méthode permet de changer le critère à minimiser au fur et à mesure que la convergence devient lente [Kar93].
Le critère proposé est décrit par:

$$\begin{aligned} G(\lambda) &= \lambda E_1 + (1-\lambda)E_2 \\ &= \lambda \sum_{k=1}^{M}\sum_{i=1}^{n_L} \frac{1}{2}\left(e_i^k\right)^2 + (1-\lambda)\sum_{k=1}^{M}\sum_{i=1}^{n_L} \Theta\left(e_i^k\right) \end{aligned} \qquad \text{(IV.29)}$$

Où λ est un paramètre tel que $\lambda \in [0,1]$.

Le coté droit de l'équation (IV.29) contient la somme de deux expressions. Le premier terme constitue la Fonction *Objectif* habituelle (Equation IV.21) appliquée sur tous les exemples d'entraînement. Quant au second terme, c'est une fonction Θ(.) appliquée sur l'erreur. Celle ci doit être définie positive, continue et differentiable et doit approximer assymptotiquement l'erreur absolue e_i^k.

Cette fonction est choisie pour un réseau LBF comme étant:

$$\Theta\left(d_i^k - y_i^k\right) = d_i^k\left(d_i^k - y_i^k\right) \tag{IV.30}$$

Où y_i^k représente la sortie du *i* éme neurone de la couche de sortie du réseau pour le *k éme* exemple en entrée.

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

$$w_{ij}^{k+1} = w_{ij}^k - \eta_k \frac{\partial G(\lambda)}{\partial w_{ij}^k} \tag{IV.31}$$

Au début de l'apprentissage, on fixe le paramètre $\lambda=1$, ce qui revient à minimiser le premier terme seulement de l'équation (IV.31). Sa valeur doit, par la suite, diminuer progressivement avec la diminution de l'erreur et l'augmentation du nombre d'itérations, jusqu'à atteindre *0* à la fin de l'apprentissage. Ceci, afin de permettre au second terme de prendre de l'importance au détriment du premier, dans l'expression du critère définie par l'équation (IV.29), 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:

$$\lambda = \lambda(E) = \exp\left(-\mu / E^n\right) \tag{IV.33}$$

Avec μ un nombre positif réel et n un entier positif. Une étude sur les valeurs de ces paramètres est faite sur N.B.Karayanis, 1993 [Kar93].

b. Robust Backpropagation

Cette méthode relève de la discipline des statistiques robustes. La robustesse dans ce contexte, fait référence à l'insensibilité aux perturbations affectant une loi de probabilité de distribution, qui gouverne un processus donné. En pratique, des estimateurs robustes ont la caractéristique d'ignorer les échantillons issus des observations infréquantes, qui diffèrent d'une manière significative par rapport aux autres. Dans le domaine de la statistique, des lois de densité réputées être robustes existent, parmi lesquelles on trouve la loi de Gauss ou celle de Cauchy [Kos92].
Une fonction linéaire n'est jamais robuste.
Des densités de probabilité basée sur la fonction *Logistic*

$$S(E_p) = \tanh\left(\frac{E_p}{2}\right) \tag{IV.34}$$

où sur la fonction de *Cauchy*

$$S(E_p) = \frac{2E_p}{1+E_p^2} \tag{IV.35}$$

sont par contre, des fonctions robustes ayant chacune ses propres caractéristiques.

Afin de profiter de cette robustesse, B.Kosko propose d'appliquer l'algorithme de Backpropagation en remplaçant le critère d'erreur défini dans l'équation (IV.21) par une fonction d'erreur $S(E_p)$ robuste qui lui est appliquée [Kos92].

c. Backpropagation avec Momentum

C'est une méthode très importante et efficace. Actuellement, la Backpropagation est pratiquement toujours, utilisée 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.
D. E. Rumelhart [Rum86] a proposé une solution qui consiste à utiliser les changements précédents des poids pour la réadaptation des poids actuels.
l'équation d'adaptation devient donc:

$$w_{ij}^l(t+1) = w_{ij}^l(t) + \eta \Delta w_{ij}^l(t) + \alpha \Delta w_{ij}^l(t-1) \tag{IV.36}$$

Le terme ajouté est appelé Momuntum (élan, quantité de mouvement), en analogie avec la Mécanique Classique, où 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 est 0.9 [Wid90]. 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 intéressants. D. W. Patterson [Pat96] 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.

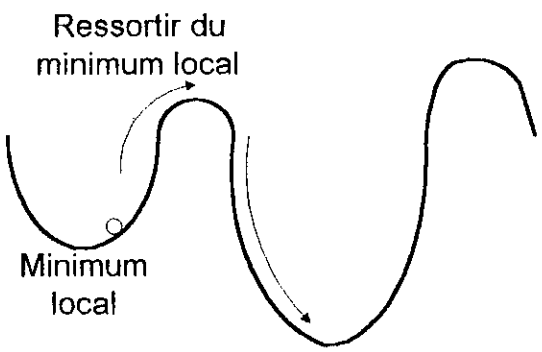


Fig. IV.7 Effet du Momentum pour échapper d'un minimum local

L'utilisation de cette méthode permet de faire sortir les poids des minimums locaux, afin de chercher d'autres optimums (figure IV.7), ce qui donne beaucoup de chances d'aboutir à un minimum global.

Le Momuntum tend d'autre part, à éviter les grands "sauts" pendant l'apprentissage, générés par le changement de la pente dans la surface d'erreur. Celui ci encourage ainsi le mouvement dans la même direction, pendant les étapes successives. Ce paramètre fait donc un lissage (filtrage passe bas) sur le parcours de la surface d'erreur [Bil91, Pat96].
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.

3. Méthodes d'optimisation de 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 seconde dérivée nous fournit, en effet, plus d'informations sur cette fonction. Ainsi d'une manière directe ou indirecte, la matrice Hessian H doit être utilisée. En effet, c'est le Hessian 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 [Kun93].
Cette dernière technique consiste à minimiser la fonction d'énergie qui est de la forme [Jer93]:

$$E(k+1) = E(k) + \dot{E}(k)\Delta W(k) + \frac{1}{2}\Delta W(k)^T \ddot{E}(k)\Delta W(k) \qquad \text{(IV.37)}$$

La surface de l'erreur est considérée comme étant quadratique.

Pour chercher le minimum, on doit choisir Δw tel que:

$$\frac{\partial E(k+1)}{\partial \Delta W} = 0 \qquad \text{IV.38)}$$

D'où l'on obtient:

$$\Delta W(k) = -\eta_k \left[\ddot{E}(k)\right]^{-1} \dot{E}(k) \qquad \text{(IV.39)}$$

où η_k représente le taux d'apprentissage qui est positif et de valeur petite.

3.1 Méthode du Gradient Conjugué

Afin de minimiser l'équation (IV.37), on se donne une direction d vers laquelle la recherche des réadaptations des poids est orientée afin d'atteindre le minimum.

$$\Delta W(k) = \eta_k d_k \qquad \text{(IV.40)}$$

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

$$d_k = -\frac{\partial E(k)}{\partial W(k)} + \beta_{k-1} d_{k-1} \qquad \text{(IV.41)}$$

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:

$$d_k^T \left.\frac{\partial E}{\partial W}\right|_{w+\eta_k d_k} = 0 \qquad \text{(IV.42)}$$

En utilisant l'équation (IV.42) dans (IV.37) et en l'appliquant sur (IV.40), on tire la valeur de η_k qui représente la valeur optimale avec laquelle les poids doivent être déplacés dans cette direction à chaque étape :

$$\eta_k = \frac{d_k^T \dfrac{\partial E(k)}{\partial W}}{d_k^T H d_k} \qquad \text{(IV.43)}$$

où H représente le Hessian de l'erreur à l'instant k.

De plus, il est même possible d'utiliser les relations (IV.40) et (IV.41) pour calculer la dérivée de l'équation (IV.37) par rapport au paramètre β_{k-1}, afin de tirer sa valeur optimale pour l'utiliser dans (IV.41) [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].

Nous résumons ci dessous cet algorithme.

Algorithme du Gradient Conjugué

1. Initialiser les poids synaptique W_0 et les directions de recherches initiales d_0

2. Calculer le critère d'erreur quadratique E_k après passage de tous les exemples d'apprentissage
2. Réajuster la direction à l'aide de l'équation (IV.41) avec le paramètre β_{k-1} qui peut être calculé à partir de (IV.37) et (IV.41) comme expliqué ci dessus.

3. Réajuster les poids suivant la direction adoptée (équation IV.40).

$$W_{k+1} = w_k + \eta_k d_k$$

où le taux d'apprentissage r_k est calculé à partir de l'équation (IV.43)

Par rapport à la Backpropagation, cette méthode permet la convergence plus rapidement ou, tout au plus, après le même nombre d'itérations. Néanmoins, celle ci comporte plus de paramètres, dont les valeurs doivent être convenablement choisies ou ajuster à chaque itération, ce qui nécessite plus de calcul et complique l'implémentation.

L'utilisation de la dérivée seconde rend le processus encore plus sensible au bruit. De ce fait, cette méthode n'est conseillée que dans le cas d'un *Block Learning*. En outre, le calcul du Hessian entraîne des complications. Pour cela, des méthodes numériques ont été mises au point pour son approximation [Kun90].

Notant, par ailleurs, que Krayannis et Vanetsanopoulos ont proposé une technique d'approximation numérique évitant le calcul du Hessian et de son inverse à l'aide d'une méthode récursive qu'ils ont proposée [Kra92].

3.2 Méthode de Leavenberg Marquard

La méthode de Leavenberg-Marquart est l'une des plus utilisées

Cette méthode consiste à considérer la surface d'erreur quadratique, en se basant sur la fonction d'énergie (IV.37).

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 inclue en plus le biais:

$$W_i^l = \left[w_{i1}^l, w_{i2}^l \ldots w_{in_{l-1}}^l, \theta_i^l \right] \tag{IV.43}$$

Soit F_i^l les vecteurs 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 la p *éme* entrée, on a:

$$F_i^L = \sum_{p=1}^{M} \sum_{i=1}^{n_l} \left(d_i^p - y_i^L \right) \left[-\frac{\partial y_i^L}{\partial W_i^l} \right] \tag{IV.44}$$

où M représente le nombre d'exemples d'entraînement, n_l le nombre de neurones dans la d couche l, y_i^L la *iéme* sortie du réseau, et d_i^p sa sortie désirée pour la p *éme* entrée.

On défini F un seul grand vecteur rassemblant tous les vecteurs F_i^l définis par l'équation (IV.44). A partir de (IV.37) et (IV.38), afin de trouver l'optimum on obtient

$$H \, \Delta W = -F \tag{IV 45}$$

où H représente la matrice Hessien de la fonction erreur en sortie du réseau (équ.IV.22).

L'équation (IV.45) 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 [Kol89]:

$$H = FF^T + \lambda \Omega \tag{IV.46}$$

Où λ est un coefficient positif. La matrice FF^T étant définie semi positive, le 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 (IV.37), afin de garantir la convergence de l'algorithme vers un minimum local. 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.

C'est cette dernière technique de Leavenberg-Marquard qui sera adoptée au court de notre utilisation de la méthode du Gradient de second ordre dans nos applications.

Voici les étapes à suivre lors de l'application de cette méthode à chaque itération.

Algorithme de la méthode de Leavenberg-Marquard

1. Pour la première itération initialiser les poids synaptique W et choisir une valeur pour λ de préférence petite et inférieur à 1.

2. Calculer le critère d'erreur quadratique E_p après passage de chaque exemple d'apprentissage.

3. Calculer les vecteur F_i^l des dérivées de l'erreur par rapport aux vecteurs poids du réseau (équ.IV.43),.définis par l'équation (IV.44). Former le grands vecteur F.

4. Calculer la matrice Hessien H à l'aide de l'équation (IV.46).

5. Aprés passage de tous les exemples d'apprentissage ($p=M$), déterminer Δw en résolvant le système linéaire (IV.45) pour l'erreur globale (Bloc Learning). La méthode de Gauss-Seidel est généralement utilisée.

Il existe d'autres variantes de cet algorithme, visant à le rendre plus efficace. Celles ci jouent, généralement, sur le choix optimal de λ, permettant de conduire vers un minimum local. Elles peuvent aussi agir notamment sur le contrôle du gradient, qui en diminuant ne permet plus de poursuivre la prospection dans d'autre direction, à la recherche d'un minimum. D'autres techniques consistent à réadapter la méthode, à travers la création d'autres paramètres, afin d'adapter le problème quand celui ci ne répond pas bien au modèle quadratique défini par l'équation equ.IV.37. Toutefois ces techniques exigent plus de calculs et rendent souvent l'algorithme plus compliqué à l'implémentation [Kun93, Jer93].

Ces méthode d'apprentissage basées sur le gradient de second ordre sont rapides à la convergence. Mais celles ci, en plus des calculs qui sont plus compliqués, restent sensibles au bruit et ne sont conseillées que pour un apprentissage en *Bloc Learning.*

4. Méthode d'Optimisation Aléatoire ROM

Ce genre de méthodes est basé sur la génération de séquences aléatoires, afin de déplacer les valeurs des poids synaptiques dans l'espace, à la recherche de l'optimum. La combinaison fournissant le meilleur résultat est retenu à chaque itération.
Cette technique a l'avantage d'être plus souple. En effet partant du principe de minimiser un critère, ces méthodes ne dépendent nullement de la nature mathématique du système sur lesquels elles sont appliquées. Elle ne nécessite pas de déterminer le gradient ou de faire un quelconque calcul dépendant du critère à minimiser.
D'autre part, cette technique permet d'échapper aux minimums locaux. Il suffit, pour cela, qu'une séquence aléatoire permettant une meilleure performance du critère se présente.

En 1965, J.Mathias a présenté une méthode aléatoire d'optimisation. Celle ci a été reprise et développée par F.J.Sollis et J.B.Wetts en 1981 [Sol81], qui ont démontré la convergence de l'algorithme vers un minimum global. Cette méthode a été enfin appliquée avec sucés, aux réseaux de neurones LBF multicouches par N.S.Baba en 1989 [Bab89].
Nous présentons ci dessous l'algorithme de cette méthode:

Algorithme de La Méthode ROM de Solis & wets Appliquée aux Réseaux de Neurones

Cette méthode utilise un bruit Gaussien pour générer les séquences aléatoires nécessaires pour la réadaptation des poids du réseau.

1. Initialiser tous les poids W du réseau Aléatoirement. Initialiser, la variance v et la moyenne de la séquence $b(0)=0$.

2. Générer une nouvelle séquence aléatoire $\zeta(k)$ de moyenne $b(k)$ et variance $v(k)$.

3. Calculer les erreurs $E = \sum_{p=1}^{M} E_p$ à la sortie du réseau pour chaque cas:

$$E(W(k)),\ E(W(k)+\xi(k)),\ \text{et}\ E(W(k)-\xi(k))$$

4.

- Si $E(W(k)+\xi(k)) < E(W(k))$, alors

$$W(k+1) = W(k)+\xi(k) \text{ et}$$
$$b(k+1) = 0.4\xi(k)+0.2b(k)$$

- Si $E(W(k)+\xi(k)) \geq E(W(k))$ et $E(W(k)-\xi(k)) < E(W(k))$, alors

$$W(k+1) = W(k)-\xi(k) \text{ et}$$
$$b(k+1) = b(k)-0.4\xi(k)$$

(IV.47)

- Sinon,

$$W(k+1) = W(k) \text{ et}$$
$$b(k+1) = 0.5b(k)$$

5. Poser $k=k+1$ et refaire de *2* à *4* jusqu'à l'obtention de l'erreur désirée.

Contrairement à la méthode de Backpropagation où le calcul du Jacobien est nécessaire Le grand avantage de cette méthode est son indépendance vis à vis du critère à optimiser sur lequel elle est appliquée. Cette caractéristique nous a amenés à l'utiliser en commande où le modèle du système est inconnu, celle ci a montré de bonnes aptitudes.
Néanmoins, pour l'implémentation elle nécessite plus de calculs, à savoir la génération du bruit Gaussien et les tests à effectuer sur l'erreur. Ce dernier point constitue son plus grand défaut. défaut.

L'utilisation de la ROM dépend de la variance du vecteur Gaussien. Une séquence avec une grande variance génère de grands changements dans l'erreur mais jamais des oscillations. Le cas échéant, elle marque des stagnations sur la surface d'erreur jusqu'à l'arrivée d'une séquence favorable. Il est intéressant de commencer l'entraînement avec une variance importante et de la faire baisser au fur et à mesure que l'entraînement avance.
Une méthode plus efficace consiste à commencer l'entraînement avec une variance de valeur importante et la multiplier par un paramètre réducteur à chaque fois que l'erreur se "bloque" pendant un certain temps sur un point [Yed95]. C'est cette technique que nous utiliserons avec la ROM dans ce travail.

En plus de ces méthodes, les techniques d'entraînement font encore l'objet de plusieurs travaux de recherches qui ont pour objectif d'améliorer encore l'apprentissage et notamment de le simplifier. Nous citons dans ce contexte la méthode mise au point par S.Ergezinger et E.Thomsen appelée *Méthode d'Optimisation Couche par Couche O.L.L (Optimization Layer by Layer Learning Method)*. Cette méthode consiste à linéariser les sorties non-linéaires (sigmoïdes) des neurones cachés en les développant en séries de Taylor de premier ordre autour des poids synaptiques calculés à chaque étape. De cette manière, l'entraînement revient à un problème d'optimisation linéaire d'un critère quadratique. Ceci permet de l'accélérer en procédant couche par couche à partir des poids liés à la sortie jusqu'à ceux en provenance de l'entrée du réseau. Afin de compenser les incertitudes dues à cette approximation des sorties du réseau, un terme de pénalité est ajouté à la fonction coût à minimiser. son rôle est de réduire l'effet des termes d'ordre supérieur négligés dans le développement en série de Taylor [Erg95].

IV.4.2.2 Réseaux LBF et approximation de fonctions:

L'approximation de fonctions constitue l'un des problèmes les plus importants en Mathématique. Le plus ancien problème connu est l'approximation d'une fonction continue, à plusieurs variables par une combinaison de fonctions continues à une seule variable. C'est le 13 éme des 23 fameux problèmes que Hilbert avait énoncé en 1900 pour orienter la recherche au début du siècle[Bla91].

Spretcher a énoncé, en 1965 un théorème -qui est en fait un développement du théorème de Kolmogorov de 1957-, stipulant que toute fonction continue sur l'intervalle $[0,1]^n$ peut être représentée, en utilisant seulement une combinaison linéaire de fonctions non linéaires continues et croissantes, à une seule variable:

$\forall n \geq 2, \exists \varphi: [0, 1] \rightarrow [0, 1]$, φ croissante

$\forall \eta \geq 0, \exists \varepsilon \ 0 < \varepsilon < \eta \ \forall f$ sur I^n

$$f(x) = \sum_{j=1}^{2n+1} \chi\left(\sum_{i=1}^{n} \lambda^i \varphi\left(x_i + \varepsilon(j-1)\right) + j - 1 \right)$$

où χ est une fonction continue et φ est une fonction continue et monotone.

En se plaçant dans le contexte de réseaux de neurones, l'équation (IV.48) contenant des doubles sommation peut être interprétée comme un réseau ayant deux couches cachées possédant respectivement *n(2n+1)* et *2n+1* de neurones. Sur la base de cela H. Nielsen a démontré en 1987 que n'importe quelle fonction peut être approximée par un réseau à quatre couches.
Mais la nature des fonctions d'activation de ces neurones était encore ambiguë. C'est Kreinovich, qui en se basant sur le théorème de Weishtrass sur l'approximation de fonctions multivariables linéaires par des régressions polynomiales, a prouvé que ces fonctions doivent être continues, monotones et dérivables [Kos92]. Ceci a orienté la plupart des chercheurs vers la fonction *Logistic (Sigmoïde)* qui, en plus, est en parfait accord avec le modèle de Mc Culloch et Pitts. C'est la classe de réseaux basés sur la fonction *discriminant* linéaire LBF.

Hornik et White ont montré en 1989 qu'une architecture neurale à sortie sigmoïde peut approximer n'importe quelle fonction Borel avec la précision désirée, à condition d'avoir assez de neurones entre la couche d'entrée et celle de sortie.
La question posée est de déterminer le nombre de couches cachées à utiliser et le nombre de neurones nécessaires pour ces couches.

Cybenko a prouvé en 1990 qu'une couche cachée suffit pour approximer n'importe quelle fonction. Mais l'inconvénient est que le nombre de neurones que doit contenir cette couche peut être "astronomique"(c'est le terme qu'il avait utilisé lui même) [Cyb89].
Chester a démontré que deux couches cachées peuvent assurer mieux l'approximation de n'importe quelle fonction continue [Hun92].

Enfin Cybenko et Funahashi ont émis un théorème en 1991 selon lequel, si *f(X)* est une fonction continue croissante et réelle, et *t(X)* une fonction continue non constante dans un sous espace S dans R
$\forall \varepsilon > 0, \exists K \in N, w_{ij} \in R, \theta_i \in R$ tel que:

$$\phi = \sum c_i f\left(\sum w_{ij} x_i + \theta_i \right) \qquad \text{(IV.48)}$$

satisfaisant: $|\phi(X,W) - t(X)| < \varepsilon$ pour $t, X \in S$

Ce dernier théorème prouve que les LBF sont capables d'approximer n'importe quelle fonction continue dans l'espace [Kun93].

IV.4.2.3 Dimension du réseau, l'entraînement et le dilemme Précision-Généralisation

La dimension d'un réseau permettant d'obtenir un meilleur résultat est impossible à fixer.
Il a été établi qu'un réseau à une seule couche cachée, peut faire l'approximation de n'importe quelle fonction [Cyb89], mais le nombre de neurones cachés n'a jamais pu être spécifié pour un problème donné.

Plusieurs travaux ont été menés sur cette question. E. B. Baum et Haussler [Bau89] ont proposé une technique où le nombre de neurones dépend du nombre d'entrées n_X, du nombre de sorties n_Y, et de celui d'exemples n_P.
D'où, un intervalle où le nombre de poids n_W nécessaire peut varier est donné:

$$\frac{n_Y n_P}{1+\log_2(n_P)} \le n_W \le n_y\left(\frac{n_P}{n_x}+1\right)(n_X+n_Y+n_P)+n_y$$

En appliquant ce résultat, Le nombre de neurones peut devenir trop important avec l'augmentation du nombre des exemples d'entraînement.
D. Chester [Hun92] a démontré que deux couches cachées avec un nombre de neurones moins important peuvent assurer d'avantage de précision avec une bonne généralisation.

En prenant toutes ces notions en considération, nous pouvons noter que c'est surtout l'expérience et le nombre d'essais qui nous oriente dans la recherche du nombre de neurones nécessaire pour un problème donné.
Il faut cependant savoir qu'un nombre important de neurones, bien que compliquant les calculs, permet d'obtenir une bonne **précision d'entraînement**. Un nombre insuffisant par contre entraînera sa dégradation.
La précision d'entraînement ne concerne, malheureusement, que les exemples avec lesquels le réseau a effectué son apprentissage; or cet apprentissage est surtout destiné à rendre le réseau capable de répondre correctement aux exemples non présents pendant l'entraînement: C'est **la généralisation**.
Un réseau trop chargé en neurones cachés souffre de manque de généralisation, échouant ainsi devant des exemples autres que ceux présents dans la base d'apprentissage. Un nombre réduit de neurones qui devrait être favorable à la généralisation rend la précision d'entraînement très difficile à obtenir, si ce n'est impossible. Ainsi augmenter la dimension est favorable à la précision d'entraînement et défavorable à la généralisation. Sa diminution en revanche améliore la généralisation mais au détriment de la précision. Ce problème constitue le dilemme **Précision-Généralisation.** Il faut donc réussir un compromis entre les deux.

D'autre part un apprentissage misant beaucoup sur la précision d'entraînement, visant donc à trop minimiser l'erreur en sortie conduit à un **surentraînement.** Ce phénomène rend le réseau carrément incapable de généraliser. En effet en "poussant" l'entraînement très loin, le réseau se spécialise sur les exemples d'apprentissage auxquelles une précision très importante lui a été demandée, lui retirant ainsi son pouvoir d'interpolation.
Un autre problème important dont souffrent les réseaux LBF est la difficulté de subir l'apprentissage avec de nouveaux exemples sans perdre les caractéristiques déjà acquises. En effet la fonction d'activation des neurones qui est non locale favorise ce phénomène d'*oubli* appelé *effet de non entraînement* et rend ces réseaux souvent incapables d'apprendre de nouveaux exemples après avoir déjà subi un premier apprentissage.[Ren95] Ceci les rend non bien adapté pour un apprentissage en temps réel pendant le déroulement de processus, notamment dans le cas de variation de leurs paramètres.

IV.4.3. Réseaux à Fonction de Base Radiale RBF

IV.4.3.1 Principes de base

Les réseaux RBF représentent des réseaux à une seule couche cachée qui peuvent être utilisés aussi bien pour la classification que pour l'approximation de fonctions.
A travers une combinaison linéaire de fonctions non-linéaires de base radiale, le fonctionnement de ces réseaux repose sur le principe des estimateurs à noyau, et le généralise du monovariable vers le multivariables.
Un estimateur à noyau considère des fonctions de R^+ vers R qui sont de la forme[Hér93]:

$$\Phi_i(\chi) = \Phi_i(\|\chi - \xi_i\|). \tag{IV.49}$$

L'idée principale a été introduite par M.J.D.Powell. Elle est basée sur l'interpolation.
Toute fonction $f(\chi)$ d'une variable $\chi \in R^n$ peut être approchée par une interpolation composée par la somme de p fonctions noyaux de forme fixée $\Phi(.)$ [Hér94]:

$$f(\chi) = \sum_{i=1}^{p} \lambda_j \Phi(\|\chi - \xi_i\|) \tag{IV.50}$$

où ξ_j représentent les noeuds de l'interpolation pour $i=1,...,n$.

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

$$f(\chi_i) = \gamma_i \qquad i=1,...,n. \tag{IV.51}$$

et $\Phi()$ est une fonction assurant la continuité aux noeuds 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].

IV.4.3.2 Architecture et fonctionnement des Réseaux RBF

Broomhead & Lowe puis Che et al ont conçu un réseau de neurones à une seule couche cachée dont le fonctionnement est basé sur l'idée des approximateurs à noyau introduite ci-dessus [Bil92]. Ainsi à partir de l'équation (IV.49) et en la considérant dans le cas multivariable, la sortie que doit délivrer ce réseau est de la forme (fig.IV.8):

$$f_k(\chi) = \sum_{j=1}^{n} \lambda_{kj} \Phi(\|\chi - c_i\|) \qquad k=1,...n;\ i=1,...m \tag{VI.52}$$

où c_i sont des valeurs que nous attribuons à chaque neurone de la couche cachée appelés *centres de classes*.
λ_{ki} les poids synaptiques reliants ces neurones aux sorties.
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[Khe94]:

$$\begin{aligned}
&\text{Forme Cubique} && \Phi(r) = r^3 \\
&\text{Forme Multiquadratique} && \Phi(r) = \left(r^2 + k^2\right)^{1/2} \\
&\text{Forme Logarithmique decalee} && \Phi(r) = \log\left(r\left(r^2 + k^2\right)\right) \\
&\text{Forme Gaussienne} && \Phi(r) = \exp\left(-r^2/\beta^2\right)
\end{aligned} \qquad \text{(IV.53)}$$

où r représente ici un réel quelconque, remplaçant le discriminant des neurones.
Parmi ces formes, la fonction Gaussienne est la plus utilisée [Ren95, Kos92] en raison de ses caractéristiques, que nous avons déjà présenté,s (cf.I.2.3). C'est cette fonction que nous utiliserons dans notre travail.
La figure (IV.8) représente l'architecture d'un réseau RBF.

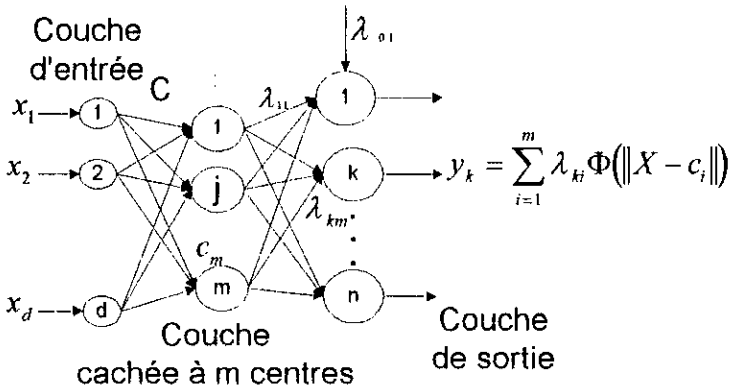


Fig. IV.8 Architecture d'un réseau RBF Multi-sorties

Celui ci est composé de trois couches (figure IV.8.). 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'entrée. En effet, chaque entrée du réseau 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:

$$d_i = \|X - c_i\| \qquad \text{(IV.54)}$$

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

$$s_i(X) = \Phi(\|X - c_i\|) \qquad 1 \le i \le m. \qquad \text{(IV.55)}$$

avec m, le nombre de neurones cachés.

Cette fonction est généralement Gausienne. D'où l'on a:

$$s_i(X) = \exp\left[-\frac{(X-c_i)^T(X-c_i)}{2\sigma_i^2}\right] \qquad \text{(IV.56)}$$

où σ_i est un paramètre qui représente une mesure de la dispersion des données associés à chaque noeud; il est souvent égal à la distance moyenne entre le centre de la classe et les exemples d'entraînement y faisant partie[Hus95].
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 aux 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:

$$y_k(X) = \sum_{i=1}^{m} \lambda_{ki} \Phi\left(\|X - c_i\|\right) \quad 1 \le k \le n \qquad \text{(IV.57)}$$

avec n le nombre de neurones de sorties.

IV.4.3.3 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

Nous présentons ci dessous un algorithme pour chacun des deux cas.

1. Méthode de Centrage Adaptatif (Adaptive Centering Method)

Comme pour les paramètres de pondération λ_{ji}, à toute valeur attribuée aux centres c_i correspondra une certaine erreur en sortie. Il est aisé de remarquer, à partir de l'équation (IV.57), 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 supervisé:

1. Initialiser les centres $C_i(0)$, les paramètres de la sortie Gaussienne de chaque neurone de la couche cachée $\sigma_i(0)$, $1 \le i \le m$ et les poids synaptiques $\lambda_{ji}(0)$, $1 \le j \le n$.

2. choisir un taux d'entraînement $\alpha<1$ variable ou fixe.

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

$$E = \sum_{p=1}^{M} \sum_{j=1}^{n} \left(d_j^p - y_j^p\right)^2 \qquad \text{(IV.58)}$$

Avec M représentant le nombre d'exemles d'entrainnement. y_j^p et d_j^p représentent réspectivement la *j éme* sortie du réseau et celle désirée pour le *p éme vecteur d'* entrées.

4. Réajuster les centres et éventuellement les paramètres des fonctions noyau (les Gaussiennes):

$$C_i(t+1) = C_i(t) + \alpha\, \Delta C_i(t)$$
$$\sigma_i(t+1) = \sigma_i(t) + \alpha\, \Delta \sigma_i(t)$$

avec: (IV.59)

$$\Delta C_i(t) = -\frac{\partial E}{\partial\, C_i(t)}$$
$$\Delta \sigma_i(t) = -\frac{\partial E}{\partial \sigma_i(t)}$$
$$1 \le i \le m$$

5. Réajuster les poids synaptiques

$$\lambda_{ji}(t+1) = \lambda_{ji}(t) + \alpha \Delta \lambda_{ji}(t)$$ (α: taux d'apprentissage).

avec: (IV.60)

$$\Delta \lambda_{ji}(t) = -\frac{\partial E}{\partial \lambda_{ji}(t)}, \quad 1 \le j \le n; \quad 1 \le i \le m$$

6. Répéter les étapes de **3.** à **5.** , le nombre de fois nécessaire jusqu'à obtention de la précision désirée en sortie.

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***, que nous avons déjà étudiée dans le chapitre II.
Les étapes de l'algorithme d'apprentissage du RBF dans ce cas, sont les suivantes:

1. Initialiser les centres $C_i(0)$, $1 \le i \le m$ avec des valeurs aléatoires et choisir un taux d'apprentissage initial $\alpha(0)<1$.

2. Calculer la distance euclidienne de l'exemple X^p par rapport à chaque classe.

$$dist_i(p) = \left\| X^p - C_i(p-1) \right\| \qquad 1 \le i \le m \tag{IV 61}$$

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

$$k = \arg\left[\min\left(dist_i(p)\right)\right] \qquad 1 \le i \le m \tag{IV.62}$$

Tel que *arg*[.] identiffie le rang du neurone (i) dans la couche cachée.

4. Réadapter les centres:

$$C_i(p) = C_i(p-1) \qquad \text{pour: } 1 \le i \le m \quad \text{et} \quad i \ne k \tag{IV.63}$$

$$C_i(p) = C_i(p-1) + \alpha(p)\left[X^p - C_i(p-1)\right] \qquad \text{pour } i=k \tag{IV.64}$$

5. Réduire le taux d'apprentissage:

$$\alpha(p) = \frac{\alpha(p-1)}{1 + \text{int}\left[\frac{p}{M}\right]^{1/2}} \tag{IV.65}$$

(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 (Voir *ART* chapitres 2), 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 écomnomiques 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*[Hus95].
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 permet 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 premiére 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 premiére téchnique constitue une extension des réseaux RBF vers des réseaux Gaussiens appelé *GPFN Gaussian 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* (cf. II.3.2). 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.

$$H_i = \left\{ x \ / \ d(x, c_i, \sigma_i) \le r_i^2 \right\}$$

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 λ_{ji} et les paramètres de dispersion σ_i sont ajustés suivant la méthode du gradient similaire à celle présentée en (IV.3.4). 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 recalculés et la précision en sortie est téstée Ainsi, les neurones cachés nécessaires sont recrutés, les uns aprés 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, Hus95].

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 Networks* développé par S.Lee et M.Kill

IV.4.3.4 Réseaux RBF et Approximation de Fonction:

A la recherche d'une bonne approximation Giroso et Pogio ont introduit en 1990 la notion de meilleur approximateur qu'ils définissent comme ce lui ayant la fonction la plus proche de celle à estimer[Hun90].
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 approximée 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.

IV.5 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 (IV.9) résume. Dans ce qui suit, nous allons étudier ces types de réseaux, ainsi que leurs algorithmes d'apprentissage

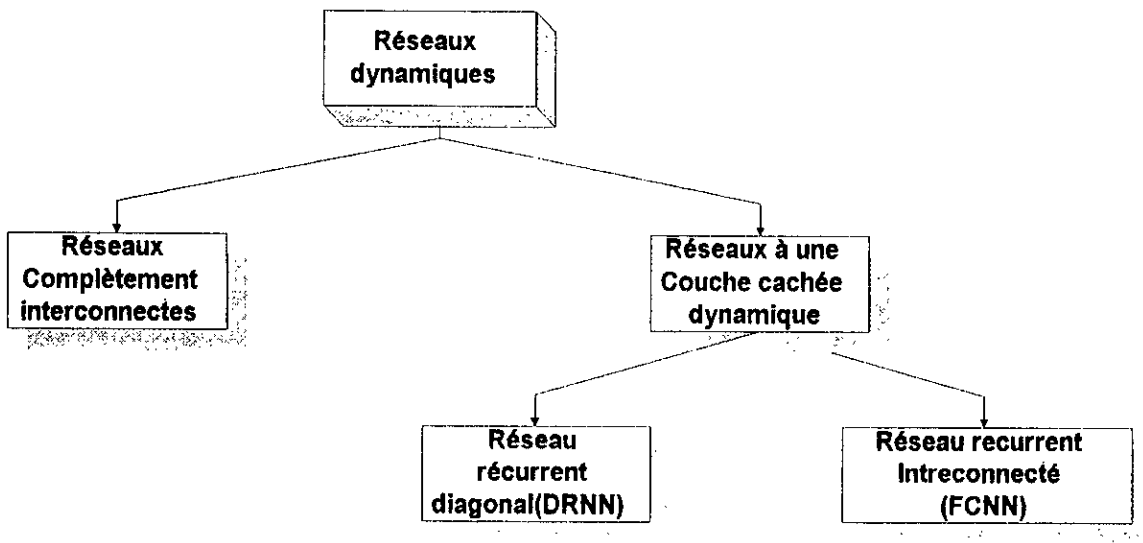


Fig. IV.9 Classification des réseaux récurrents suivant leurs architectures

IV.5.1 Réseaux de neurones complètement interconnectés

C'est le modèle le plus général de réseaux récurrents. Celui ci est, Généralement, constitués 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, il sont moins [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 signalé de ce type de réseaux.

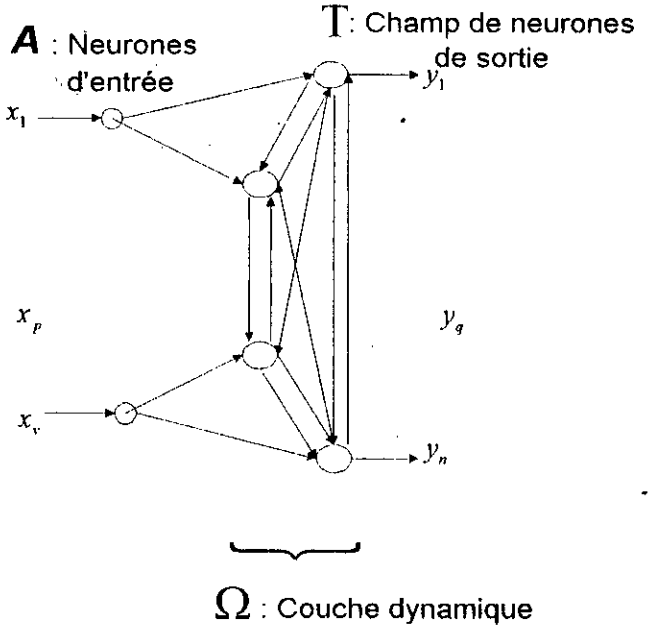


Fig. IV.9 Architecture d'un réseau Dynamique à structure complètement interconnecté.

1. Recurrent Backprobagation

Applicable aux réseaux entièrement interconnectés,. la Recurrent Backpropagation est basée sur la méthode d'apprentissage proposée par F. Almeida [Kar93]. Sur la figure (IV.9), nous avons représenté ce réseau par une couche d'entrée ordinaire et une couche dynalique entiérement connéctée.
Le comportement de la couche dynamique est régie par le système d'équations différentielles suivantes:

$$\tau \frac{dz_i}{dt} = -z_i + f(u_i) + I_i \qquad i=1,..n \qquad \text{(IV.66)}$$

n:nombre de neurones dynamiques.

où: z_i représente la sortie du i éme neurone.
I_i représente un biais externe tel que l'on a:

$$I_i = \begin{cases} x_i & \text{si le neurone } i \text{ est lié aux neurones d'entreé qui sont representés par le champs A} \\ 0 & \text{sinon} \end{cases} \quad \text{(IV.67)}$$

où x_i représente l'entrée exterieure injéctée au réseau.

avec:

$$u_i = \sum_j w_{ij} z_j \quad \text{(IV.68)}$$

f(.) est une fonction du type sigmoïde.

L'équation (IV.66) donne à l'équilibre:

$$\begin{aligned} z^*_i &= f\left(u^*_i\right) + I_i \\ &= f\left(\sum_j w_{ij} z^*_j\right) + I_i \end{aligned} \quad \text{(IV.69)}$$

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:

$$E = \frac{1}{2}\sum_k E_k^2 \quad \text{(IV.70)}$$

tel que:

$$E_k = \begin{cases} d_k - z^*_k = d_k - y^*_k & k \in \text{T (champs de sortie)} \\ 0 & \text{sinon.} \end{cases} \quad \text{(IV.71)}$$

où y_k représente la sortie du *k éme* neurone et d_k. représente sa sortie désirée
pour cela la méthode de Backpropagation est appliquée.
on a donc:

$$\Delta w_{pq} = -\alpha \sum_i \frac{\partial E_i}{\partial w_{pq}} = \sum_i \frac{\partial z^*_i}{\partial u^*_i} \frac{\partial u^*_i}{\partial w_{pq}} \quad \text{(α:taux d'apprentissag)} \quad \text{(IV.72)}$$

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 (IV.72), on tire la relation suivante propre aux réseaux dynamiques

$$\frac{\partial z^*_i}{\partial w_{pq}} = f'\left(u^*_i\right)\left(\delta_{ip} z^*_q + \sum_j w_{ij} \frac{\partial z^*_j}{\partial w_{pq}}\right) \quad \text{(IV.73)}$$

où δ représente le sembole de Kröneker.
La méthode de F.J Pineda, que nous expliquons dans l'annexe C., consiste à considérer un second réseau identique au premier, à chaque étape de l'apprentissage.
Ce réseau est définie par une relation identique à l'équation (IV.66):

$$\dot{v}_i = -v_i + \sum_p v_p f'\left(u_p\right) w_{pi} + E_i \quad \text{(IV.74)}$$

Cette méthode d'entraînement peut donc être résumée par l'application des étapes suivantes:
1. Obtenir l'état d'équilibre du réseau régi par l'équation (IV.66) .et Calculer l'erreur E à sa sortie.
2. Obtenir les états d'équilibre v_p^* du réseau auxiliaire défini par l'équation (IV.72).
3. La réadaptation des poids synaptiques est déduite à partir des équations (IV.72), (IV.73) (voir Annexe C)

$$\Delta w_{pq} = \alpha f'\left(u_p^*\right) v_p^* z_q^* \qquad \text{(IV.75)}$$

Pour les valeurs initiales des états du réseau principal, du fait que ce qui est rechercher n'est rien d'autre que les états d'équilibre, elles peuvent être initialisée 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 long, notamment pour l'établissement des états d'équilibre des équations (IV.66) et (IV.74). 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** [Bar 89]. Williams & Zipser ont proposé une méthode que nous présentons ci dessous.

2. 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 $t = t_1$.
Nous expliquons ci-dessous la technique utilisée.
Sous sa version discrète, le réseau de la figure (IV.9) est régit par [Wil89]:

$$u_k(t) = \sum_{l\in\Omega} w_{kl} y_l + \sum_{l\in A} w_{kl} x_l = \sum_{l\in A\cup\Omega} w_{kl} z_l(t) \qquad \text{(IV.76)}$$

$$y_k(t+1) = f\left[u_k(t)\right] \qquad k=1,\ldots n \qquad \text{(IV.77)}$$

Nottons 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:

$$E_k(t) = \begin{cases} d_k(t) - y_k(t) & k \in \mathrm{T} \text{ (champs de sortie)} \\ 0 & \text{sinon.} \end{cases} \qquad \text{(IV.78)}$$

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

$$J(t) = \frac{1}{2}\sum_{k\in\Omega}\left[E_k(t)\right]^2 \qquad \text{(IV.79)}$$

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

$$J_{totale}(t_0, t_1) = \sum_{t=t_0+1}^{t_1} J(t) \tag{IV.80}$$

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

$$\Delta w_{pq}(t) = -\alpha \frac{\partial J(t)}{\partial w_{pq}} \tag{IV.81}$$

De (IV.78), on a:

$$\frac{\partial J(t)}{\partial w_{pq}} = -\sum_{k \in \Omega} E_k(t) \frac{\partial y_k(t)}{\partial w_{pq}} \tag{IV.82}$$

Le problème est donc, toujours, le calcul de la dérivée sur l'équation (IV.82)
A partir de (IV.76) et (IV.77), on a:

$$\frac{\partial y_k(t+1)}{\partial w_{pq}} = f'\left(u_k(t)\right)\left(\delta_{pk}\, z_q + \sum_{j \in A \cup \Omega} w_{kj} \frac{\partial z_j}{\partial w_{pq}}\right) \tag{IV.83}$$

En posant :

$$p_{pq}^k(t) = \frac{\partial y_k(t)}{\partial w_{pq}} \tag{IV.84}$$

et en utilisant (IV 77), on tire une relation récurrente:

$$p_{pq}^k(t+1) = f'\left(u_k(t)\right)\left[\sum_{l \in \Omega} w_{kl} p_{pq}^l(t) + \delta_{pk} z_q(t)\right] \tag{IV.85}$$

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

L'application de cette méthode peut donc, être résumée par les étapes suivantes:
Après initialisation des poids $w_{pq}(0)$ à des valeurs aléatoires et les dérivées $p_{pq}^j(0)$ à *0*.
A chaque étape entre $t = t_0$ et $t = t_1$:

- Calculer $p_{pq}^k(t)$ en utilisant les équations (IV.77) et (IV.85).
- Calculer la réadaptation des poids:

$$\Delta w_{pq}(t) = \alpha \sum_{k \in \Omega} E_k(t) p_{pq}^k(t) \tag{IV.86}$$

La correction totale à appliquer aux poids, à la fin de la trajectoire se calcule par la somme des termes de réadaptation (équation IV.86) calculées à chaque instant:

$$\Delta w_{pq} = \sum_{t=t_0+1}^{t_1} \Delta w_{pq}(t) \tag{IV.87}$$

Cet algorithme garde les poids constants pendant tout l'intervalle. Williams et Zipser ont proposé une autre version, appelée RTRL (*Real Time Recurrent Learning)*, où la

réadaptation des poids est effectué 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 les sorties du réseaux dans les équations (IV.76) et (IV.85) par les sorties désirées. cette méthode est appelé Teached-Forced Real Time Recurrent Learning.
Nous présentons dans l'annexe D. l'algorithme complet du Teached-Forced RTRL.

IV.5.2 Réseaux de neurones à une couche cachée dynamique

Ce modèle consiste en un réseau à une couche cachée qui est la seule à être récurrente; les couches d'entrée et de sortie étant statiques.
Il a été établi que ce type de réseaux est plus efficace que les réseaux entièrement interconnectés. En effet, les performances du réseau de 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 les feed-back ne concernent qu' un groupe limité de neurones. En identification, par exemple, L.Ljung a mentionner que le choix d'un modèle ayant un minimum de paramètres, pouvant identifier le système, est toujours mieux [Cha95]. Un nombre réduit de poids pour ce cas ne peut donc être que préférable.

IV.5.2.1. Réseaux à une couche cachée entièrement interconnectée

Ce type de réseaux contient une seule couche cachée dynamique, qui envoie ses sorties vers la couche de sorties dont les neurones sont linéaires (figure IV.10).
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.

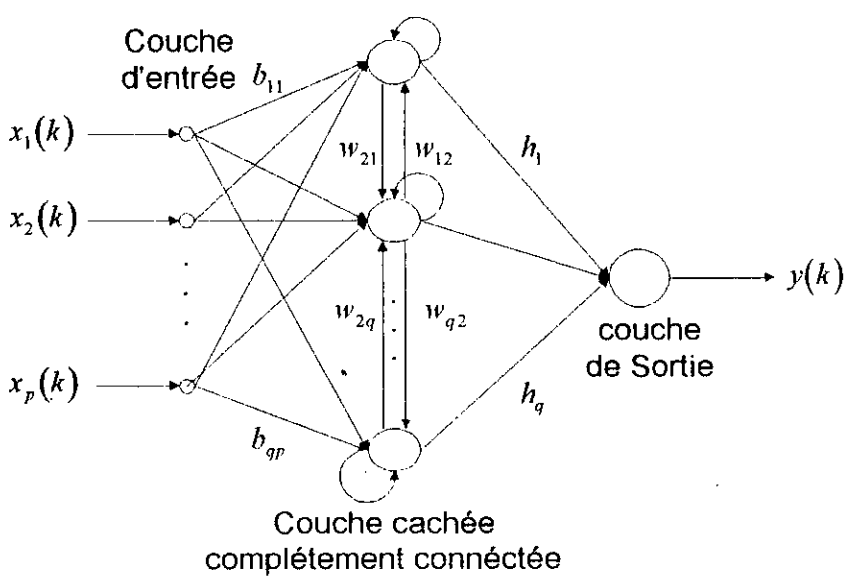


Fig. IV.10 Architecture d'un réseau de neurones à une couche cachée dynamique

Le réseau est régit par l'équation suivante:

$$U = -U + Wf(U) + BX(k) \tag{IV.88}$$

$$y(k) = h^T U^* \tag{IV.89}$$

où $U \in \Re^q$ et $f: \Re^q \to \Re^q$ est un vecteur composé de fonctions sigmoïde. $W \in \Re^{q \times q}$, $B \in \Re^{q \times q}$ et $h \in \Re^q$ représentent les matrices des poids de connections entre neurones.
U^* Dans (IV.71) représente le vecteur des états d'équilibre stable de l'équation (IV.88). pour les entrées $X(k)$ à l'instant k. Le réseau décrit ci-dessus est à une seule sortie dans sa dérniére couche.

A l'équilibre, les équations (IV.88) et (IV.89) donnent :

$$u_i^* = \sum_{j=1}^{q} w_{ij} f_j(u_j^*) + \sum_{l=1}^{p} b_{il} x_l(k) \tag{IV.90}$$

$$y(k) = \sum h_i u_i^* \tag{IV.91}$$

où 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 (IV.91) et (IV.89) indiquent que la sortie du réseau est linéaire.

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

$$E(k) = \left[d(k) - y(k)\right]^2 = \left[d(k) - h^T U^*\right]^2 \tag{IV.92}$$

La méthode de Backpropagation est appliquée:

$$\begin{aligned} h_i(k+1) &= h_i(k) - \eta_1 \frac{\partial E(K)}{\partial h_i(k)} \\ w_{ij}(k+1) &= w_{ij}(k) - \eta_2 \frac{\partial E(K)}{\partial w_{ij}(k)} \\ b_{ij}(k+1) &= b_{ij}(k) - \eta_3 \frac{\partial E(K)}{\partial b_{ij}(k)} \end{aligned} \tag{IV.93}$$

Des équations (IV.92) et (IV.93), on peut aisément déduire:

$$\begin{aligned} h_i(k+1) &= h_i(k) + \eta_1 \left[d(k) - y(k)\right] u_i^* \\ &\qquad i = 1,\ldots,q \\ w_{ij}(k+1) &= w_{ij}(k) + \eta_2 h_i(k) \left[d(k) - y(k)\right] f_j(u_j^*) \\ &\qquad i,j = 1,\ldots,q \\ b_{ij}(k+1) &= b_{ij}(k) + \eta_3 h_i(k) \left[d(k) - y(k)\right] x_j(k) \\ &\qquad i = 1,\ldots,q \text{ et } j = 1,\ldots,p \end{aligned} \tag{IV.94}$$

La stabilité de ce réseau dépend beaucoup des poids initiaux et du choix des taux d'apprentissage. Ces derniers doivent être, de préférence, décroissants en fonctions du temps.

De plus, certaines conditions sur les paramètres des fonctions d'activation, doivent être vérifiées pour assurer la convergence. Des fonctions qui se saturent rapidement devraient permettre une convergence rapide [Kark93].

IV.5.2.2 Réseaux Diagonalement Récurrents DRNN

La différence de ce réseau par rapport au réseau précèdent, est que sa couche cachée ne comporte de feed-back qu'entre chaque neurone et lui même (figure IV.11). 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 réseaux est minimisé[Cha95].

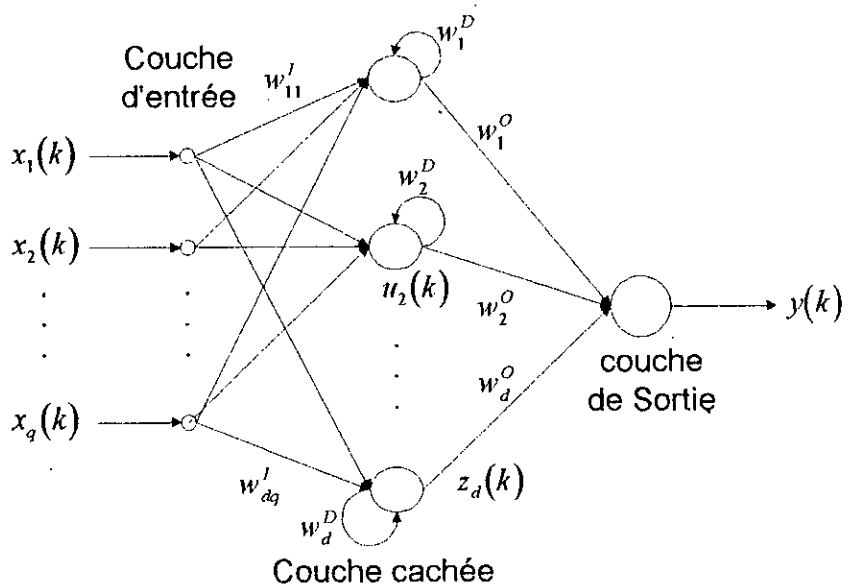


Fig. IV.11 architecture d'un réseau récurrent diagonal (DRNN).

Pour un modèle discret, le fonctionnement du réseau est décrit par (fig. IV.11):

$$y(k) = \sum_{j=1}^{d} w_j^O z_j(k) \tag{IV.95}$$

$$z_j(k) = f\left(u_j(k)\right) \tag{IV.96}$$

$$u_i(k) = w_i^D z_i(k-1) + \sum_{j=1}^{q} w_{ij}^I x_j(k) \tag{IV.97}$$

Où pour chaque instant k, $x_i(k)$ représente la i éme entrée, $z_j(k)$ est la sortie du j éme neurone récurrent $u_j(k)$ est la somme pondérée des entrées du j éme neurone récurrent et $O(k)$ est la sortie du DRNN. La fonction $f(.)$ dans l'équation (IV.96) est la fonction sigmoïde et les matrices $W^I \in \Re^q$ $w^D \in \Re^d$ et $W^O \in \Re^{d \times p}$ dans les équations (IV.96) et (IV 98) 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 celle de 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:

$$E(k) = \frac{1}{2}\left[d(k) - y(k)\right]^2 = \frac{1}{2}\left[d(k) - W^O z(k)\right]^2 \qquad \text{(IV.98)}$$

En appliquant la Backpropagation, on a:

$$W(k+1) = W(k) - \eta \frac{\partial E(k)}{\partial W(k)} \qquad \text{(IV.99)}$$

A partir des équations (IV.95), (IV.96), (IV.97) et (IV.98), on obtient:

$$\begin{aligned} w_i^O(k+1) &= w_j^O(k) + \eta_1\left[d(k) - y(k)\right] z_i(k) \\ w_i^D(k+1) &= w_i^D(k) + \eta_2\left[d(k) - y(k)\right] W_i^O P_i(k) \\ w_{ij}^I(k+1) &= w_i^I(k) + \eta_3\left[d(k) - y(k)\right] W_i^O Q_{ij}(k) \end{aligned} \qquad \text{(IV.100)}$$

Avec :

$$\begin{aligned} P_i(k) &= \frac{\partial z_i(k)}{\partial w_i^D} \\ Q_{ij}(k) &= \frac{\partial z_i(k)}{\partial w_{ij}^I} \end{aligned} \qquad \text{(IV.101)}$$

A partir de (IV.96) et (IV.97), on peut déterminer les expressions de (IV.101) à chaque instant par un calcul itératif:

$$\begin{aligned} P_i(k) &= f'(u_i)\left(z_i(k-1) + w_i^D P_i(k-1)\right) \\ Q_{ij}(k) &= f'(u_i)\left(x_j(k) + w_i^D Q_{ij}(k-1)\right) \end{aligned} \qquad \text{(IV.102)}$$

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

Cette méthode d'apprentissage garantit la convergence, sous condition 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 probleme d'identification, pour lequel il a utilisé ce réseau, que les valeurs optimales de ces parametres peuvent être calculées par la formule ci-dessous:

$$\eta^* = \frac{1}{g^2(k)_{\max}} \qquad \text{(IV.103)}$$

IV.5.3 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:

$$y(k+1) = f(y(k), u(k)) \qquad \text{(IV.104)}$$

Pour tout système de la forme (IV.104), 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 (IV.85) avec la précision ε sur l'intervalle *[0;T] et* sur les ensembles U et X. [Riv95].

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 intéressante,s 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 rend le 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 approximée avec une structure très simple, à base de neurones linéaire. L'apprentissage des *regression Networks* 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].

Conclusion

Dans ce chapitre, nous avons effectué une étude détaillée sur les réseaux, dont l'entraînement est basé sur les méthodes d'apprentissage supervisé.

Les réseaux LBF constituent un modèle très efficace, qui a à son actif la majorité des applications des réseaux de neurones, notamment en identification et commande de processus.

Pour l'entraînement de ces réseaux, la méthode de Backpropagation constitue l'algorithme d'apprentissage, qui reste le plus utilisé. Cet algorithme souffre, néanmoins de certaines limitations. La vitesse de convergence est lente et dépend étroitement de la base d'apprentissage. De plus, cet algorithme se bloque souvent dans des minimums locaux.

Les méthodes basées sur le Gradient du second ordre constituent des algorithmes d'apprentissage, qui sont beaucoup plus rapides que la Backpropagan grâce au choix judicieux de la direction de recherche. Ces algorithmes permettent d'obtenir de bons résultats avec un nombre de neurones moins important. Le plus grand inconvénient, que l'on a relevé au court de notre utilisation de ces méthodes, est les calculs qui sont relativement plus compliqués.

L'algorithme d'optimisation aléatoire, basé sur la méthode développée par Solis et Wets, constitue une alternative très intéressante, notamment dans les applications en automatique. En effet, cet algorithme ne dépend pas du système pour lequel le réseau est utilisé. De plus, sa technique de recherche permet d'atteindre un minimum global.

En outre, pour leur architecture les réseaux LBF souffrent de l'indétermination du nombre optimal d'éléments dans chaque couche, en fonction du problème à traiter, qui reste toujours ouverte. Ceci, malgré les quelques "recettes" qui sont parfois données et que nous avons présenté dans ce chapitre. Mais concernant la dimension du réseau, il est clair qu'un réseau à deux couches cachées constitue un approximateur universel.

Les réseaux RBF se distinguent par leur caractéristique de meilleurs approximateurs. En effet, ces réseaux sont dotés de la caractéristique de calcul et d'adaptation paramétrique locaux. Ceci fait que ces réseaux soient à l'abri de plusieurs problèmes que connaissent les LBF, comme le *sur-apprentissage*, ou la difficulté d'introduction de nouveaux exemples sans toucher à la précision d'apprentissage des exemples précédemment appris.

Les réseaux dynamiques sont très adaptés pour la modélisation de processus dynamiques. L'apprentissage de ces réseaux, qui est basé sur la méthode de Backpropagation, est néanmoins plus complexe.

La recherche de nouvelles techniques neurales plus adaptées pour l'approximation ne s'arrête pas là. Plusieurs nouvelles variantes sont actuellement en stade de recherche, comme les réseaux probabilistiques[Ren 95]. Ces réseaux qui ont connu ces derniers temps un regain d'intérêt, s'inspirent des processus physiques dont l'évolution vers l'équilibre est régie par des lois probabilistiques. Ces réseaux constituent une extension du modèle de *Boltzman* [Fre92]. Pour leur entraînement, la méthode du *recuit simulé* reste la plus développée [Kar93, Pat96]. Ces réseaux à apprentissage supervisé, basé sur la minimisation de fonction d'énergie, commencent à donner des résultats prometteurs en identification [Ren95]. La mise en oeuvre de tels réseaux consomme, néanmoins, des temps de calcul souvent prohibitifs, à cause du grand nombre de probabilités à estimer. De ce fait, ces derniers restent encore peu utilisés; mais ils sont encore en étude dans plusieurs projets de recherche[Hèr 94, Pat96].

CHAPITRE V:

IDENTIFICATION DES SYSTEMES PAR RESEAUX DE NEURONES

Chapitre V

IDENTIFICATION DES SYSTEMES PAR RESEAUX DE NEURONES

Introduction

L'identification de systèmes trouve son importance dans le fait que la conception d'une stratégie de commande est, le plus souvent établie, sur la base d'un modèle.

L'efficacité des réseaux de neurones en identification revient à leurs capacités d'approximation universelle. C'est à partir des théorèmes, que nous avons déjà cités (cf.chap.IV), faisant de ces réseaux un outil efficace pour l'approximation de fonction, et le sucés de ces derniers en ce domaine, que leur utilisation a été étendue à la modélisation de systèmes. En effet dés la fin de la dernière décennie et le début des années *90* ,La modélisation de systèmes a connu l'introduction des réseaux de neurones [Nar 90].

Si l'identification de systèmes linéaire a été largement étudiée et ne pose plus de problèmes insurmontables, il n'en est pas de même pour les systèmes non-linéaires. Ces derniers sont souvent de formes complexes et les informations disponibles s'y rapportant ne sont pas toujours suffisantes pour une bonne approximation. C'est dans ces cas que l'utilisation des réseaux de neurones est d'un apport très intéressant.

L'efficacité, des réseaux de neurones statiques multicouches LBF entraînés par la méthode de Backpropagation en identification a été prouvée tout au long des dernières années[Nar90, Asr94]. En effet plusieurs travaux ont été effectués où ces réseaux ont montré leurs aptitudes dans l'établissement de modèles Entrée-sortie, pour les systèmes non-linéaires. Ceci, avec tous les avantages que ces méthodes neurales peuvent présenter, par rapport aux méthodes classiques utilisées [Bur93, Chu90, Nav90, Bil92].

Dans ce chapitre, hormis ce type de réseaux, nous nous intéresserons aux autres différents modèles de réseaux de neurones, que nous avons développés dans le chapitre précédent (LBF, RBF, Dynamique, ...). Nous nous intéresserons aussi, à leurs méthodes d'apprentissage, en vue d'examiner les possibilités qu'ils peuvent offrir en identification et les avantages que peuvent présenter certaines par rapport à d'autres.

Ce chapitre se compose principalement de trois parties.

Dans la première partie, Nous présenterons les principales structures qui existent pour l'identification des systèmes par réseaux de neurones. Les problèmes que certaines peuvent poser ainsi que les avantages qu'elles peuvent apporter serons examinés.

Dans la seconde partie, nous utiliserons les autres modèles de réseaux que nous avons développés dans notre travail, tels que les réseaux RBF et les réseaux dynamiques, pour les appliquer en identification. Nous nous intéresserons d'autre part aux différentes méthodes d'apprentissage, y compris celles relatives aux réseaux LBF. Ceci, afin de comparer et tirer les conclusions sur les réseaux ainsi que les algorithmes pour leur entraînement, qui présentent de meilleures alternatives pour l'identification de systèmes. Pour cela, plusieurs modèles de systèmes sont utilisés.

A la fin, et à la lumière de tous les résultats obtenus dans ces premières applications, nous nous proposerons de faire l'identification d'un réacteur chimique en vu d'éffectuer sa commander dans le chapitre suivant.

V.1 Etude de l'Identification des systèmes par réseaux de neurones

Les réseaux de neurones, comme nous l'avons montré précédemment, offrent des avantages très intéressants lorsqu'ils sont utilisés pour l'approximation de fonctions sur la base d'exemples. Il est donc, naturel de penser à les utiliser en identification de processus.

Cependant, comme dans le cas d'identification classique, la question d'identifiabilité reste toujours ouverte en identification par réseaux de neurones. En effet démarrant d'une certaine structure, rien ne garantit qu'un système peut être identifié par cette structure ou une autre. Ce problème n'ayant pas été résolu théoriquement, on suppose toujours que le système traité est identifiable par la structure choisie [Hun92].
Dans le domaine neural, les seuls résultats qui peuvent nous guider, dans l'identification de processus, sont ceux relatifs à l'approximation des fonctions, que nous avons largement exposés dans le chapitre précédent. Ainsi, rien ne peut nous orienter sur la structure neurale adéquate pour tel ou tel problème, si ce n'est les théorèmes d'approximation par réseaux de neurones, dont le contenu est encore général.

Nous supposerons que les systèmes à identifier sont de forme générale:

$$y(k) = f\left(y(k-1), y(k-2), \ldots, y(k-n),\ u(k-1), u(k-2), \ldots, u(k-m)\right) \qquad (V.1)$$

Où $f(.)$ est une fonction non-linéaire inconnue.

Selon l'objectif à atteindre, le processus d'identification par réseaux de neurones peut se faire de deux manières différentes:

1. Identification en Ligne

Dans ce cas, le système est identifié en temps réel pendant le déroulement de la commande du processus. Ce type d'identification est nécessaire si l'on utilise une stratégie de commande adaptative. Le réseau doit faire son apprentissage dans la boucle de commande. Ce type d'identification peut poser certains problèmes. Les données d'apprentissage qui ne sont disponibles que graduellement, résultent, en plus de la dynamique du système, surtout du régulateur et de l'objectif de la commande; ce qui peut compromettre les conditions de l'identifiabilité, en particulier la richesse suffisante de l'entrée d'excitation du processus. C'est le problème d'excitation persistante où une erreur de prédiction tendant vers zéro ne garantit pas une convergence optimale des paramètres [Riv95].

2. Identification hors-ligne

Dans ce cas, le réseau est placé en parallèle avec le système. Il est excité par les mêmes entrées que le système; et son entraînement consiste à minimiser l'erreur entre la sortie du système et celle fournie par le réseau (figure V.1). Dans cette approche, comme en identification classique, le choix des entrées joue un rôle important. Les entrées utilisées pendant l'entraînement doivent être représentatives de toutes les classes de signaux qui viendront exciter le système pendant son utilisation. Ces signaux doivent, d'autre part, être suffisamment riches en fréquence afin de bien

exciter la dynamique du système. Des séquences aléatoires uniformément distribuées peuvent être utilisées pour y arriver [Bil92].

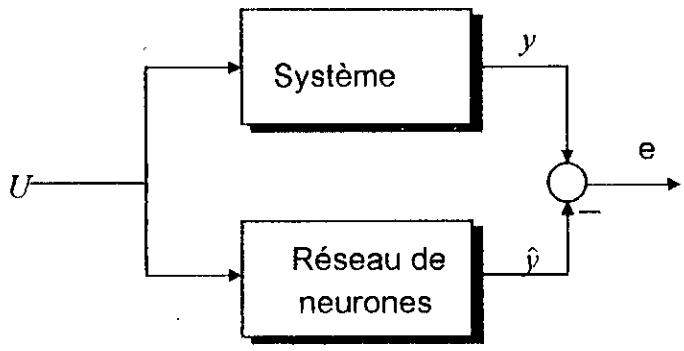


Fig. V.1 Schéma de principe de l'identification des systèmes par réseaux de neurones.

V.1.2. Structure d'identification par réseaux de neurones

Les structures suivant lesquelles le réseau est entraîné à approximer le système peuvent être de deux formes différentes.

V.1.2.1. Structure à apprentissage Série-Parallèle

Dans ce cas, les sorties réelles du système sont utilisées pour constituer les entrées récurrentes du réseau en fonction desquelles celui ci fait la prédiction de ses sorties futures. L'utilisation de cette stratégie est préférable du point de vue de la stabilité du processus et de la simplicité de l'algorithme (figV.2).

La fonction approximée par le réseau dans ce cas est:

$$\hat{y}(k) = f\big(y(k-1), y(k-2), \ldots, y(k-n),\ u(k-1), u(k-2), \ldots, u(k-m)\big) \qquad \text{(V.2)}$$

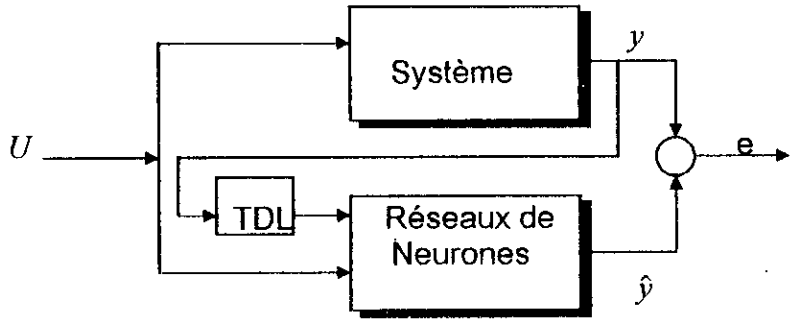


Fig.V.2 Structure d'identification neurale Série-Parallèle

V.1.2.2. Structure à apprentissage Parallèle

Dans cette structure, on utilise les prédictions du réseau comme ses entrées récurrentes à partir desquelles, il déterminera ses sorties futures(figure V.3)
Ce retour de la sortie, qui fait que le réseau devienne récurrent à son entrée, dépend de l'ordre du système à modéliser sur lequel des connaissances à priori doivent donc être déjà disponibles.

La fonction dont le réseau doit faire l' approximation dans ce cas, est de la forme:

$$\hat{y}(k) = f\left(\hat{y}(k-1), \hat{y}(k-2), \ldots, \hat{y}(k-n), \text{u(k-1)}, u(k-2), \ldots, u(k-m)\right) \qquad \text{(V.3)}$$

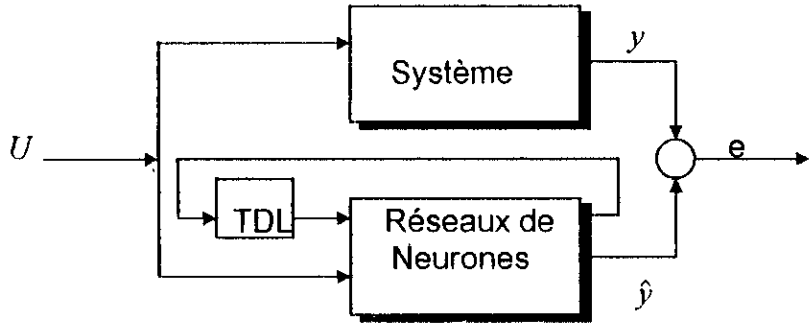


Fig.V.3 Structure d'identification neurale Parallèle.

Cette structure pose généralement des problèmes de stabilité et n'est utilisée qu'une fois que le réseau a reçu suffisamment d'entraînement en série-Parallèle et a convergé.
Par ailleurs lors de l'utilisation du réseau dans la boucle de commande, c'est sous cette structure que celui ci est mis.

V.1.3. Identification des systèmes dynamiques par les réseaux récurrents

Dans la structure d'identification Parallèle, c'est la dynamique du système qui doit être modélisée à travers les bouclages. Ces feed-backs rendent le réseau récurrent en entrée. Dans ce contexte, R.J.Williams propose d'utiliser les réseaux complètement récurrents, dont nous avons déjà étudié les architectures et les algorithme d'apprentissage dans le chapitre IV. Ces réseaux sont en effet dotés déjà d'une architecture interne qui leur permet de modéliser les systèmes dynamiques. Les entrées de ces réseaux peuvent se limiter uniquement au signal de commande actuel $u(k)$ et à la sortie $\hat{y}(k)$ générée par l'entrée précédente[Ren95]. Avec cette structure, il n'est pas nécessaire de connaître l'ordre du système à identifier, le réseau est libre à travers sa représentation interne récurrente, à s'adapter au problème pour modéliser son comportement dynamique.

Cependant les théorèmes de capacité d'approximation des réseaux de neurones statiques ne peuvent pas être directement étendus aux réseaux dynamiques.
Nous avons vu dans le chapitre IV que la propriété d'approximateur universelle des réseaux de neurones prend un sens différent s'il s'agit d'un réseau dynamique
A travers le théorème d'approximations de fonctions dynamiques par les réseaux récurrents que nous avons présentés, il ressort que la modélisation d'un système dynamique par un réseau récurrent n'est pas globale, mais restreinte à un domaine des espaces d'état et d'entrée, sur un intervalle de temps fini. Une identification faite sur la base de cette approche peut donc ne pas refléter des caractéristiques fondamentales du processus qu'elle est censée approcher, sa stabilité par exemple. Quant à la question d'identifiabilité, elle reste toujours ouverte[Bur93].

En outre, S.Qin [Qin92] a observé, dans un travail de comparaison sur les aptitudes des ces réseaux de neurones en identification de systèmes dynamiques, que les réseaux récurrents sont plus robustes au bruit que les réseaux statiques. Mais dans d'autres travaux c'est l'inverse qui est observé [Ren95]; ceci ne permet donc pas de conclure dans un sens ou dans un autre.

V.2. Applications sur l'Identification des systèmes par différents réseaux de neurones

Dans cette partie, nous utiliserons, en plus des réseaux LBF, les autres modèles de réseaux de neurones déjà étudiés dans notre travail tels que les réseaux RBF et les réseaux dynamiques. Ceci afin de tirer les conclusions sur les réseaux et les méthodes d'apprentissage qui présentent de meilleures alternatives pour l'identification de systèmes.

Nous avons choisi pour cette étude, quatre modèles qui ont été sélectionnés sur la base de leurs formes et complexités, qui sont différentes.

Modèle 1

Dans cette partie le modèle choisi est décrit par l'équation aux différences suivante:

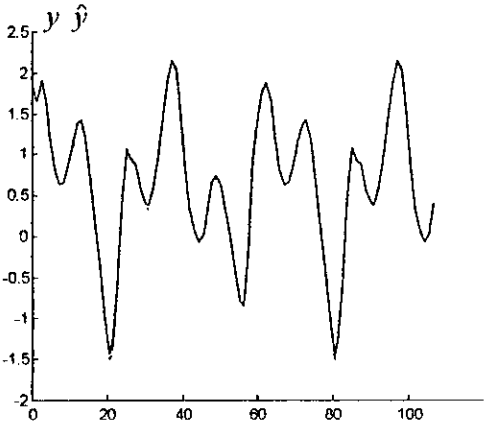
$$y(k+1) = \frac{-0.9y(k) + u(k)}{1 + y(k)^2} \qquad (V.4)$$

Pour l'identification de ce système, deux réseaux **LBF** à deux entrées sont utilisés et entraînés séparément par la technique série-parallèle. Le premier est entraîné par la méthode de Backpropagation avec Momentum et taux d'apprentissage adaptatif. Pour le deuxième, la méthode du Gradient de second ordre de leavenberg Marquard est utilisée. Une séquence d'entrées aléatoires dont les valeurs appartiennent à l'intervalle [-2,2] est utilisée pour l'entraînement des deux réseaux.
Pour le premier réseau, deux couches cachées respectivement de 10 et 12 neurones ont été nécessaires. Pour son entraînement, Le taux d'apprentissage initial est de 0.1, et le paramètre du momentum utilisé est égal à 0.9.
La convergence vers une erreur globale de 0.0001 est obtenue au bout de 14000 itérations. La figure (V.4.a) montre les résultats obtenus après entraînement de ce réseau. L'entrée avec laquelle le système est excité est définie par : $u(k) = \sin\left(\frac{2\pi k}{10}\right) + \sin\left(\frac{2\pi k}{25}\right) + \sin\left(\frac{2\pi k}{5}\right)$

Le réseau entraîné par la méthode du Gradient de second ordre a pu donner le même résultat mais avec une seul couche cachée à 15 neurones. Le nombre d'itérations d'entraînement est pratiquement égal à celui du premier réseau. La figure (V.4.b) montre la réponse du réseau pour le même signal d'excitation appliqué au premier. On remarque que la réponse du réseau et celle du système sont identiques.

Nous remarquons, ainsi, que la méthode du Gradient de second ordre nous a permis d'obtenir rapidement la solution qui existe pour l'approximation de cette fonction, à savoir un réseau **LBF** avec une seule couche cachée. Ce que nous n'avons pas pu obtenir au début avec la méthode de la Backpropagation de premier ordre.

a.



b.

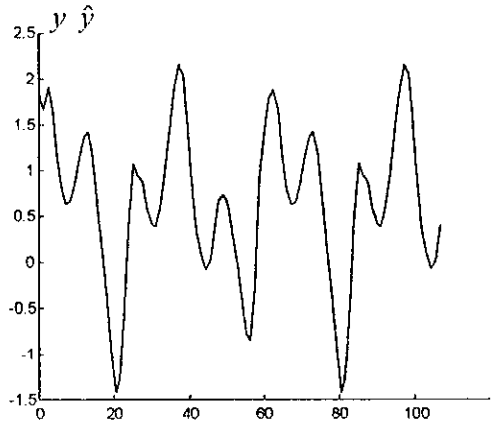
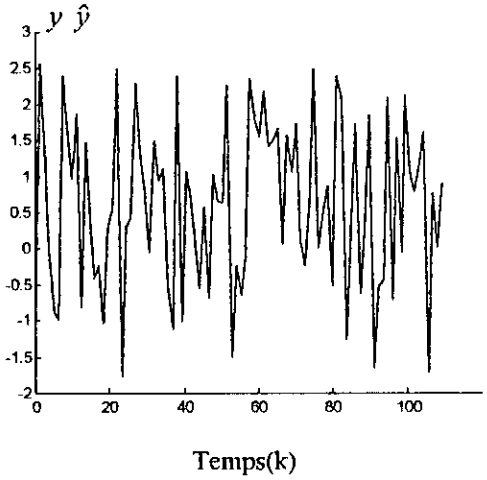


Fig. V.4.a. Sortie du système et du modèle neural pour une séquence d'entrées *U(k)*.
a. Réseau **LBF** entraîné par la Backpropagation avec momentum et taux d'apprentissage adaptatif.
b. Réseau du type **LBF** .entraîné par la Méthode de Leavenberg Marquard.
····· Sortie du système.
——Sortie du modèle neural.

Le réseau entraîné par la méthode de Leavenberg Marquard, est moins chargé en nombre de neurones, celui ci peut donc présenter de problèmes de précision (cf Chap.IV.4.2.3). Afin de mieux comparer les capacités des deux réseaux, nous les avons excités avec une séquence d'entrées aléatoires riche en fréquences. Nous remarquons sur la figure (V.5) que les performances du réseau à une seule couche cachée sont aussi bonnes que le réseau à deux couches cachées. Le nombre de neurones entraîné avec la méthode de Levenberg Marquard est donc suffisant pour construire un modèle pour le système.

a.



b.

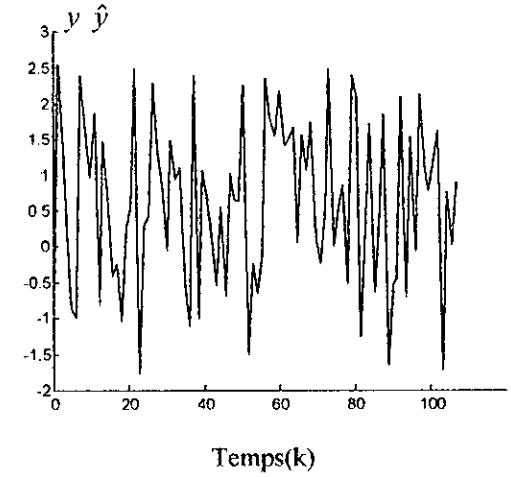


Fig. V.4.b. Sortie du système et du modèle pour deux séquences d'entrée aléatoire riche en harmoniques
a. Réseau **LBF** entraîné par la Backpropagation avec momentum et taux d'apprentissage adaptatif.
b. Réseau du type **LBF** .entraîné par la Méthode de leavenberg Marquard.
····· Sortie du système
—— Sortie du modèle neural

Ainsi, la méthode du gradient de second ordre est arrivée à modéliser le système avec une structure neural beaucoup moins chargée que celle du réseau entraîné par la Backpropagation avec momentum et taux d'apprentissage adaptatif. Ceci montre la puissance des méthodes du second ordre et leur rapidité de convergence. Il faut, cependant noter que cette méthode de Leavenberg Marquard est beaucoup plus complexe en implémentation que la méthode de Backpropagation.

Modèle 2

Dans ce cas le système est décrit par le modèle suivant:

$$y(k+1) = \frac{y(k)}{1+y(k)^2} + u^3(k) \qquad (V.5)$$

Pour l'identifier, nous avons utilisé une structure d'apprentissage Série parallèle. Le premier réseau utilisé est un RBF à 2 entrées et des neurones cachés, dont les sorties sont Gaussiennes. L'entraînement est effectué avec des entrées aléatoires dans l'intervalle [-2.5,2.5]. Le paramètre de dispersion σ a été fixé pour tous les neurones pendant l'entraînement à une valeur de 0.5. Après 92 itérations d'apprentissage hiérarchique le réseau a convergé avec une erreur d'environ 10^{-5} en générant 91 neurones gaussiens.

Un second réseau différent du premier a été utilisé pour la modélisation de ce système; c'est un réseau du type **LBF**. Celui ci a été entraîné par la méthode du Gradient du second ordre. Le réseau est à 2 entrées et une couche cachée de 15 neurones. L'Entraînement a été effectué avec la même séquence aléatoire. L'adaptation a nécessité 15000 itérations pour une erreur d'entraînement sur tous les exemples inférieur à 10^{-4}.

Pour les résultats, nous avons testé les deux réseaux avec une trajectoire aléatoire contenant plusieurs harmoniques. Nous remarquons sur la figure (V.5), que les sorties des réseaux et celles du système sont pratiquement indiscernables. Il en ressort que le **RBF Gaussien** est arrivé à faire une très bonne approximation du système avec plus de neurones mais un entraînement plus court. Ceci montre l'avantage que nous offrent les réseaux RBF en identification à travers, notamment, leur apprentissage qui peut se faire plus rapidement.

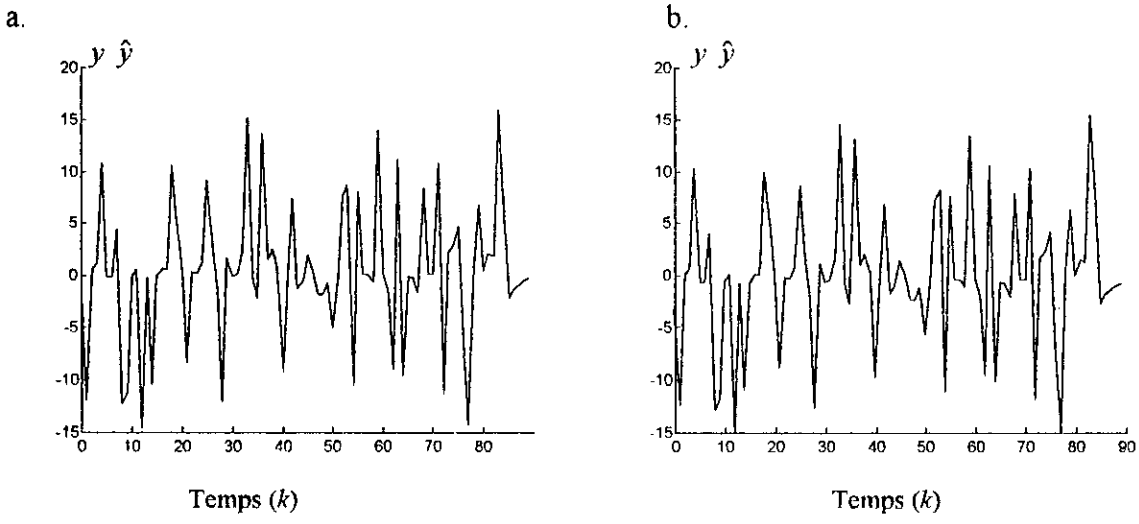


Fig. V.5 Sortie du système et du modèle. a. Réseau **RBF**
b. Réseau **LBF** entraîné par la méthode du gradient du second ordre.
····· Sortie du système
—— Sortie du modèle neural

Modèle 3

Dans ce cas, nous allons effectuer l'identification d'un système dont le modèle est de la forme suivante:

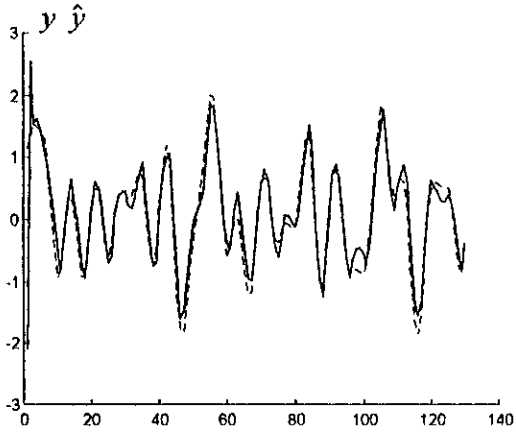
$$y(k+1) = \frac{1}{5}\left(\frac{y(k)y(k+1)\left(y(k+1)+2.5\right)}{1+y^2(k)+y^2(k+1)} + u(k)\right) \qquad (V.6)$$

Pour celà, deux réseaux de neurones de types différents sont entraînés séparément, avec en entrées des séquences aléatoires dans l'intervalle [-1.5,1.5].
Le premier, est un réseau **RBF** à 3 entrées avec une couche cachée Gaussienne. Le paramètre de dispersion a été fixé pour tous les neurones pendant l'entraînement à une valeur de 0.7. Son entraînement, qui s'est effectué sous la forme série-parallèle a nécessité 97 itérations pour obtenir 96 neurones cachés et aboutir à une erreur sur tous les exemples de 0.00009.

Le second réseau est un **réseau dynamique** à couche cachée entièrement interconnectée à 3 neurones dynamiques. Les taux d'apprentissages utilisés sont adaptatifs decroissants. les valeurs initiale sont de 0.01 pour la couche d'entrée et celle de sortie, et 0.1 pour les neurones dynamiques. Après 25 itérations d'entraînement en série parallèle, le réseau a commencé a convergé mais l'erreur évoluait très peu. La valeur de la somme d'erreur sur tous les exemples était de 8.8. La figure (V.6) montre la réponse du réseau et celle du modèle pour une séquence d'entrées *U(k)*, ainsi que l'erreur sur toute la trajectoire aprés cet apprentissage en série-paralléle tel que:

$$u(k) = \sin(2\pi/10) + \sin(2\pi/25) + \sin(2\pi/13) + \cos(2\pi/7) + \sin(2\pi/17)$$

a.



b.

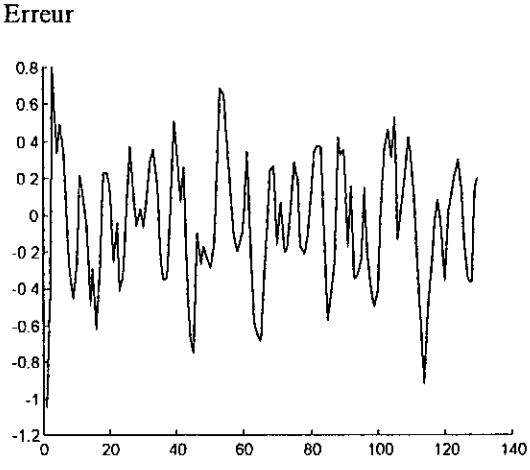


Fig. V.6 Identification du système par un réseau dynamique en série-paralléle.
a.Sortie du système et du modèle à base de réseau dynamique
pour une séquence d'entrée *u(k)*.
····· Sortie du système
—— Sortie du modèle neural.
b. L'erreur sur toute la trajectoire.

Le réseau ayant entamé sa convergence, nous avons arrêté l'entraînement et rebouclé sa structure en réinjectant sa sortie vers son l'entrée. Le risque d'instabilité de l'apprentissage étant écarté, l'entraînement est relancé sous la structure d'identification parallèle (cf.V1.2.2). L'erreur a augmenté au début, puis a commencé à diminuer. L'apprentissage a duré pendant 800 itérations. A la convergence l'erreur a atteint 0.0001 sur toute la séquence d'entraînement.

La figure (V.7) montre la réponse du réseau dynamique, ainsi que celle du réseau RBF, pour une trajectoire aléatoire qui leur a été injectée en entrée. Nous constatons que les deux réseaux ont réussi à bien faire l'approximation de ce système dynamique.

A travers notre utilisation du réseau dynamique, nous avons réussi à obtenir un modèle du système (Fig.V.7), qui est basée sur la représentation dynamique interne du réseau récurrent. Ce dernier n'a nécessité qu'un nombre réduit de neurones cachés qui sont dynamiques.

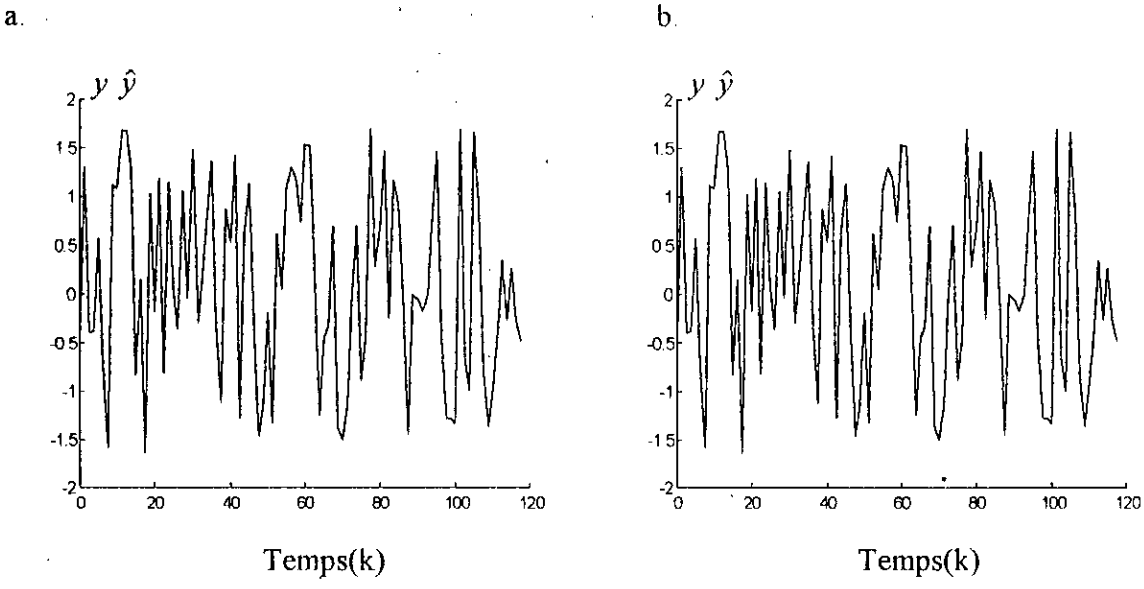


Fig. V.7 Sortie du système et du modèle.
a. Réseau **RBF.** b. Réseau dynamique .
····· Sortie du système
—— Sortie du modèle neural.

Nous avons d'autres part, relevé, dans cette opération d'identification, la nécessité de commencer toujours l'entraînement en structure série-paralléle. Pour passer à une structure parallèle, nous avons été obligés d'attendre jusqu'à ce que le réseau dynamique commence à converger. Dans le cas ou le réseau n'a pas suffisamment converger, l'entraînement du réseau sous la structure parallèle divérge.

Modèle 4

Dans cette partie, nous allons s'intéresser à l'identification d'un système multivariables. Le système choisi est du type MIMO et est décrit par le modèle discret suivant:

$$\begin{bmatrix} y_1(k+1) \\ y_2(k+1) \end{bmatrix} = \begin{bmatrix} \dfrac{y_1(k)}{1+y_1(k)^2} \\ \dfrac{y_1(k)y_2(k)}{1+y_1(k)y_2(k)} \end{bmatrix} + \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \qquad (V.7)$$

Pour son identification, nous avons utilisé un **RBF Gaussien,** entraîné par la technique série-paralélle avec un apprentissage hiérarchique. Ce réseau est à quatre entrées et avec un paramètre de dispersion σ de valeur de 0.5 pour tous les neurones cachés. L'entraînement est effectué avec des séquences d'entrées aléatoires $u_1(k)$, $u_2(k)$ dans l'intervalle [-3,3]. Le nombre de centres gaussiens générés pendant l'apprentissage est de 138. La figure (V.8) montre la réponse du système et celle du modèle pour des vecteurs d'entrées:

$$u_1(k) = \sin(2\pi k/10) + \sin(2\pi k/25) + \sin(2\pi k/5) + \sin(2\pi k/11) + \sin(2\pi k/5) + \sin(2\pi/7)$$

$$u_2(k) = \sin\left(2\pi k/7\right) + \sin\left(2\pi k/13\right) + \sin\left(2\pi k/11\right) + \sin\left(2\pi k/17\right) + \cos\left(2\pi k/23\right) + \left(\sin\left(2\pi k/11\right)\right)^2$$

a. b.

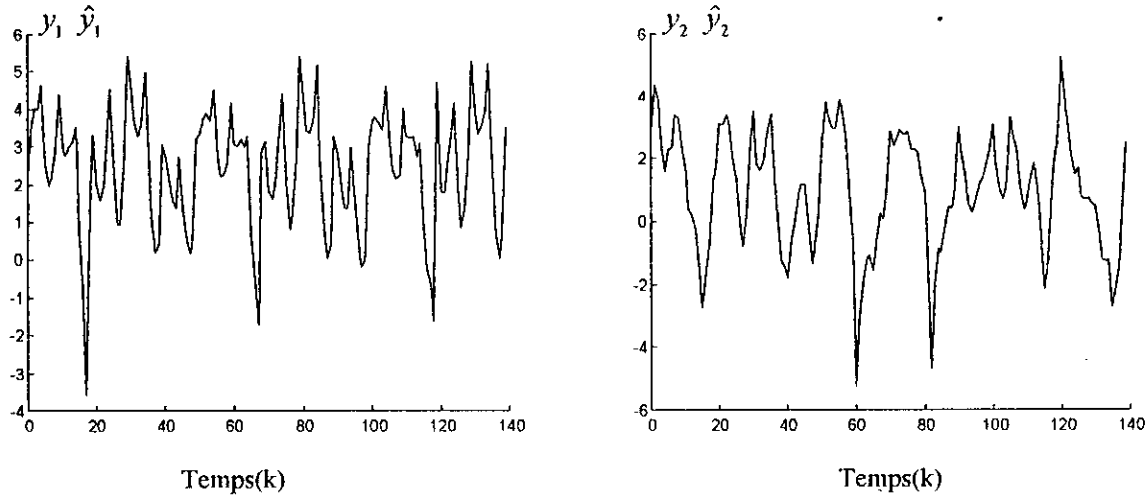


Fig. V.8 Sortie du système et du modèle.
a. Premières sortie y_1 et $\hat{y}_1$; b. Deuxièmes sortie y_2 et $\hat{y}_2$
····· Sortie du système
—— Sortie du modèle neural.

Nous constatons que les sorties du réseau RBF et celles du système sont indiscernables. Ainsi, un seul réseau RBF a été utilisé pour l'identification du système multivariable avec succès. Cette

opération montre la capacité des réseaux RBF Gaussiens en modélisation de systèmes. C'est leur caractéristique de calcul local qui leur permet de telles aptitudes.

V.3 Identification d'un réacteur chimique

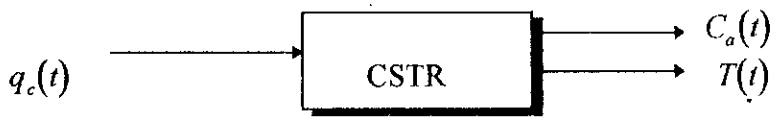
Sur la base des résultats que nous avons obtenus ci dessus, nous nous proposons de faire l'identification d'un réacteur chimique, que nous allons contrôler dans le chapitre suivant. Dans cette partie, nous allons concevoir un modèle neural pour ce réacteur, qui sera utile pour l'établissement du neuro-contrôleur qui le commandera.

Le modèle du système est défini par [Lig95]

$$\dot{C}_a(t) = \frac{q}{v}\left(C_{a0} - C_a(t)\right) - k_0 C_a(t)\exp-\left(E/RT(t)\right) \tag{V.8}$$

$$\dot{T}(t) = \frac{q}{v}\left(T_0 - T(t)\right) + k_1 C_a(t)\exp-\left(E/RT(t)\right) + k_2 q_c(t)\left(1-\exp-\left(k_3/q_c(t)\right)\right)\left(T_{c0} - T(t)\right)$$

La nature de ce réacteur, son étude en boucle ouverte, les informations le concernant ainsi que la nature des coefficients se trouvant dans ce modèle sont détaillées lors de sa commande (cf. VI.2). Quant aux données numériques et les dimensions du réacteur celles ci sont données dans l'annexe B.



Ce système est commandé à l'entrée par un flux $u(t)=q_c(t)$ et à la sortie, nous avons la concentration du produit $C_a(t)$ qui est à la température $T(t)$.

Ce systeme a été échantillonné avec un pas d'échantillonnage $\Delta t=0.01s$ (equ.VI.19). Pour l'identification, nous avons choisi d'utiliser la méthode série parallèle; Ceci afin d'éviter l'instabilité du processus d'entraînement.

Un premier réseau **LBF** est d'abord choisi pour modéliser le système. Ce réseau est à deux couches cachées respectivement de 10 et 12 neurones.Son apprentissage est éffectué à l'aide de la méthode de Levenberg Marquard. Une séquence aléatoire de flux $q_c(t)$ dans intervalle [0 220] est utilisée en entrée. Celle ci est normalisée, afin d'éviter la saturation et le blocage du réseau. Après 6000 itérations, le réseau a convergé vers une erreur de 2×10^{-6}. Nous l'avons testé avec des trajectoires différentes la figure (V.10.a.) Montre la réponse du réseau et celle du modèle. Le réseau de neurones suit le modéle avec une bonne précision.

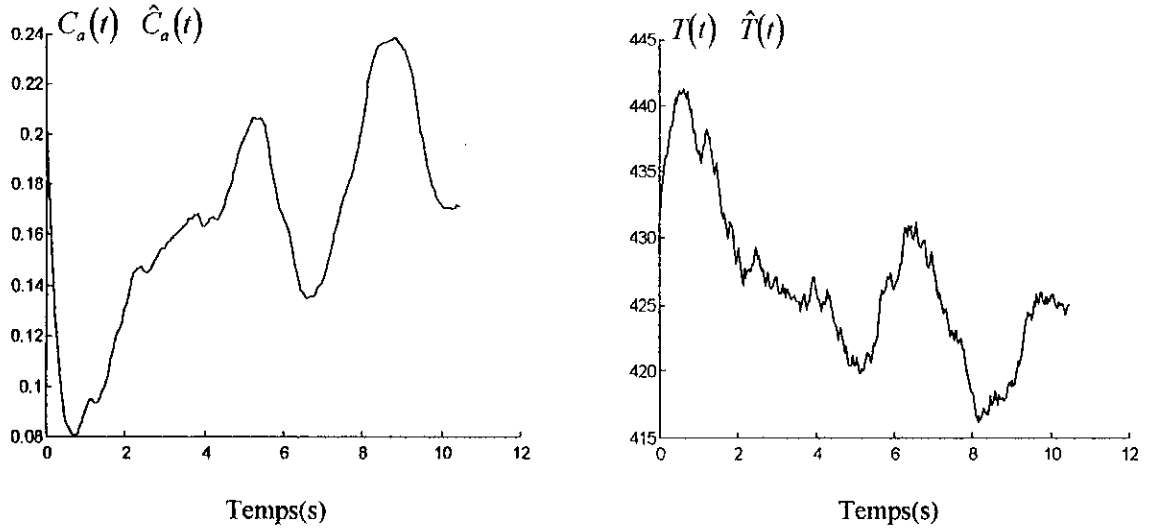


Fig.V.10.a Identification du réacteur dans le premier intervalle de fonctionnement [0,0.35] par le réseau LBF entraîné par la méthode de Levenberg Marquard
——Soties du modèles neural
····· Soties du système.

Un autre réseau est utilisé pour identifier le système. Celui ci est du type **RBF Gaussien**. Pour son entraînement, nous avons choisi un apprentissage hiérarchique avec le coefficient de dispersion σ, qui est constant et égal à 0.5 pour tous les neurones gaussiens du réseau. A la convergence, *130* neurones cachés ont été formés. La figure (V.10.b.) montre la réponse du réseau pour la même trajectoire proposée au réseau LBF, précédemment entraîné.

Nous remarquons que ce réseau nous a permis d'obtenir un modèle précis pour le système.

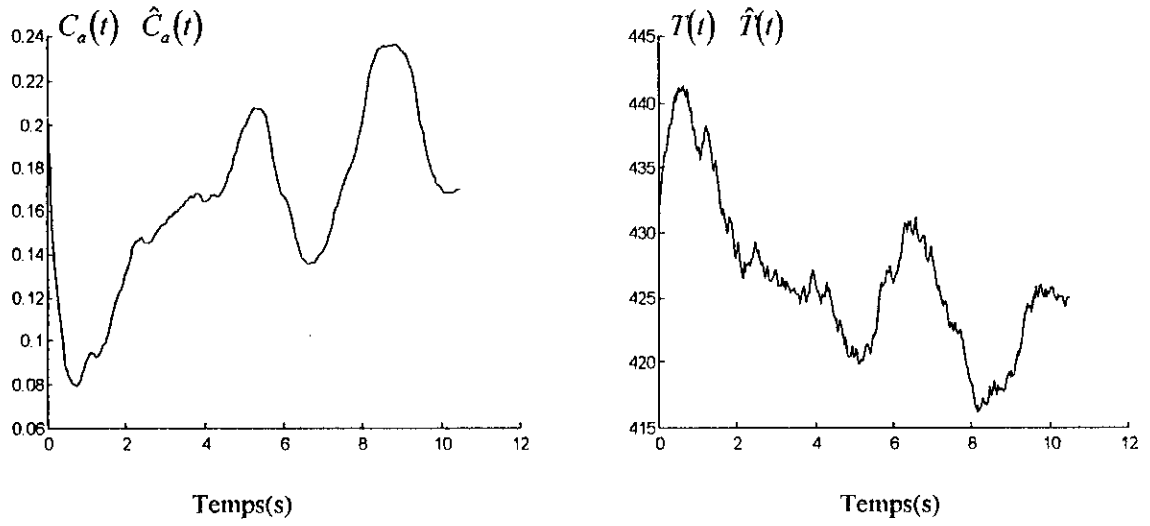


Fig.V.10.b Identification du réacteur dans le premier intervalle de fonctionnement [0,0.35] par le réseau RBF
——Soties du modèle neural
····· Soties du système.

Les opérations d'identification que nous avons effectuées dans cette partie, sont dans un intervalle de la concentration du produit allant jusqu'à *0.35*. Il nous a été, complètement, impossible d'identifier le système dans un intervalle plus grand sans perdre en précicion. En effet toutes les tentatives se sont soldées par un échec quand on a testé les réseaux à la généralisation. C'est le cas pour les deux types de réseaux utilisés, à savoir RBF et LBF. En effet, il est impossible de joindre précision d'entraînement et capacité de généralisation sur tout l'intervalle de fonctionnement du système qui peut aller jusqu'à $C_a(t) = 1$. En plus, le fait que les entrées soient récurrentes a accentuer, encore le problème.

Ainsi, nous avons opté à utiliser deux autres réseaux différents qui seront entraînés à identifier le système dans des intervalles différents. Le premier de *0.3* à *0.65* et le second de *0.6* à *0.95*.

Pour cette opération, nous avons choisi les réseaux **RBF Gaussiens,** afin de bénéficier de leur apprentissage local. Le premier réseau est destiné à identifier le système dans l'intervalle de concentration *0.3 à 0.65*. Nous l'avons entraîné avec la méthode d'apprentissage hiérarchique. Le paramètre de dispersion σ est fixé à *0.5* pour tous les neurones cachés; Il sera maintenu à cette valeur durant l'entraînement. A la fin de l'apprentissage, le nombre de neurones gaussiens générés est de 203 neurones cachés; l'erreur sur toutes les entrées à la convergence est de 0.00004.

La figure (V.11) montre la réponse du réseau pour une séquence d'entrée aléatoire. Nous constatons que ce réseau à 203 neurones gaussien est arrivé à faire une modélisation du système avec une bonne précision.

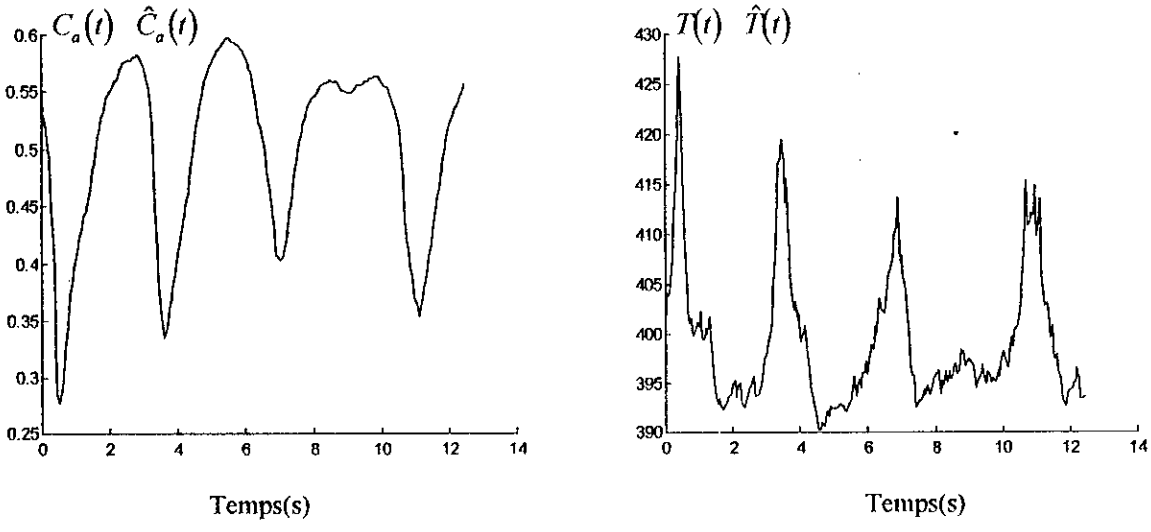


Fig.V.11 Identification du réacteur dans lesecond intervalle de fonctionnement [0.3,0.65] par le réseau RBF
—— Soties du modèle neural
····· Soties du système.

Le troisième réseau RBF gaussien est utilisé pour identifier le système dans l'intervalle de concentration de *0.6 à 0.95*. Ce réseau est entraîné par la méthode hiérarchique avec un paramètre de dispersion fixé à $\sigma = 1.1$ pour tous les neurones. Avec ce paramètre, relativement élevé par rapport à celui du réseau précédent, le nombre de neurones gaussiens générés est 150 pour converger vers une erreur d'environ 0.00004. La figure (V.12) montre la réponse du réseau identificateur et celle du système pour une séquence d'entrées aléatoires.

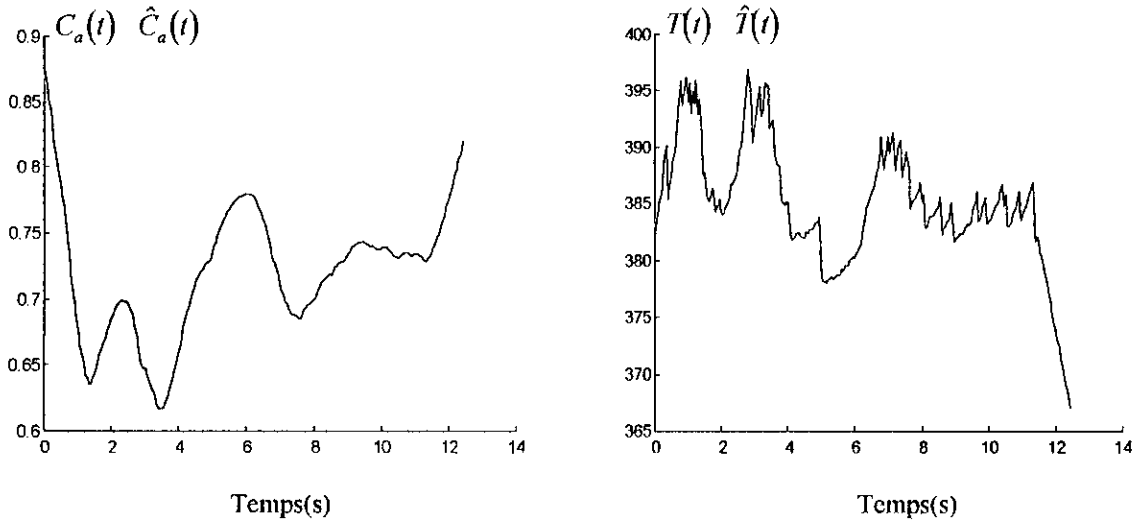


Fig.V.12 Identification du réacteur dans le troisième intervalle de la concentration *[0.6,0.95]* par le réseau RBF
——Soties du modèle neural
·····Soties du système.

Nous remarquons que ce réseau est arrivé à identifier le système avec précision. Il est important de noter que l'entraînement de ce réseau a généré moins de neurones par rapport au réseau utilisé pour l'identification dans le second intervale de fonctionnement. En effet, avec un taux d'apprentissage relativement élevé, les classes générées dans la couche gaussiennes ont des rayons importants. Ainsi le nombre de classes nécessaire pour couvrir toute cette zone de fonctionnement s'en trouve réduit. Il ne faut, cependant, pas trop faire augmenter ces rayons, sinon la capacité du réseau de généralisation sera touchée.

Ainsi, les réseaux RBF Gaussiens nous ont permis de modéliser le système avec une bonne précision. En effet, grâce aux caractéristiques locales de ce type de réseaux, l'identification dans chacun des différents intervalles de fonctionnement est faite avec succès. Notre utilisation d'un réseau pour chaque intérval est dans le but d'assurer la précision de ce modéle, qui va étre utilisé pour le contrôl du réacteur dans le chapitre VI.

Conclusion

Dans cette partie, nous avons examiné les différentes structures possibles d'identification neurale, puis nous avons effectué des opérations d'identification en utilisant les différents réseaux étudiés avec un intérêt particulier aux différentes méthodes d'apprentissage. Dans la dernière partie, nous avons effectué l'identification d'un réacteur chimique, qui sera contrôlé dans le chapitre qui vient.

Pour les réseaux LBF multicouche statique, nous avons constaté que la méthode du Gradient de Second Ordre permet au réseau identificateur de converger beaucoup plus rapidement qu'avec la méthode de Backpropagatoin. De plus, à la lumière des résultats que nous avons obtenus, nous constatons que cette méthode d'entraînement peut trouver des solutions avec des structures de réseaux considérablement moins chargées, en nombre de neurones. Il est utile de rappeler, dans ce cadre, qu'une structure avec un nombre important de neurones peut donner une très bonne précision d'entraînement mais échouer en généralisation.

Notre utilisation des réseaux dynamiques a montré que ces réseaux permettent une bonne identification des systèmes avec un nombre de neurones réduit. Ces réseaux constituent des modèles très intéressants pour l'identification des systèmes dynamiques. Leur structure interne récurrente permet de créer des modèles avec un minimum d'informations sur ces derniers. Le problème avec ces réseaux est leur entraînement, qui est très délicat et qui peut être sujet à l'instabilité. Le choix de taux d'apprentissage adaptatifs nous parait très important et permet d'obtenir des résultats très intéressants.

Les réseaux RBF constituent des modèles très intéressants, qui sont beaucoup adaptés pour l'identification des systèmes. Leur fonctionnement, basé sur une classification puis une approximation, a donné de très bons résultats au court de nos applications et avec un nombre d'itération moins important pour la détermination des poids synaptiques. Ces réseaux nous ont permis un entraînement beaucoup plus rapide et moins onéreux par rapport aux réseaux statique LBF. Ainsi, on peut conclure que le fonctionnement local des réseaux RBF et leur adaptation, rendue plus simple grâce à la couche classificatrice, rendent leur apprentissage rapide et permettent à ces réseaux d'être très appropriés pour l'identification des systèmes.

Dans notre identification du réacteur chimique, les réseaux RBF gaussiens et les réseaux LBF ont été utilisés. Nous avons été obligés d'identifier le système sur trois intervalles différents avec des réseaux RBF différents. En effet, il est important de faire baisser l'erreur en sortie lors de cette opération. Nous avons relevé que ces erreurs en sortie deviennent plus importantes lorsque ces entrées deviennent récurrentes. C'est sous cette structure que nous allons nous servir de ce réseau (récurrent en entrée) pour effectuer la commande du réacteur. D'autre part, la caractéristique local des réseaux Gaussiens nous a permis de concentrer l'apprentissage de chaque modèle sur un des trois intervalles choisis.

Enfin, selon notre étude les réseaux RBF Gaussiens constituent une alternative très intéressante aux réseaux LBF généralement utilisés. Au vu des avantages que nous offre l'utilisation de tels réseaux, nous concluons que ceux ci constituent un moyen très approprié pour les opérations de modélisation.

CHAPITRE VI:

COMMANDE DES SYSTEMES PAR RESEAUX DE NEURONES

Chapitre VI
COMMANDE DES SYSTEMES PAR RESEAUX DE NEURONES

Introduction

L'identification et la commande ont été parmi les premières applications pour lesquelles les réseaux de neurones ont été utilisés, après leur succès dans les opérations de classification, et, notamment, dés l'apparition de la Backpropagation.
En effet avec leur capacité d'approximation universelle, les réseaux de neurones se sont présentés comme une sérieuse alternative pour s'attaquer aux problèmes d'identification et de commande de systèmes non linéaires.

La capacité de ces réseaux à s'adapter suivant le comportement désiré a poussé leur utilisation dans la commande de systèmes dynamiques, et dans le réglage adaptatif. En effet, dans ce cas il est question d'estimer des paramètres pour la détermination de la commande nécessaire[Ahm93, Asr93]. Les caractéristiques des réseaux de neurones ont faits que la commande soit au centre des leurs applications. Nous visons particulièrement parmi ces caractéristiques, le traitement parallèle et distribué, la rapidité et la robustesse au bruit, qui sont d'une extrême importance en commande de systèmes [Bav87, Eil88].

Le grand essor des réseaux de neurones en Automatique, ne date pas de très longtemps. C'est en 1990 que K.S. Narendra et K. Parthasarathy [Nar90], de Yale University aux USA, ont démontré que les réseaux de neurones peuvent être utilisés comme un outil dans le domaine de commande de système dynamique. Les travaux utilisant la technique neural pour s'attaquer aux systèmes dynamiques et non linéaires se sont par la suite multipliés rendant l'utilisation de ces réseaux courante en Automatique[Has92, [Kha96, Nag96].

Ce chapitre se compose de deux parties principales.

Dans la première partie, Nous présenterons les différentes stratégies de commandes par réseaux de neurones. Nous nous intéresserons aux phénomènes liés à l'utilisation des techniques neurales, particulièrement l'apprentissage, dont nous ferrons une étude englobant les différents cas, pouvant se présenter dans les stratégies adoptées.

Dans la seconde partie, des applications de commande par réseaux de neurones seront présentées, où plusieurs stratégies de commande sont proposées.
Dans un premier temps, nous utiliserons un bras de robot, où nous proposerons un régulateur neural linéarisant auto-ajustable, à base de réseaux de neurones RBF Gaussiens.
Par la suite, nous commanderons un réacteur chimique par sa température. Pour cela, nous proposerons trois stratégies, où plusieurs modèles de réseaux sont utilisés, notamment les réseaux ART2 à apprentissage non-supervisé.

A la fin, nous tirerons les conclusions nécessaires, et nous présenterons les perspectives offertes, à travers l'évolution des techniques neurales dans le domaine de l'Automatique.

VI.1. Etude de la commande des systèmes par Réseaux de neurones

La commande neurale est étroitement liée à l'identification. En effet dans la plupart des stratégies de commande on utilise un modèle neural direct ou inverse du système à commander.

Les réseaux de neurones ont été fortement sollicités ces dernières années pour la commande des systèmes non linéaires. Ainsi on peut trouver plusieurs stratégies qui sont proposées dans différents travaux.

Pour faire l'étude de ces stratégies, il est d'abord important de les classifier selon le rôle que les réseaux de neurones doivent remplir dans chaque cas.

VI.1.1 Stratégies de Commande Neurale

Le premier critère selon lequel ces commandes peuvent être distinguées se rapporte directement à l'apprentissage des réseaux. Nous distinguons deux cas différents:

Le premier cas est celui de la commande adaptative où les poids du réseau continuent à être adaptés pendant son utilisation. Le réseau effectue un apprentissage en ligne, ce qui est très délicat et son utilisation se fait sous certaines conditions que nous examinerons. Le second cas est celui où le réseau est entraîné hors ligne, pour être utilisé par la suite sans modification de ses paramètres.

Nous présentons ci-dessous (fig VI.1) une classification plus générale des différentes principales stratégies de commande utilisant les réseaux de neurones.

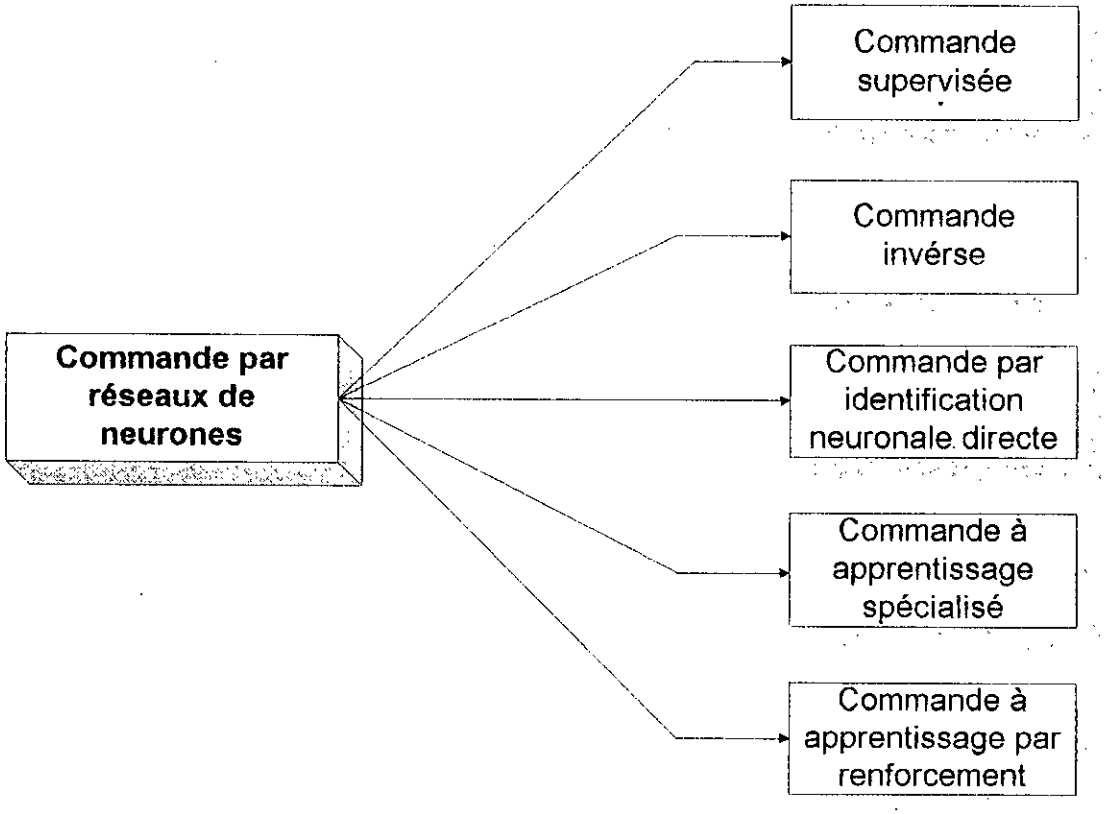


Fig. VI.1 Classifications des différentes structure de commande

VI.1.1.1. Commande supervisée

Cette méthode qui, généralement, offre la possibilité de remplacer l'opérateur humain dans une chaîne de commande, consiste à reproduire une loi de commande déjà existante (fig.VI.2). De plus elle présente les avantages qui sont caractéristiques aux réseaux de neurones, à savoir le traitement parallèle distribué, et la rapidité de calcul.

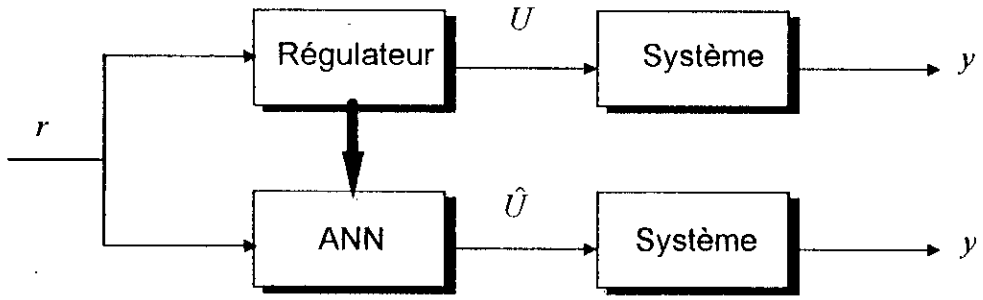


Fig. VI. 2 Schéma de principe de la commande supervisée

VI.1.1.2. Commande inverse

Cette commande consiste à chercher d'abord, un modèle inverse du processus, en utilisant ses sorties *y(t)* (résultant d'une entrée donnée) comme entrée du réseau (fig.VI.3). Après apprentissage, le réseau est placé devant le système pour le commander en boucle ouverte.
L'apprentissage de ce réseau peut s'effectuer de trois maniérés différentes:

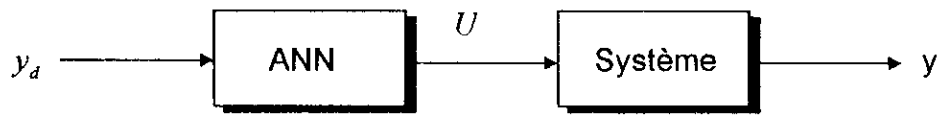


Fig. VI.3 Schéma de la commande inverse

a. Apprentissage indirecte

On injecte au réseau les sorties désirées y_d afin qu'il fournisse un signal de commande u. La réponse du système pour ce signal de commande est injectée à un second réseau qui doit être une copie du premier (figure VI.4). L'idée est de faire tendre l'erreur entre la sortie du réseau qui assure la commande et celle du second réseau $e=u-v$, vers zéro. Mais ceci n'implique pas nécessairement, que la sortie $\varepsilon = y_d - y$ (fig.VI.4), s'annule. En effet, lors de l'entraînement, le réseau peut fournir le même signal de commande pour des sorties désirées différentes. Un tel entraînement peut conduire à une solution triviale indésirable; un réseau à sortie identiquement nulle, par exemple [Psa90].

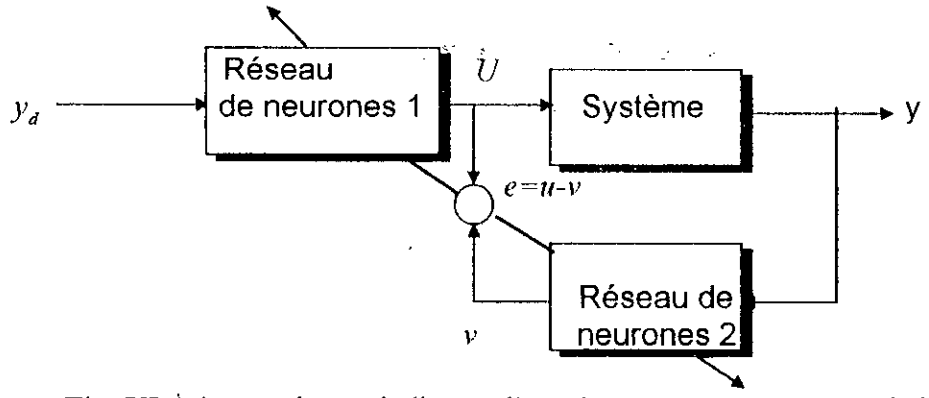


Fig. VI.4 Apprentissage indirecte d'un réseau pour une commande inverse

b. Apprentissage général

Dans l'apprentissage général, on utilise les signaux de commande que l'on injecte au système. Les sorties du système constituent les entrées avec lesquelles le réseau est entraîné (figure VI.5). Contrairement au cas précédent, dans cette stratégie on peut choisir les signaux de commande qui constituent les entrées du système. Le problème cette fois ci est que nous ne disposons pas d'informations sur les sorties que ces dernières peuvent produire. En effet, les entrées choisies pendant l'apprentissage peuvent ne pas du tout convenir aux objectifs recherchés en sortie. Cet entraînement peut, d'autre part, sortir de la zone d'intérêt que l'on recherche, à savoir même de la zone de fonctionnement du système (l'entraînement se fait hors ligne). En outre, comme dans le cas précédent, l'existence et l'unicité d'un processus inverse n'est pas évidente notamment pour les systèmes dynamiques [Ren95].

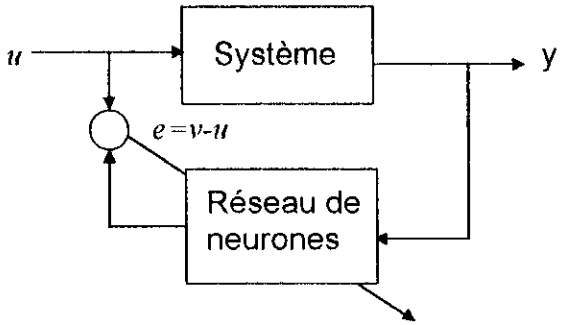


Fig. VI.5 Apprentissage général d'un réseau pour une commande inverse.

c. Commande inverse avec entraînement spécialisé

Dans cette approche, on remédie aux problèmes rencontrés dans les deux techniques précédentes. Les sorties désirées sont choisies, et l'erreur à minimiser est celle à la sortie du système. Un réseau de neurones est entraîné pour fournir les signaux de commande nécessaires pour ramener les sorties du système aux valeurs désirées (fig.VI.6). Cette structure, qui n'est qu'un cas particulier de la commande à apprentissage spécialisé, constitue l'approche la plus efficace. Néanmoins, si les principaux inconvénients de l'entraînement général sont évités, l'apprentissage du réseau pose un sérieux problème, que nous examinerons dans le cadre de la commande spécialisée.

VI.1.1.3 Commande par identification neurale directe

Cette structure est très générale, et peut être souvent utilisée. Un modèle suffisamment précis du système doit être établi, afin de l'utiliser pour la construction d'une stratégie de commande. Un réseau de neurones est entraîné, pour modéliser le système avec suffisamment de précision par les procédures d'identification neurale déjà étudiées. Les sorties de ce réseau sont utilisées pour l'optimisation d'un critère, sur la base duquel la commande est calculée. Cette approche englobe plusieurs types de commande, et est largement utilisée notamment en commande adaptative.

Dans une stratégie de commande, C.Chen utilise cette approche, où deux réseaux identificateurs ont été entraînés pour la modélisation d'un système, afin de les utiliser dans l'élaboration d'un régulateur Auto-ajustable [Che90]. Cette structure peut être aussi utilisée pour la minimisation d'un critère optimal, dont le rôle est de ramener les sorties futures du système vers un comportement de référence. Le réseau, dans ce cas, est utilisé comme prédicateur. Ainsi, J.Hunt a présenté une stratégie appelée *commande prédictive* où le réseau identificateur est utilisé pour prédire les sorties du système sur un horizon fini. Les prédictions de ce réseau sont utilisées, dans une routine d'optimisation numérique, qui a pour rôle de minimiser un critère de performance, et calculer le signal de commande nécessaire pour cela [Hun92].

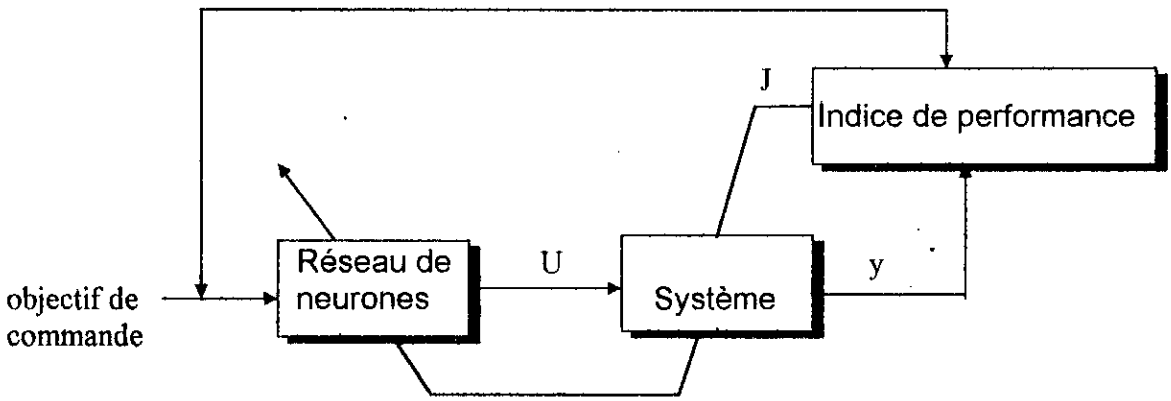


Fig. VI.6 Schéma de commande à apprentissage spécialisé.

VI.1.1.4. Commande avec apprentissage spécialisé

Dans cette approche, un réseau de neurones commande le système directement, et utilise l'erreur à la sortie du système (ou, en général, un indice de performance), pour ajuster ses poids en ligne (fig.VI.6) (fig.VI.7). Cette méthode, qui est très proche de la commande adaptative traditionnelle, est très intéressante, notamment si on dispose d'assez de connaissances sur le système à commander. Cependant, problème qui est inhérent à l'apprentissage des réseaux de neurones, l'entraînement du neuro-contrôleur peut poser de sérieuses difficultés. En effet, l'apprentissage des réseaux régulateur nécessite, en général, la rétro-propagation du jacobien du critère à minimiser, de la sortie à travers les différentes couches du réseau; Ceci nécessite des connaissances, a priori, sur le système à commander (le Jacobien, ou la sensibilité), pour pouvoir calculer correctement les modifications à apporter aux poids synaptiques [Lig95]. Pour remédier à ce problème, qui se pose notamment lors de la commande adaptative avec modèle de référence, des solutions ont été apportées. Nous aborderons ce problème dans une étude, que nous consacrerons à l'entraînement des réseaux contrôleurs, à la fin de ce chapitre.

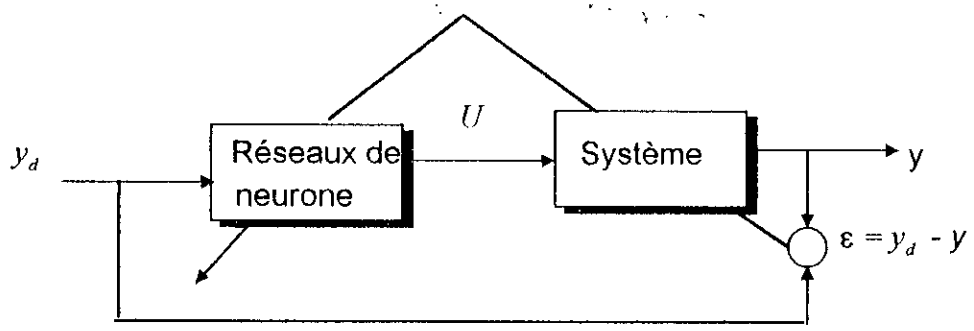


Fig.VI.7 Schéma d'un type simple d'apprentissage spécialisé

VI.1.1.5. Commande à apprentissage par renforcement

La stratégie de cette commande repose sur la minimisation d'un critère, appelé fonction critique. Son approche se distingue, par rapport aux autres, par le fait que son utilisation ne nécessite pas de modèle pour le système, et par sa technique d'apprentissage propre à elle. L'entraînement est basé sur l'apprentissage par renforcement, où deux réseaux sont associés (figure VI.8): Le réseau critique, qui évalue le critère, et le réseau d'action, qui doit générer la commande[Fuk92]. C'est le réseau critique, qui oriente la réadaptation des poids du réseau contrôleur. Ce réseau envoie, à chaque instant un signal, appelé signal de renforcement, qui soit renforce les poids actuels du contrôleur ou les punit, suivant l'effet apporté sur le critère à optimiser. Il est clair que la recherche du minimum dans cette approche est, plutôt, heuristique (notamment par rapport à la méthode de Backpropagation) et la direction de recherche n'est pas très précise. Cependant, cette méthode a été utilisée avec succès [Sag92]; elle présente, en effet, l'avantage de ne pas nécessiter beaucoup de connaissances sur le système à commander. Ceci, rend l'utilisation de la commande à apprentissage par renforcement appropriée dans les situations, où les informations font défaut [Hun92].

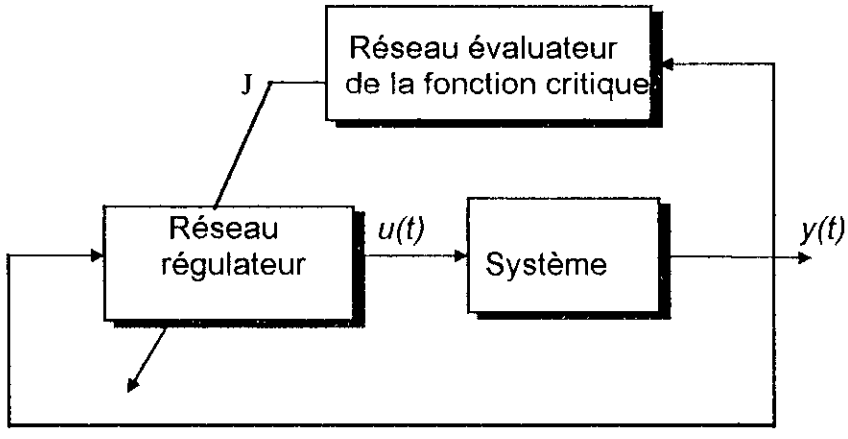


Fig. VI. 11 Schéma de la Commande par la fonction critique.

VI.1.2 Commande adaptative par réseaux de neurones

La commande adaptative par réseaux de neurones est, généralement, appliquée en remplaçant les fonctions remplies par les différents blocs utilisés en commande adaptative classique, par des réseaux de neurones. M.Saad a montré qu'une telle commande peut nous libérer de la nécessité d'établissement d'un modèle paramétrique de représentation pour le système, et de l'estimation de ces paramètres. Ceci, grâce à la capacité des réseaux de neurones, en estimation de modèles dynamiques et leurs inverses; ce qui la rend adaptée pour la commande des bras de robots [Saa94a] [Saa94b].
Cette commande neurale a été réalisée sous son aspect de régulateurs *Auto-ajustables* et aussi celui de *modèle de référence.*

VI.1.2.1 Régulateur Auto-ajustable

Un régulateur Auto-ajustable à base de réseaux de neurones, vise à exploiter les capacités de ces réseaux en approximation, pour l'estimation des paramètres définissant le système, afin de les utiliser par le régulateur, pour le calcul de la commande.
Tokita et al [Fuk92] proposent un régulateur Auto-ajustable pour la commande d'un bras de robot manipulateur, où un réseau de neurones est utilisé pour l'estimation des paramètres, définissant le système utilisé. Ces paramètres sont utilisés pour identifier la dynamique du système, et prédire ses sorties futures. Le régulateur est basé sur la minimisation de l'erreur de prédiction.

F.C.Chen [Che90] propose, d'une manière générale, un processus non linéaire commandé par un régulateur Auto-ajustable, utilisant les réseaux de neurones. Le régulateur est "ordinaire", et le réseau de neurones sert à estimer les paramètres utilisés par ce régulateur.

Le système, pour lequel cette stratégie est utilisée, est non linéaire et de la forme:

$$\begin{aligned} y(k+1) = f\big(y(k), y(k-1), \ldots, y(k-n),\ u(k-1), u(k-2), \ldots, u(k-n)\big) + \\ g\big(y(k), y(k-1), \ldots, y(k-n),\ u(k), u(k-1), \ldots, u(k-n)\big)u(k) \end{aligned} \tag{VI.1}$$

Si les fonctions *f(.) et g(.)* sont connues, pour que la sortie du système $y(k+1)$ suive la sortie désirée $d(k+1)$, on peut utiliser une commande de la forme:

$$u(k) = -\frac{f(.)}{g(.)} + \frac{d(k+1)}{g(.)} \tag{VI.2}$$

Dans le cas contraire, un réseau de neurones est utilisé, pour faire l'approximation de ces fonctions, qui seront par la suite utilisées pour calculer la commande.

Pour un système de premier ordre, on a:

$$y(k+1) = f\big(y(k)\big) + g\big(y(k)\big)u(k) \tag{VI.3}$$

Ainsi, en estimant les fonctions $\hat{f}(.)$ et $\hat{g}(.)$ par les réseaux de neurones, un modèle du système défini par l'équation (VI.1) peut être élaboré.

Ce modèle est donc défini par l'équation suivante:

$$\hat{y}(k+1) = \hat{f}\left(y(k), W(k)\right) + \hat{g}\left(y(k), V(k)\right)u(k) \tag{VI.4}$$

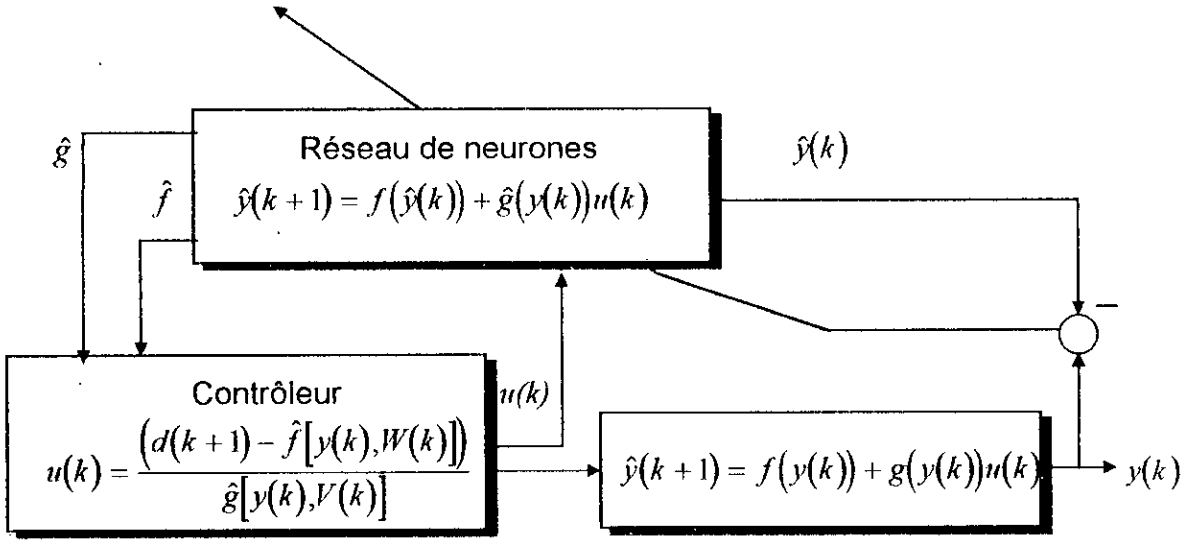


Fig. VI.9. Régulateur Auto-ajustable utilisant les réseaux de neurone

Pour que le système poursuive la sortie désirée $d(k)$, La commande à appliquer au système et au modèle doit, donc être de la forme:

$$u(k) = \frac{\hat{f}\left[y(k), W(k)\right]}{\hat{g}\left[y(k), V(k)\right]} + \frac{d(k+1)}{\hat{g}\left[y(k), V(k)\right]} \tag{VI.5}$$

A partir de (VI.2) et (VI.4), la sortie du système sera de la forme:

$$y(k+1) = f\left(y(k)\right) + g(k)\left\{-\frac{\hat{f}\left[y(k), W(k)\right]}{\hat{g}\left[y(k), V(k)\right]} + \frac{d(k+1)}{\hat{g}\left[y(k), V(k)\right]}\right\} \tag{VI.6}$$

Les poids synaptiques *V(k)* et *W(k)* peuvent, donc, être réadaptée en minimisant l'erreur:

$$E(k) = \left(d(k+1) - y(k+1)\right)^2 \tag{VI.7}$$

Pour cela, l'algorithme de Backpropagation est utilisé. Dans la réadaptation des poids, la fonction *g(k)* inconnue est remplacée par son signe pour le calcul des dérivées [Che90].

VI.1.2.2. Commande adaptative avec modèle de référence

Dans cette stratégie, la commande a pour rôle d'obliger le système à 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:

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 coïncident avec ceux du système réel (fig.VI.10). C'est ce qui est appelé en commande adaptative classique *le principe d'équivalence certaine* [Äst89].
La réussite d'une telle commande dépend étroitement de la précision avec laquelle le réseau de neurones arrive à estimer les paramètres du système.

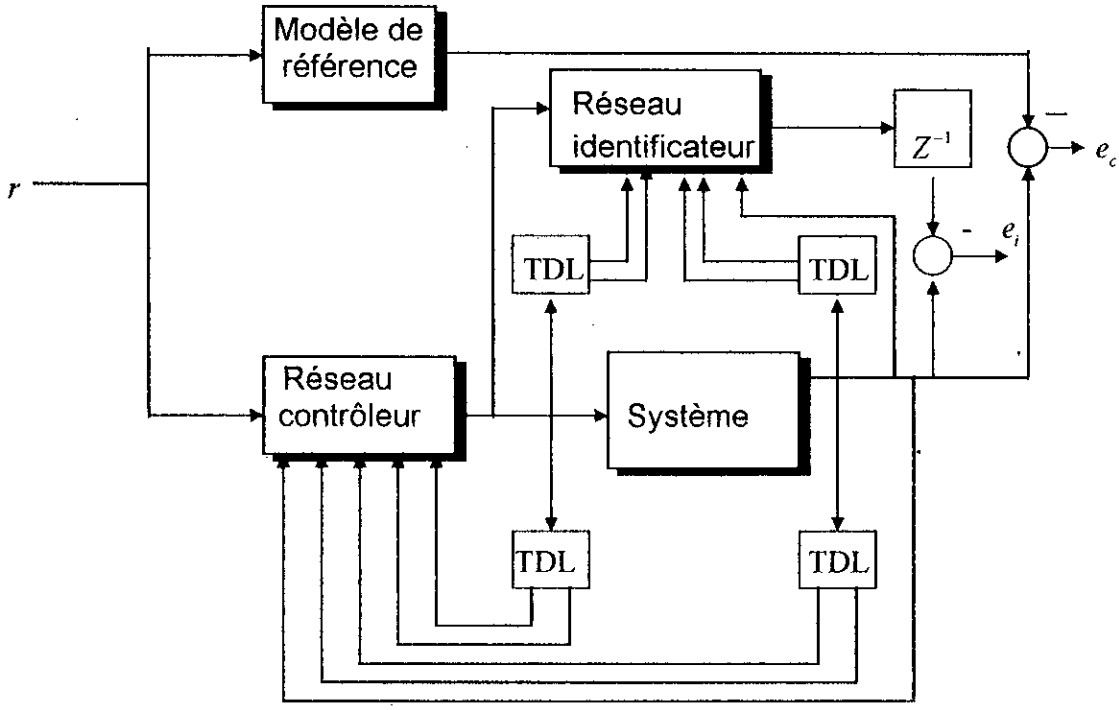


Fig.VI.10 Commande adaptative indirecte par réseaux de neurones

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 (fig.VI.11).
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* [Nar89]. Cependant, dans le contexte neural, 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 ne pas créer d'effet déstabilisateur. Cette condition est nécéssaire aussi, 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 [Ren 95].

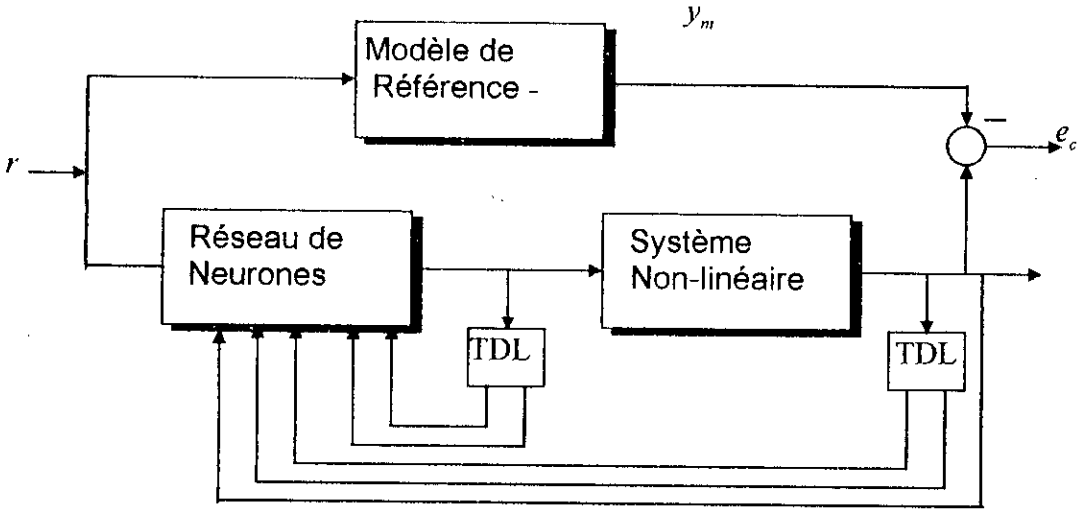


Fig.VI 11 Commande adaptative directe par réseaux de neurones.

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.

Dans ce qui suit, nous allons faire un examen des techniques qui ont été utilisées, séparément, dans différents travaux pour surmonter cet obstacle, et nous exposerons une solution, que nous avons proposé dans une application spécifique dans notre travail.

VI.1.3 Apprentissage des réseaux de neurones contrôleurs

Le problème d'apprentissage des réseaux de neurones en commande est lié aux sorties désirées que les réseaux contrôleurs 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'entraînement 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 rétro-propagation 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.

Dans nos recherches bibliographiques, nous avons rencontré plusieurs cas où des solutions, plus où moins spécifiques aux différentes situations, ont été apportées.

La solution la plus triviale consiste à calculer la chaîne de dérivées partielles jusqu'à aboutissement au réseau. Dans ce cas le calcul du Jacobien du système est donc nécessaire.

en posant $e(k) = \left(d(k) - y(k)\right)^2$

ona: (VI.8)

$$\frac{\partial\, e(k)}{\partial W} = -\left(d(k) - y(k)\right)\frac{\partial y(k)}{\partial W}$$

Si nos connaissances sur le système le permettent, ce calcul peut être fait. D.Psaltis[Psa 90] 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 lui 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.Saerens et A.Soquet [Ren 95] proposent, dans ce cas, de "se contenter" du signe du Jacobien dans l'équation (VI.8) pour l'entraînement du réseau. En effet, cette information est souvent facile à obtenir à partir de connaissances qualitatives sur le système, et nous guide sur le sens de variation de la sortie. Mais l'entraînement 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 [Psa 90]. Celui ci propose d'approcher ce calcul, en provoquant des petites perturbations locales δu aux entrées u injectées au système, et relever à chaque étape les variations δy à la sortie. Ainsi, on aura dans la chaîne de différentiations, qui nous mène vers la sortie du réseau contrôleur dans l'équation (VI.8):

$$\frac{\partial y}{\partial u} \approx \frac{y(u+\delta u) - y(u)}{\delta u} \qquad \text{(VI.9)}$$

Enfin, la solution, qui peut faire l'unanimité, est proposée par Nguyen et Widrow [Wid 92]. Elle consiste à utiliser un réseau de neurone entraîné à modéliser le système par les procédés d'identification, que nous avons déjà étudiés (cf. chap.V). Ce réseau, appelé *réseau émulateur*[Ren95], est utilisé comme un "canal " à travers lequel l'erreur est acheminée, pour parvenir au réseau contrôleur. Ce réseau sera excité par les mêmes 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[Nar92].

Cette solution sera utilisée lors de la commande d'un réacteur chimique, que nous effectuerons dans ce travail(cf.VI.4.3).

Ainsi nous obtenons pour le calcul des dérivées de l'erreur par rapport au vecteur poids w_j du neuro-contrôleur:

$$\frac{\partial E}{\partial w_j} = \frac{1}{2}\sum_k \frac{\partial}{\partial w_j}\left(d_k - \hat{y}_k\right)^2 = -\sum_k \left(d_k - \hat{y}_k\right)\frac{\partial \hat{y}_k}{\partial w_j} \qquad \text{(VI.10)}$$

où $\hat{y}_k$ représente la *k iéme* sortie du réseau qui identifie le système (émulateur) et E l'erreur quadratique entre la sortie désirée, d_k, et celle du réseau émulateur.
Le jacobien est approximé en calculant les dérivées partielles du réseau identificateur, comme dans le cas de la méthode de Backpropagation classique (cf.VI.4.2.1) La rétro-propagation est appliquée sans, toutefois, modifier les poids du réseau émulateur.

G.Lightbody a utilisé la technique du réseau émulateur pour estimer 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 [Ligh95].

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 Backpropagation pour l'entraînement du réseau contrôleur.
Pour éviter ce calcul, nous proposons dans notre étude une solution basée sur l'application de la méthode d'Optimisation Aléatoire ROM (présenté au chapitre II) pour l'entraînement du réseau.
Cette méthode s'est révélée bien adaptée pour les problèmes de commande à entraînement spécialisé. Il a été, en effet, constaté qu'elle ne dépend pas du système à commander et ne nécessite des informations sur ce dernier. Elle est utilisée pour l'optimisation d'un critère. De ce fait, cette méthode nous libère des contraintes mathématiques rencontrées dans l'utilisation de la Backpropagation. Cependant notre utilisation de cette technique, qui est basée sur des tests effectués à chaque itération, -ce qui constitue son inconvénient principal- pour un entraînement en temps réel a nécessité son adaptation à ce type de problème (cf.VI.3.2). Il a été, notamment, nécessaire de limiter la vitesse d'adaptation des poids, afin de ne pas affecter la stabilité du processus.

VI.2 Commande basée sur la classification par régions de fonctionnement

Dans notre travail, nous avons proposé une structure de commande neurale, permettant de contrôler les systèmes, dont le fonctionnement varie d'une région à l'autre. Elle peut aussi être utilisée, pour les systèmes dont l'étendu de fonctionnement est très important, pour que la commande soit générée par un seul neuro-controleur.
l'idée de base de notre procédure consiste à utiliser un réseau classificateur à entraînement non-supervisé, du type que nous avons présenté dans le second chapitre de ce travail (cf. Ch.II) (ART, BAM, Hamming...). Ceci, afin de classifier les entrées et les états du système dans l'espace suivant la zone de fonctionnement.
Pour cela le fonctionnement du système est étudié dans chacune de ces zones. Des régulateurs neuraux adéquats peuvent, par la suite être conçus pour le fonctionnement du système à l'intérieur de chacune de ces zones, suivant les différents objectifs à atteindre.

Le réseau à apprentissage non supervisé est entraîné avec les états et les entrées de commande du système. La classification se fait par les algorithmes que nous avons présentés dans le chapitre II.
Le réseau classificateur et les autres différents Neuro-contrôleurs sont disposées comme indiqué sur la figure (VI.12). Ce réseau joue le rôle d'un "aiguilleur" qui stimule le bon régulateur à chaque instant pendant le fonctionnement du système, suivant l'état de ce dernier.

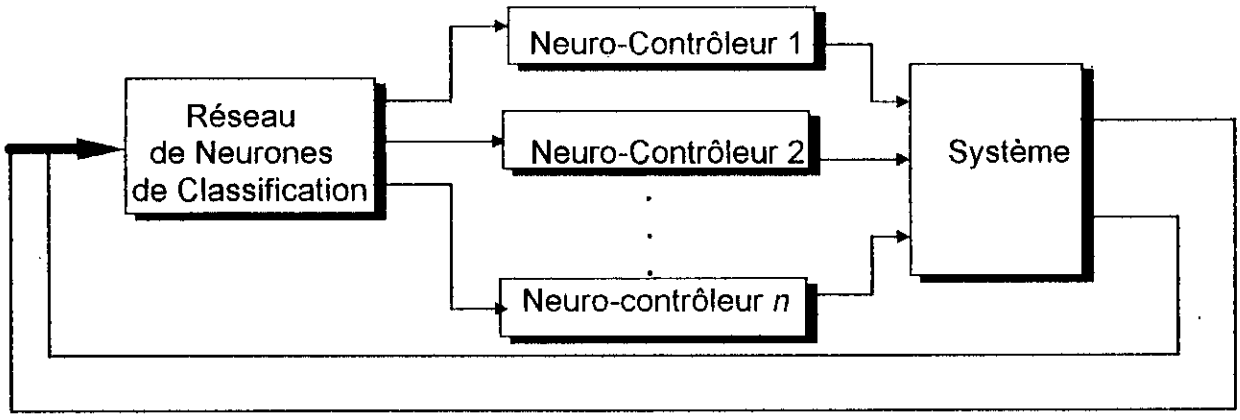


Fig VI.12. Schéma explicatif sur le principe de la commande élaborée basée sur la classification.

Une telle stratégie de commande, qui a montré une très bonne efficacité dans nos applications, acquiert une grande importance. Celle ci nous permet de commander les systèmes dont l'objectif de commande peut varier d'un intervalle à l'autre. En outre, celle ci permet une prise de décision plus rapide, en temps réel, et notamment de concevoir un dispositif entièrement neural, pour le contrôle de ces systèmes.

Dans cette partie, nous avons passé en revue les différentes stratégies de commande par réseaux de neurones. A partir des différentes structures de commande rencontrées dans nos recherches bibliographiques, nous avons essayé de regrouper et classifier les différentes techniques afin de rendre l'exposé plus clair.

Nous avons rencontré dans la littérature des applications de commandes très intéressantes qui généralement peuvent être classés parmi les différentes approches que nous avons présentées. Parmi ses applications se trouvent des méthodes récentes qui ne sont pas encore répondues. Nous citons la *commande avec modèle interne* présentée en utilisant les réseaux de neurone par Hunt et Sbarbaro [Sba92]. Dans cette méthode deux réseaux dont l'un présente le modèle direct du système et l'autre le modèle inverse sont utilisés et judicieusement placés dans une boucle de régulation afin d'assurer la réjection de certaines perturbations [Ren95].

Enfin il est important de remarquer que l'efficacité des réseaux de neurones en commande est très étroitement liée à leurs aptitudes en approximation. En effet dans la plupart des cas de commande l'établissement d'un modèle direct ou inverse du système est nécessaire. Et l'identification de processus non linéaire est une discipline où les réseaux de neurones continuent de montrer de bonnes performances. Ceci renforce le succès des réseaux de neurones dans la commande de processus non linéaires.

VI.3 Commande d'un bras de robot

La commande de bras de robots est l'une des applications où les réseaux de neurones ont souvent été sollicités. En effet l'aptitude des réseaux de neurones à s'adapter ou à modéliser les systèmes rend leur utilisation pour la commande des robots très intéressante, particulièrement lorsqu'il s'agit d'une commande adaptative.

Dans cette application nous utiliserons une commande neurale linéarisante auto-ajustable. Pour le contrôle d'un bras de robot. Dans un premier temps, une commande neurale supervisée lui est, d'abord appliquée.

Nous avons choisi un bras de robot de classe quatre qui est parmi les 8 classes de bras manipulateurs que D.Stoten a établis [Sot89]. Ce manipulateur comprend une articulation rotationnelle identifiée par l'angle θ_2 et deux articulations translationnelles représentées par les variables d_1 et d_2 [Gue95].
La figure (VI.13) représente un schéma pour ce bras de robot.

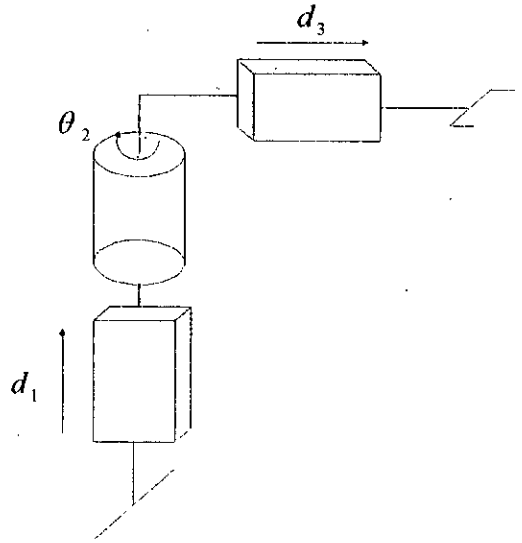


Fig. VI 13 Schémas du bras de robot

Le vecteur des coordonnées généralisées correspondant au modèle géométrique du bras est:

$$q^T = [q_1 \quad q_2 \quad q_3] = [d_1 \quad \theta_2 \quad d_3] \tag{VI.11}$$

Le modèle dynamique du bras est décrit par :

$$\begin{aligned} \dot{X} &= AX + BU + D \\ Y &= CX \end{aligned} \tag{VI.12}$$

Où:

$X^T = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6] = [\dot{q}_1 \quad \dot{q}_1 \quad q_2 \quad \dot{q}_2 \quad q_3 \quad \dot{q}_3]$ est le vecteur d'état.
$Y^T = [Y_1 \quad Y_2 \quad Y_3] = [q_1 \quad q_2 \quad q_3]$ vecteur des sorties du système.
$U^T = [U_1 \quad U_2 \quad U_3]$ vecteur commande.

$$A=\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{f_1}{m_1+m_3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{f_2}{m_3 j^*} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -\frac{f_3}{m_3} \end{bmatrix}; \quad B=\begin{bmatrix} 0 & 0 & 0 \\ \frac{k_1}{m_3+m_1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{k_2}{m_3 j^*} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{k_3}{m_3} \end{bmatrix} \qquad \text{(VI.13)}$$

$$C=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}; \quad d^T=\begin{bmatrix} 0 & -g & 0 & -\frac{2\left(x_5-\frac{l_2}{2}\right)x_4x_6}{j^*} & 0 & \left(x_5-\frac{l_2}{2}\right)x_4^2 \end{bmatrix} \qquad \text{(VI.14)}$$

$$j^*=\frac{l_2^2}{3}-l_2x_5+x_5^2$$

Tel que pour chaque liaison ($i=1,...,3$), on a:

f_i : Coefficient de frottements visqueux.
m_i : Masse de la liaison.
l_i : Longueur de la liaison.
Avec $F_1=k_1U_1$ et $F_3=k_3U_3$ pour les forces appliquées aux liaisons translationnelles et $\tau_2=k_2U_2$ pour le couple appliqué à la liaison rotationelle.

Les valeurs numériques des caractéristiques du robot sont données par D.Stoten [Gue 85]:

$m_1=20\ Kg$; $m_3=10\ Kg$; $l_2=1.5\ m$; $g=9.81\ ms^{-2}$; $k_1=100\ N/V$; $k_2=10\ Nm/V$;
$k_3=10\ N/V$; $f_1=30\ Nsm^{-1}$; $f_2=7.825\ Nm\ rad^{-1}s$; $f_3=20\ Nsm^{-1}$.

VI.3.1 Commande Neurale par retour linéarisant

Dans un premier temps, nous nous proposons d'effectuer, par réseau de neurones, la linéarisation du bras de robot à travers un retour non linéaire[Asr93]. A travers un placement de pôles nous commanderons ce bras à suivre une trajectoire de référence.

Soit un système non linéaire de la forme suivante:

$$\dot{X}=f(X)+b(X)U \qquad \text{(VI.15)}$$

où $f(X)$ et $b(X)$ sont des fonctions non-linéaires. En général, afin de compenser les non-linéarités du système décrit par l'équation (VI.15) et le contrôler à suivre un modèle de référence, La commande doit avoir la forme suivante:

$$U = \frac{-K^T X + cw - f(X)}{b(X)} \tag{VI.16}$$

Tel que K et c définissent la dynamique du système de référence et w représente le signal de référence $w^T = (r_1, r_2, r_3)$.

La stratégie adoptée consiste à entraîner un réseau à l'approximation d'un retour non linéaire de sorte, qu'après apprentissage, le robot soit découplé.
La conception de ce retour d'état est basée sur la géométrie différentielle. Cette méthode consiste à transformer les variables d'état du système en une forme canonique de commande, puis calculer les retours d'état de façon à compenser les non-linéarités et découpler le système. Après l'entraînement du réseau, chaque sous- système se comporte comme un double intégrateur [Isi 89].

A partir des équations (VI.12), (VI.13), (VI.14), les commandes nécessaires à générer pour cela peuvent être calculées et donc définies par:

$$\begin{aligned}
U_1 &= \frac{m_1 + m_3}{K_1}\left(\frac{f_1}{m_1 + m_3} x_2 + g\right) + \frac{m_1 + m_3}{K_1} r_1 \\
U_2 &= \frac{m_3}{K_2}\left(\frac{f_2}{m_3} x_4 + 2\left(x_5 - \frac{l_2}{2}\right) x_4 x_6\right) + \frac{m_3 j^*}{K_2} r_2 \\
U_3 &= \frac{m_3}{K_3}\left(\frac{f_3}{m_3} x_6 - \left(x_5 - \frac{l_2}{2}\right) x_4^2\right) + \frac{m_3}{k_3} r_3
\end{aligned} \tag{VI.17}$$

où r_1, r_2 et r_3 représentent les entrées de référence de chacun des trois sous systèmes.

Il s'agit donc, dans ce cas, d'une commande supervisée (cf. VI.1.1.1). Le schéma de commande global est montré sur la figure (VI.14).

Trois réseaux de neurones sont utilisés. Le premier est un réseau LBF à deux couches cachées, respectivement de 9 et 10 neurones et deux sorties, entraîné par la méthode de Backpropagation avec Momentum et taux d'apprentissage dynamique. Les deux autres sont des réseaux RBF à trois entrées, une couche cachée gausienne et deux sorties.

Chaque réseau, contrôlant une articulation, est organisé comme montré sur la figure (VI.15)

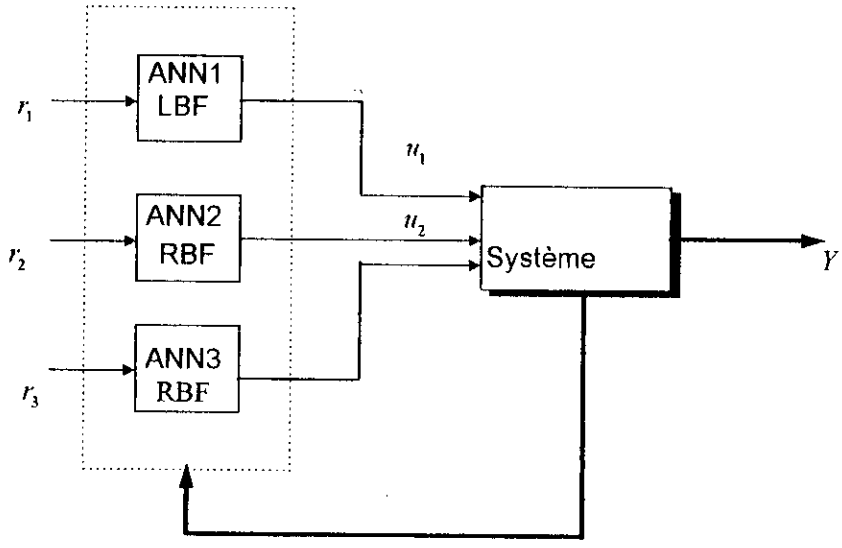


Fig. VI.14 Schéma de la commande neurale linéarisante.

Pour le réseau LBF le taux d'apprentissage initial est égal à 0.1, et le coefficient du momentum à 0.9. Son entraînement a nécessité 2000 itérations. Les réseaux RBF ont un coefficient de dispersion commun à tous les neurones égal à 0.4. Pour leur entraînement, 96 neurones cachés ont été créés pour ANN2 et 102 pour ANN3; ceci afin d'obtenir à la sortie une erreur en position de l'ordre de 10^{-4} pour chaque articulation.

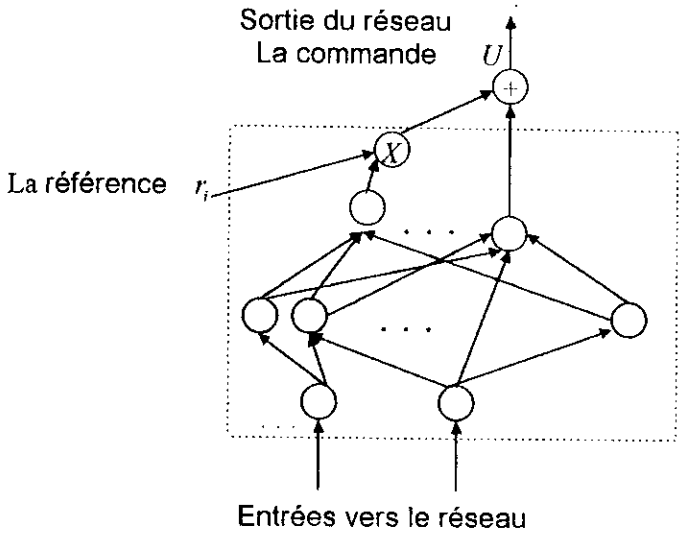


Fig. VI.15 Schéma des sorties des réseaux générant la commande de chaque articulation

Nous avons excité le système avec des entrées indicielles. La figure (VI.16) présente les réponses de chaque sous-système avec les commandes générées par le neuro-contrôleur.

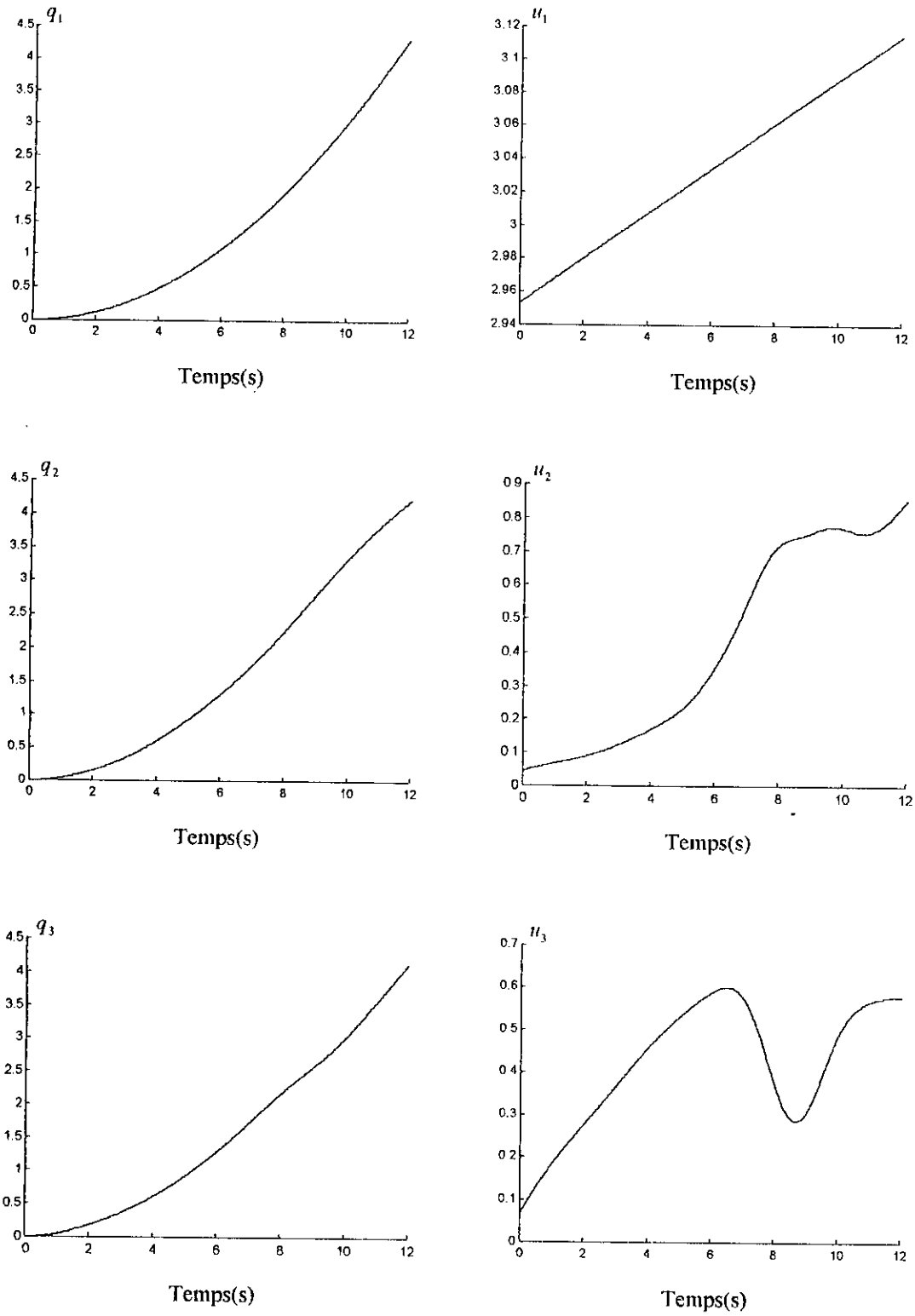
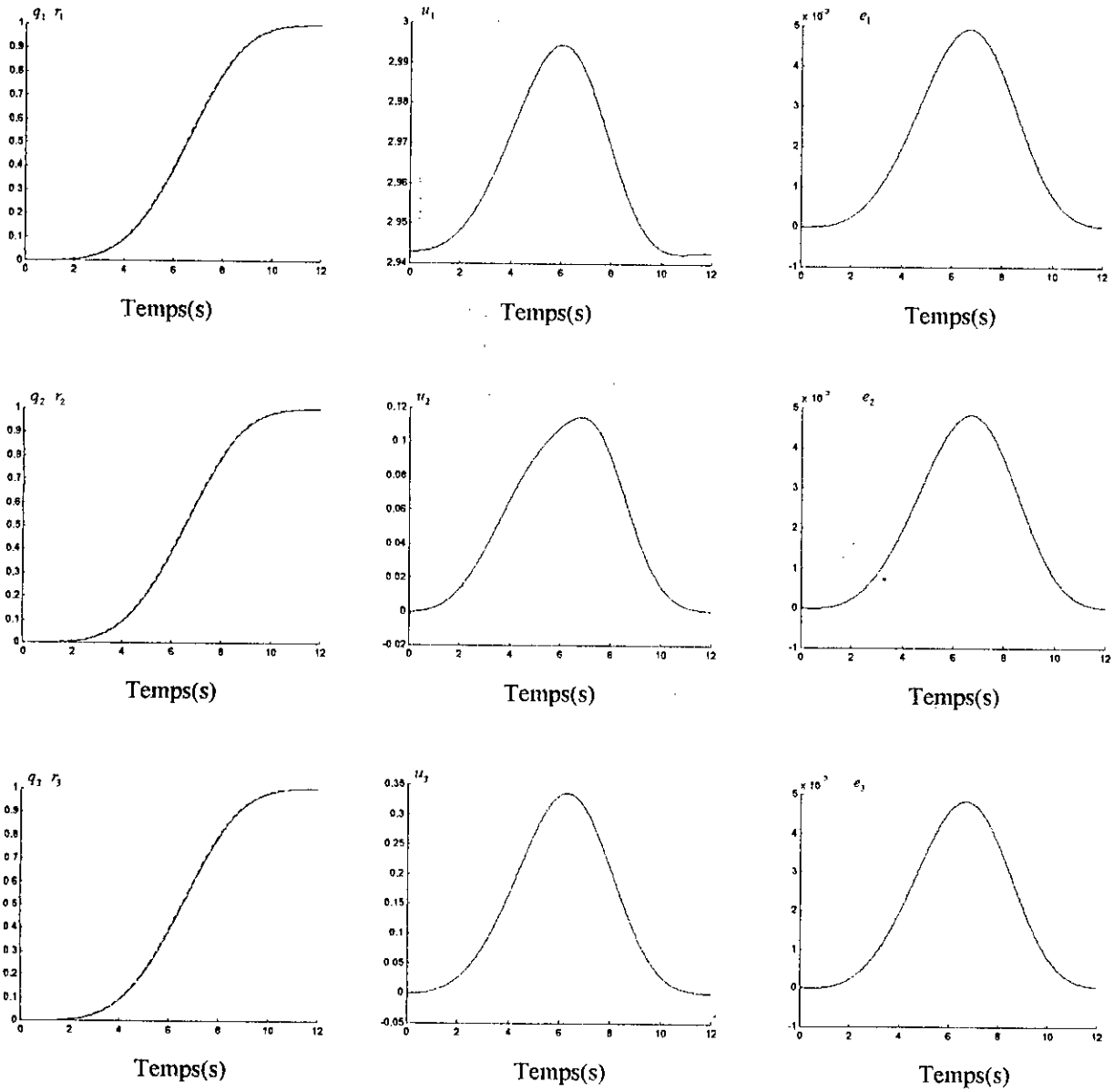


Fig.VI.16 réponse des trois articulations en position pour des entrées indicielles.

Nous remarquons que chaque sous-système a un comportement intégrateur. Ceci montre que les trois articulations du bras ont été linéarisées et découplées.

Nous avons procédé à un placement de pôles pour ces trois articulations. Les pôles choisis sont $p_1 = -20; p_2 = -15$. Nous avons testé le bras manipulateur avec une trajectoire cycloïde. Celle ci est lisse et vérifie les conditions de continuité et de dérivabilité. Les réponses de chaque articulation, les commandes générées par le régulateur neural, ainsi que l'erreur en position de chaque articulation sont montrées sur la figure (VI.17).



a. Position b. Commande c. Erreur

Fig.VI.17 réponse du bras en position pour une trajectoire cycloïde.
——— Positions des trois articulations.
····· Trajectoirede référence.

Dans cette partie, il s'agit d'une commande supervisée. Le neuro-contrôlleur, à base de deux réseaux RBF et un réseau LBF, est entraîné une seule fois pour découpler le système et le linéariser, à travers un retour qui compense les non-linéarités.
Cette structure de commande est figée, et ne peut pas faire face aux variations que le système peut subir. Or, pour un bras manipulateur, il est plus intéressant de concevoir une commande adaptative, qui peut s'ajuster pour éviter que les performances en sortie ne se dégradent. En effet, un bras de robot peut voir ces paramètres modifiés. Les raisons de telles modifications peuvent revenir aux effet de vieillissement ou, notamment, aux variations des masses.

Pour cela, nous proposons dans la partie suivante un régulateur neural auto-ajustable, dont les paramètres varient lors du fonctionnement du système.

VI.3.2 Commande Neurale linéarisante Auto-Ajustable

Dans cette partie, un régulateur auto-ajustable, à base de réseaux de neurones RBF Gaussiens, est utilisé pour la commande du bras manipulateur. Ainsi, les réseaux RBF Gaussiens seront utilisés pour un entraînement en temps réel. Ce régulateur auto-ajustable doit faire face aux modifications qui vont être apportées sur le modèle du bras (équ. VI.12;VI.13). Ces modifications représentent des cas d'erreur d'identification ou de variation des paramètres du bras de robot $(m_1, m_3, l_2, k_1, k_2, k_3, f_1, f_2, f_3)$., dues à multiples raisons

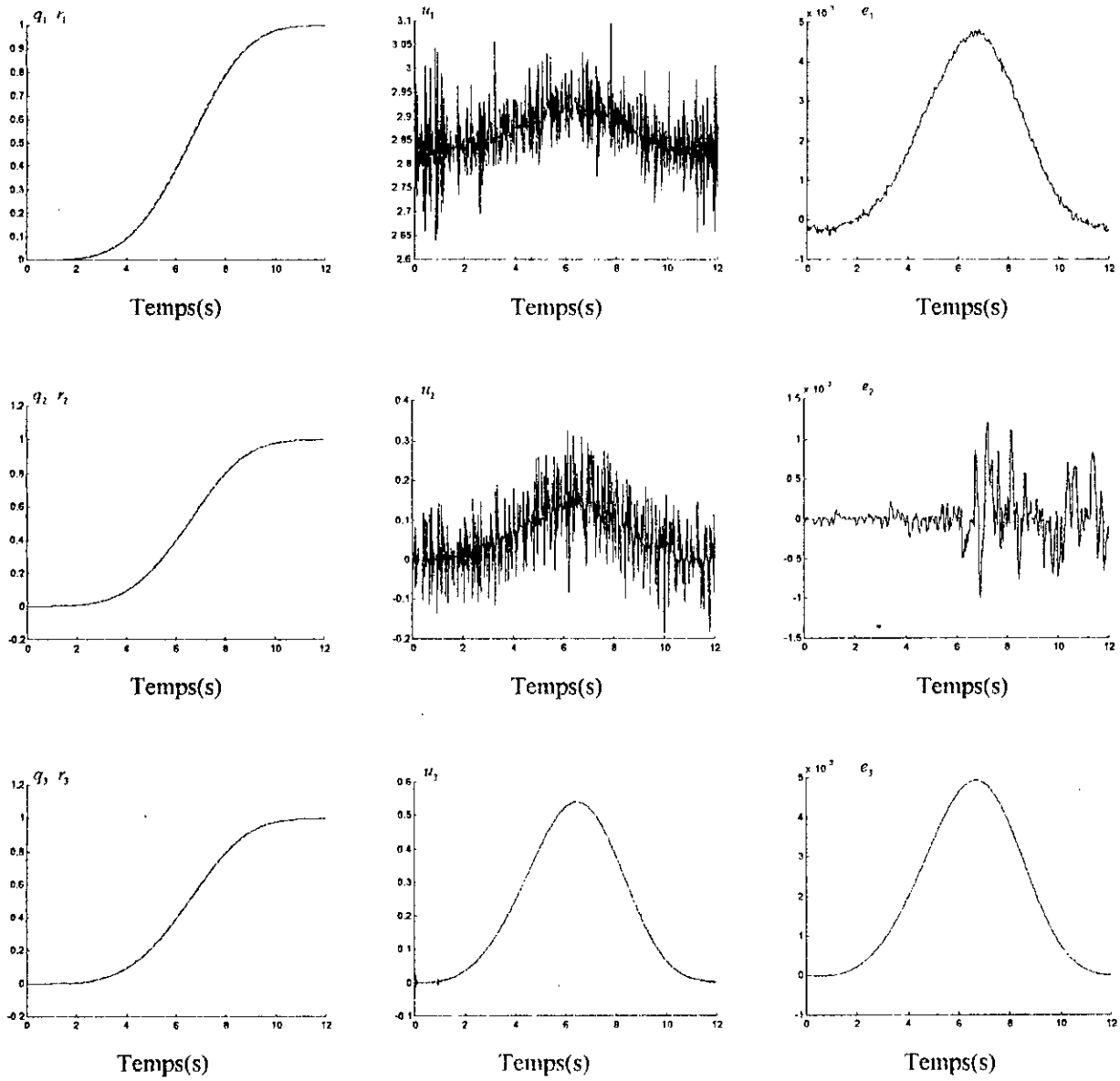
Trois réseaux RBF seront disposés de la même manière que les réseaux utilisés précédemment, de sorte à linéariser le système (fig.VI.14) et les mêmes pôles seront maintenus. Ainsi, un seul réseau ANN1 est remplacé par un réseau RBF à 85 neurones cachés, entraîné d'une manière identique que le réseau LBF. Après introduction des changements dans les paramètres du robot, nous allons constater l'aptitude des réseaux RBF à s'adapter pendant le fonctionnement et suivre la référence.

Nous avons choisi les réseaux RBF à cause de leurs caractéristiques de calcul local et de réadaptation paramétrique, qui est aussi locale; ce qui nous a permis de les ajuster en temps réel sans occasionner des grandes altérations pouvant déstabiliser le système.

Afin de ne pas risquer de perturber le système, les centroïdes, issus de la classification effectuée dans l'espace des états du bras de robot ne sont pas modifiés pendant la réadaptation du contrôleur neural auto-ajustable. C'est les poids synaptiques reliés aux sorties seulement qui sont appelés à se réadapter.

N'ayant pas connaissance sur les paramètres du robot, et donc sur les valeurs des commandes nécessaires à générer, la méthode de Backpropagation ne peut pas être directement utilisée. Nous avons utilisé l'algorithme d'optimisation aléatoire ROM pour cet apprentissage. Notre choix pour cette méthode est motivé par le fait que celle ci optimise un critère sans exiger la connaissance des sorties désirées. De plus, contrairement à la Backpropagation, celle ci utilisée avec une variance de petite valeur évite les oscillations importantes pendant l'apprentissage. Une variance très petite permet, d'autre part, à notre système de distinguer entre les erreurs dues aux changements des entrées et celles dues aux variations des paramètres (cf.VI.2.2.1). Cette méthode a été adaptée à être utilisée en Data Learning, afin de lui permettre une utilisation en temps réel. Ainsi, après chaques deux étapes en Data Learning, les poids donnant un meilleur résultat en sortie sont choisis puis maintenus. La variance du bruit gaussien utilisé est décroissante avec une valeur initiale de 0.01.

Des modifications sont portées sur les paramètres du modèle définissant le fonctionnement du bras (équ.VI.12; VI.13, VI.14).
Dans la figure (IV.18), nous présentons les réponses du système en position, la commande générée par le neuro-contrôleur, et l'erreur à la sortie, pour des modifications de 300 à 400% apportées sur le modèle du robot.

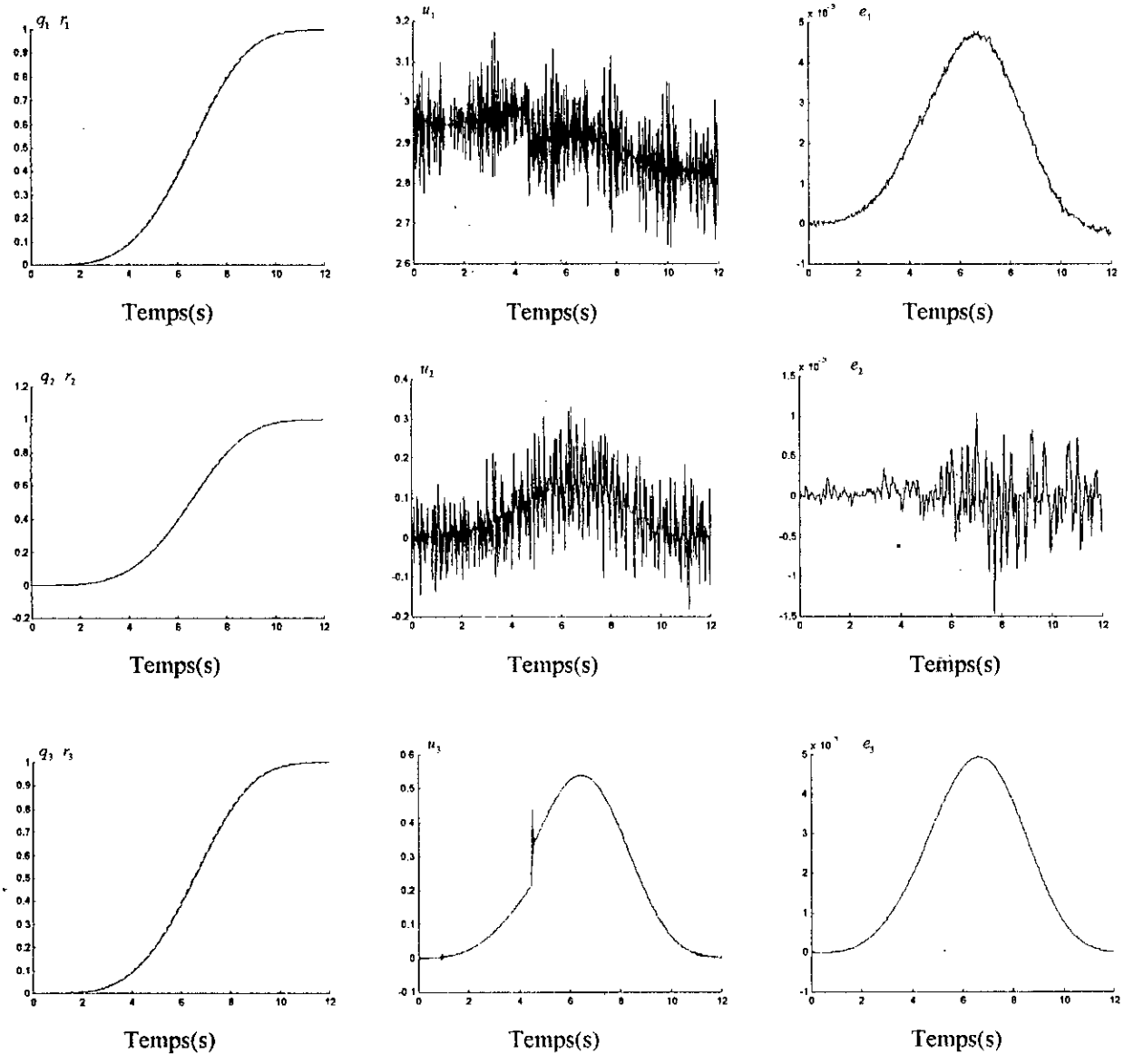


a. Position b. Commande c. Erreur

Fig.VI.18 Réponses des trois articulations en position pour des variations de 300 à 400% des paramètres du bras.
——— Réponse en position du bras manipulateur.
····· Trajectoire désirée.

Nous remarquons sur cette figure que l'erreur augmente au début, pour converger à la fin vers des valeurs très petites. Le contrôleur neural pérforme son apprentissage pendant le fonctionnement du système. Nous remarquons, d'autre part, que la commande présente des distorsions, qui sont très visibles dans le cas des deux premières articulations.

Afin de mieux observer le comportement du système, nous l'avons testé pour des variations brusques survenues au court du fonctionnement au milieu de la trajectoire. La figure (VI.19) montre les réponses des articulations, l'erreur en sorties, ainsi que les commandes générées par le régulateur auto-ajustable. Nous remarquons la réaction brusque de la commande après la survenue des modifications, afin de réduire l'écart en sortie de chaque articulation.



a. Position. b. Commande. c. Erreur.

Fig.VI.18 Réponses des trois articulations en position pour de brusques variations brusques de 300 à 400% des paramètres du bras à *t=4.5 s.*

—— Réponse en position du bras manipulateur.

····· Trajectoirede référence.

Au court de ces simulations, le régulateur neural s'ajuste continuellement. Ceci rend souvent l'allure de la commande fluctuante. Or, parfois l'erreur en sortie est acceptable et ne nécessite pas de réadaptation. Ainsi, afin d'économiser le temps et l'énergie, nous avons opté à appliquer un gel sur la réadaptation du régulateur neural. Ce gel inhibe cette réadaptation à chaque fois que l'erreur en sortie est jugée satisfaisante. La valeur de l'erreur au-delà de laquelle le régulateur doit ajuster ses paramètres est fixé à 0.001.

Nous avons testé le régulateur neural dans ces conditions. La figures (VI.20) présentent les réponses des trois articulations, ainsi que la commande générée et l'erreur en sortie pour les mémes variations opérées sur les paramétres du bras dans le premier cas.

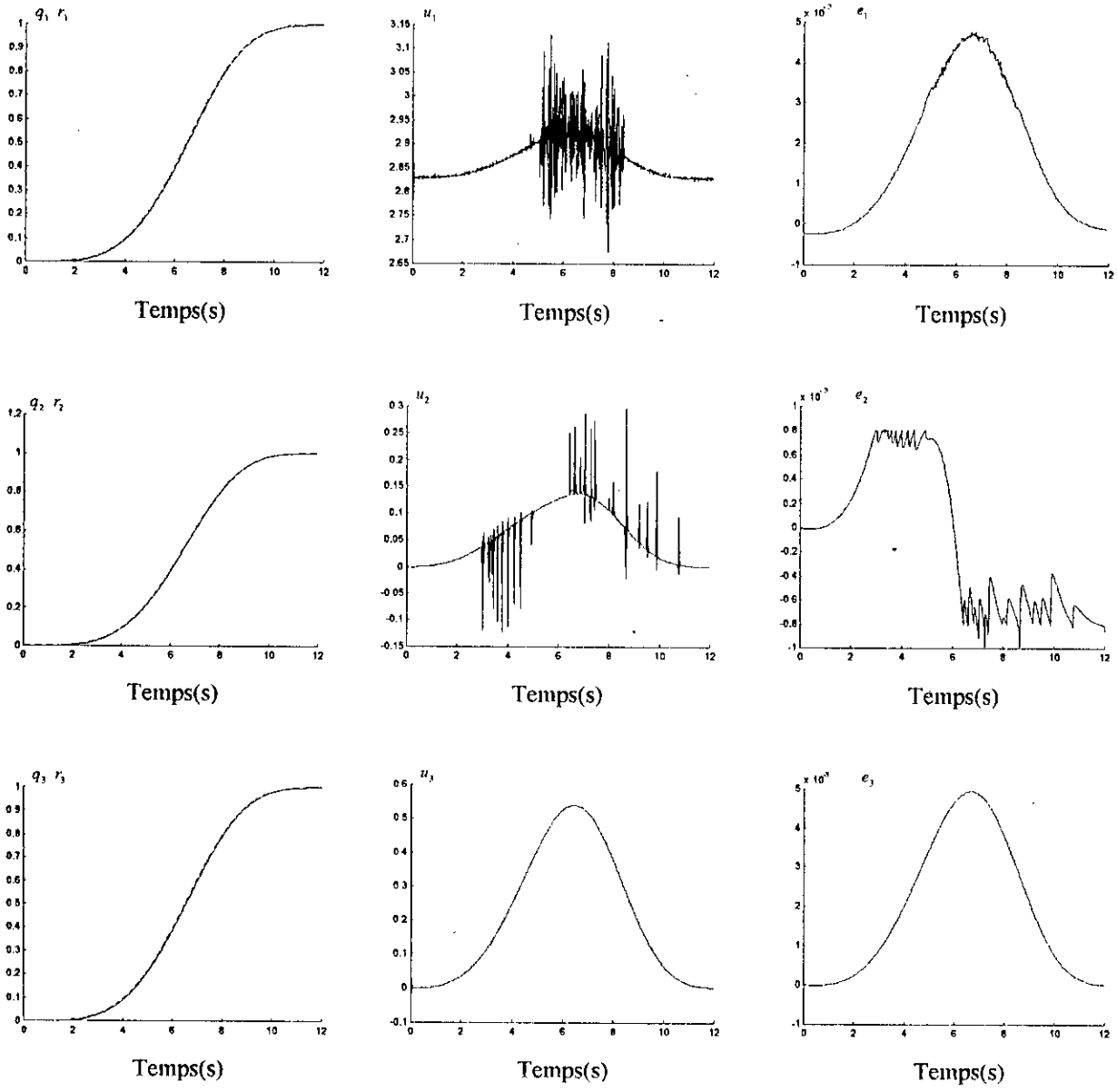


Fig.VI.20 Réponses des trois articulations en position pour des variations de 300 à 400% des paramètres du bras avec application du gel.
——— Réponse en position du bras manipulateur.
····· Trajectoire désirée.

De même, nous avons testé le régulateur pour des variations brusques. La figure (VI.21) montre les résultats obtenus.

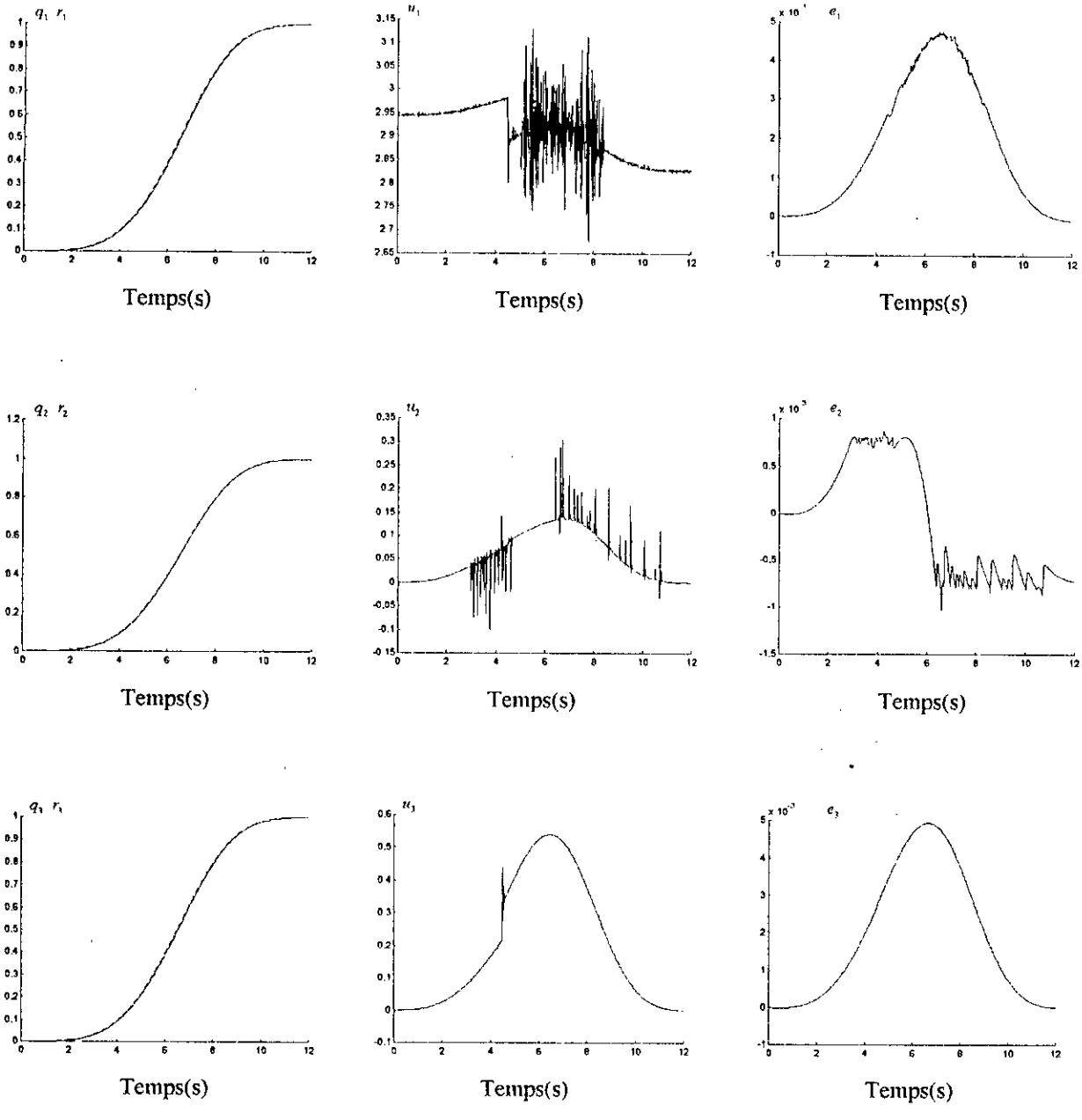


Fig.VI.21 Réponses des trois articulations en position pour des variations brusques de 300 à 400% des paramètres du bras à $t=4.5s$ avec application du gel.

——— Réponse en position du bras manipulateur.

····· Trajectoire de référence

Nous remarquons que le bras suit la trajectoire avec la même précision. L'allure de la commande générée par le neuro-contrôleur présente moins de distorsions. De plus, on remarque que les valeurs de la commande sont, en général, moins importantes que dans le premier cas. De ce point de vue, il est plus intéressant d'utiliser ce régulateur auto-ajustable sous cette dernière forme.

Il est, cependant, important de remarquer que l'erreur en sortie est plus importante lorsque le gel est appliqué. En effet, sans application du gel, les paramètres du réseau sont constamment ajustés, contrairement au second cas, où il y a cumul de l'erreur lors de l'intervention de la réadaptation.

Dans cette application, le neuro-contôleur à base de réseaux RBF Gaussiens est parvenu à suivre les variations des paramètres du système, en effectuant un apprentissage en temps réel. C'est à travers cet apprentissage, que le régulateur s'auto-ajuste adéquatement. La classification des entrées dans sa couche gaussienne, dont nous avons gardé les centroïdes, lui permet d'effectuer l'optimisation dans la couche de sortie plus facilement.

La méthode ROM, utilisée pour cet apprentissage, nous a permis d'éviter le problème du calcul du Jacobien. Cette méthode, appliquée pour un tel entraînement, a été utilisée en Data Learning avec un test par étape. Associée avec une valeur faible et adaptative de la variance, cette utilisation en temps réel a donné des performances satisfaisantes.

VI.4 Commande d'un réacteur chimique CSTR (Continuous Stirred Tank Reactor) par sa température.

Les systèmes chimiques sont connus comme étant souvent fortement non linéaires et surtout dépendant de plusieurs paramètres. En effet en dehors des réactifs et des produits, le déroulement des réactions chimiques reste influer par d'autres paramètres qui gouvernent la dynamique de ces réactions suivant leur nature.
La température est un paramètre dont l'influence est importante. En effet, la chaleur qui est une manifestation de l'énergie à l'intérieur des réacteurs chimiques évolue d'une manière non linéaire suivant les lois de la physique thermodynamique. La non-linéarité et le comportement complexe occasionnés par ce facteur nécessitent un système de commande qui doit s'y adapter. Ainsi, Trouver une loi de commande pour faire face à l'influence de tel paramètre et aux incertitudes de l' évolution que celui ci peut engendrer constitue à lui seul un problème à part dans la commande des réacteurs.

Dans l'étude qui vient nous allons essayer d'établir des techniques de commande basés sur les réseaux de neurones pour le contrôle d'un réacteur chimique vis à vis au problème de la température. Ainsi des commandes neurales utilisant les différents types de réseaux étudiés nous permettront de commander ce système dans tout son intervalle de fonctionnement.

VI.4.1 Modélisation du systeme

A l'intérieur du CSTR (Continuous Stirred Tank), deux composants sont mélangés afin de produire un troisième composant avec une concentration $C_a(t)$ et une température du mélange $T(t)$ (fig.VI.22).

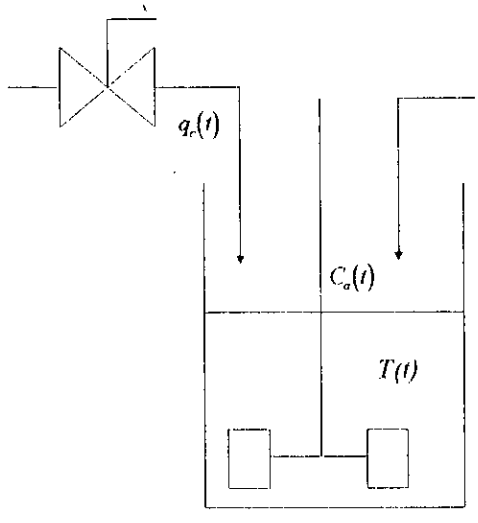


Fig.VI.22 Shéma du CSTR

Le modèle de ce réacteur est donné par J.Morningrid et al. Il est décrit par les équations différentielles non linéaires suivantes [Lig 95]:

$$\dot{C}_a(t) = \frac{q}{v}\left(C_{a0} - C_a(t)\right) - k_0 C_a(t)\exp-\left(E/RT(t)\right) \qquad \text{(VI.18)}$$

$$\dot{T}(t) = \frac{q}{v}\left(T_0 - T(t)\right) + k_1 C_a(t)\exp-\left(E/RT(t)\right) + k_2 q_c(t)\left(1 - \exp-\left(k_3/q_c(t)\right)\right)\left(T_{c0} - T(t)\right)$$

Ou:
C_{a0} représente la concentration de l'alimentation en entrée produite par la réaction.
T_0 représente la température du produit à l'entrée supposée constante et égal à 350 K.
T_{c0} représente la température du refroidissant supposée constante et égal à 350 K.
k_0, E/R, v, k_1, k_2 et k_3 représentent les constantes chimiques et thermodynamiques relatives à ce problème.

Les natures et les données numériques de tous ces paramètres, ainsi que les dimensions du réacteur sont données dans l'annexe B.

Ce réacteur effectue une réaction chimique exothermique. Ce qui veut dire que celle ci dégage de la chaleur pendant le déroulement de la réaction. l'augmentation de la température influe sur la dynamique de la réaction. En effet le dégagement de chaleur ralenti cette réaction et fait diminuer la concentration du produit. En introduisant un fluide refroidissant avec un débit $q_c(t)$ (qui représente la commande à appliquer.), la température peut être modifiée et donc la concentration du produit contrôlée.

G. Lightboby a utilisé les réseaux de neurones à fonction de base linéaire LBF (réseaux à couches statiques) pour une commande avec un modèle de référence linéaire dans un intervalle de ± 0.02 autour d'une concentration d'équilibre de $0.1\ mol/l$. Par la suite il a utilisé les techniques classiques de commande linéaire -à savoir un PID- dans cet intervalle [Lig 95].

Dans ce qui suit, nous utiliserons le modèle du CSTR pour mettre en oeuvre quelques stratégies de commande en utilisant les réseaux **RBF**, les réseaux **LBF** et le **ART2,** afin de commander le système dans tout son intervalle de fonctionnement.

VI.4.2 Etude du système en boucle ouverte

Afin de bien se saisir du problème, nous proposons d'étudier son comportement en boucle ouverte avant d'entamer la commande.

Nous avons échantillonné le système avec un pas d'échantillonnage $\Delta t = 0.01 s$. Ainsi les équations décrivant son fonctionnement deviennent:

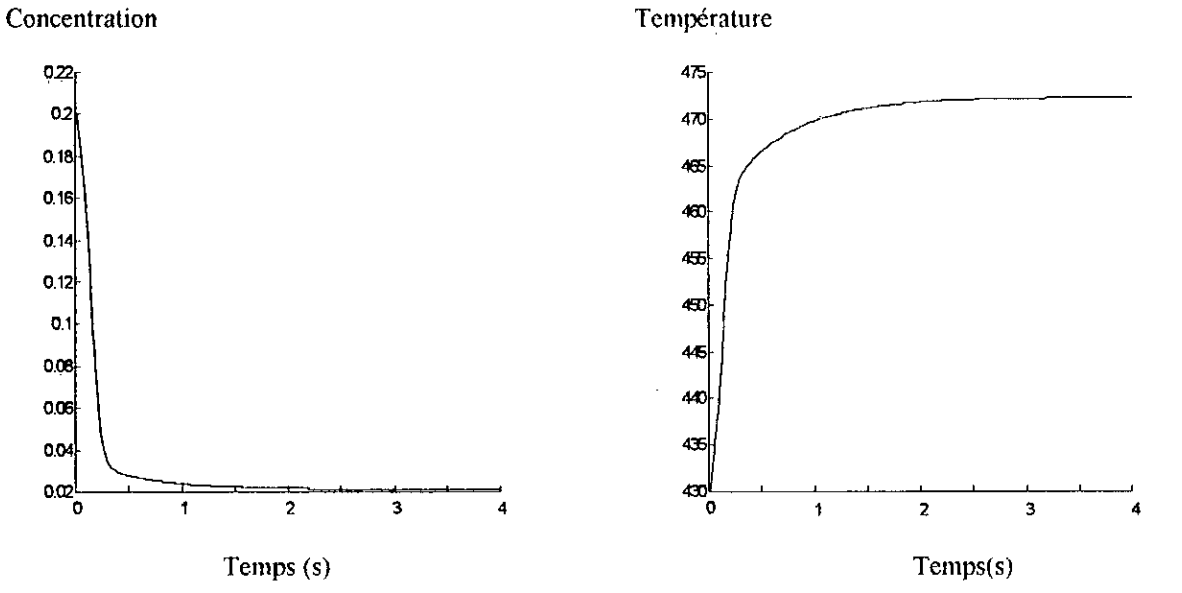
$$C_a(k+1) = C_a(k) + \Delta t\left[\frac{q}{v}\left(C_{a0} - C_a(k)\right) - k_0 C_a(k)\exp-\left(E/RT(k)\right)\right] \qquad \text{(VI.19)}$$

$$T(k+1) = T(k) + \Delta t\left[\frac{q}{v}\left(T_0 - T(k)\right) + k_1 C_a(k)\exp-\left(E/RT(k)\right) + k_2 q_c(k)\left(1-\exp-\left(k_3/q_c(k)\right)\right)\left(T_{c0} - T(k)\right)\right.$$

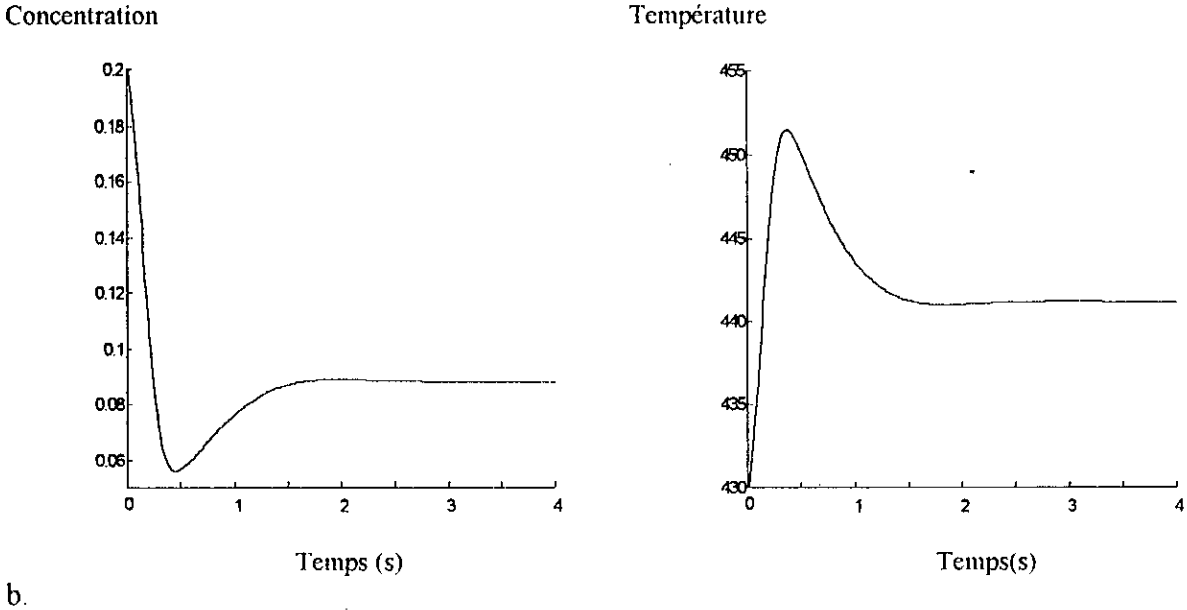
A travers le système d'équations (VI.19), nous remarquons que dans cette réaction une augmentation de concentration provoque une augmentation de la température. Cette augmentation de la température est naturellement compensée par une diminution de la concentration (equ. VI.19). En effet suivant les lois de la chimie thermodynamique, cette augmentation favorise le déroulement de la réaction dans le sens opposé. Ainsi, c'est ces deux faits contradictoires qui font le problème posé ce système.

Nous avons effectué une série de simulations afin de faire ressortir les caractéristiques essentielles relatives au comportement du système.

Pour les mémés états initiaux $C_a(0)= 0.2$ et $T(0)=450$, nous avons testé le système avec des flux en entrée qui sont différents, q_c=90 puis q_c=120. On constate sur la figure ci-dessous (fig.VI.23) que le système est stable. L'allure des sorties est, par ailleur, différentes.



a.



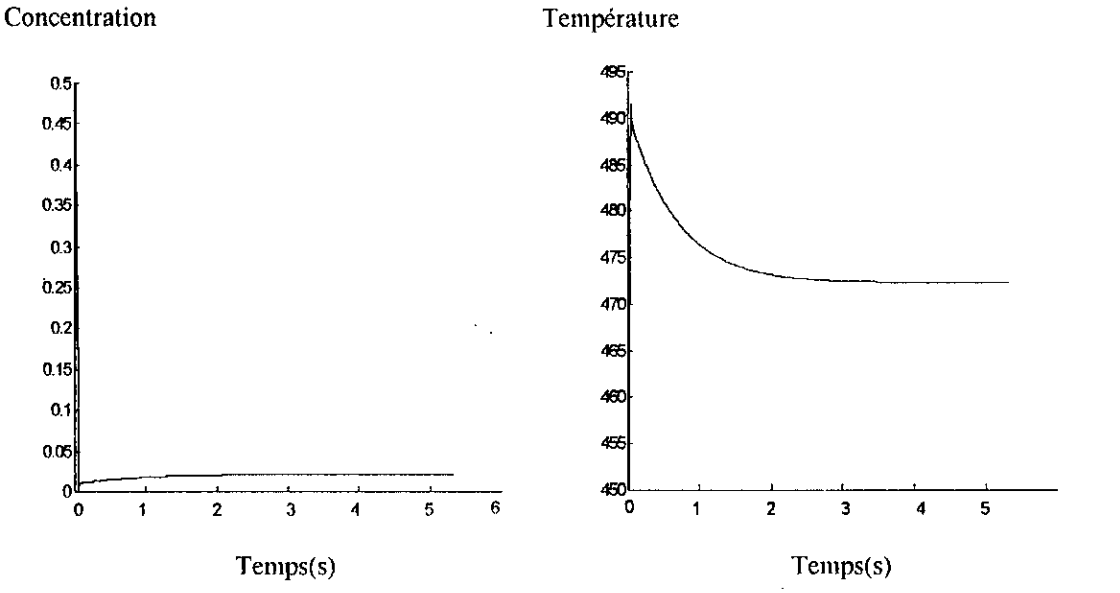
b.

Fig. VI.23. Réponse du système pour des conditions initiales identiques avec des flux de commande différents.

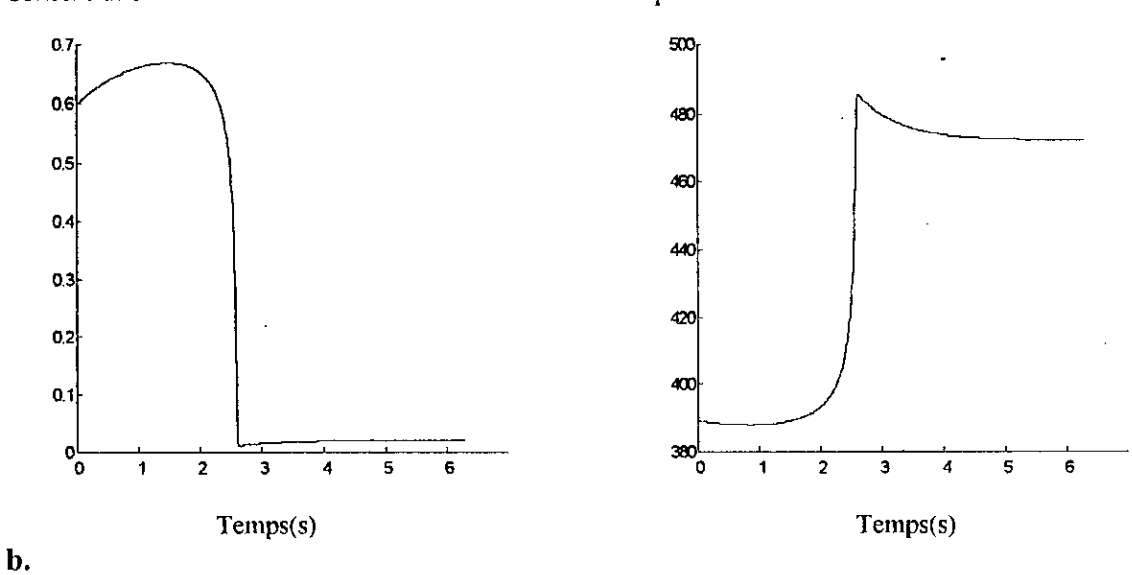
(a) $C_a(0)= 0.2$ $T(0)=430$ q_c=90.
(b) $C_a(0)= 0.2$ $T(0)=430$ q_c=120

L'état du système dépend sensiblement des conditions initiales. Ainsi pour une même entrée (flux du fluide refroidissant), le système se comporte différemment si les conditions initiales sont différentes et à l'équilibre celui ci atteint des états différents.
La figure (VI.24) montre les réponses du système pour un même flux en entrée de *60 l/min.* et des conditions initiales différentes.

a.



a.

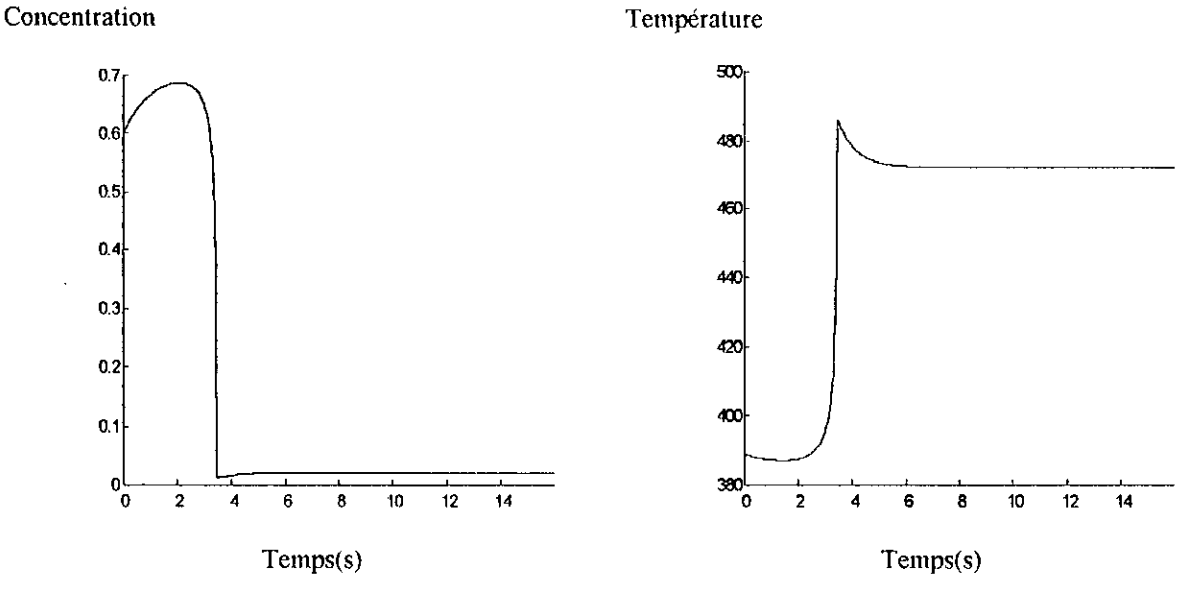


b.

Fig.VI.24 Réponse du système pour un même flux q_c et les conditions initiales différentes
(a) $C_a(0)= 0.5$ $T(0)=450$ q_c=60
(b) $C_a(0)= 0.6$ $T(0)=390$ q_c=60

Dans la figure ci-dessous (fig. VI.25), nous remarquons que pour un même flux de commande appliqué en entrée la réponse du système peut être très différente si les conditions initiales sont légèrement modifiées. L' effet, interactif entre température et concentration est trés sensible. Ainsi pour de tel situations un flux de commande important et adéquat est nécéssaire.

a.



b.

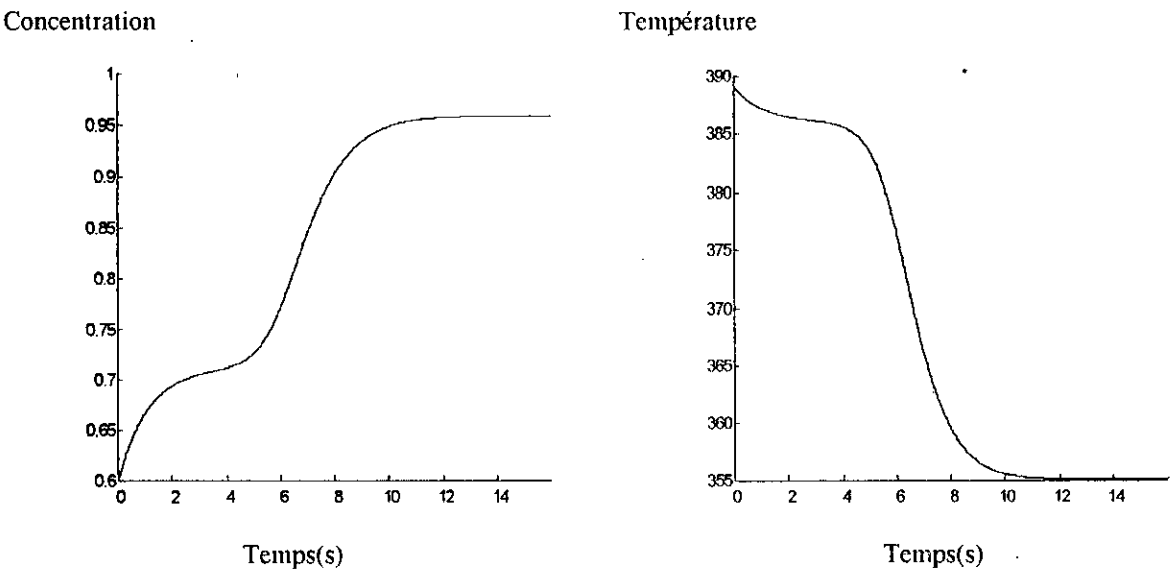


Fig. VI.25 comportement du système pour un léger changement de la concentration initiale
(a) $C_a(0)= 0.6\ \ T(0)=389\ \ q_c=60.$
(b) $C_a(0)= 0.61\ \ T(0)=389\ \ q_c=60.$

Ainsi, nous constatons que ce système dépend sensiblement des conditions initiales. Son comportement diffère, en effet, pour une même entrée suivant les conditions initiales. Pour une même valeur de commande à l'entrée, on aboutit à des sorties différentes à l'équilibre si les conditions initiales sont légèrement différentes. Il faut néanmoins noter que le réacteur est stable dans l'intervalle de fonctionnement comme l'indiquent les courbes.

VI.4.3 Commande du Système avec Compensation de l'effet de la Température

Sachant qu'à l'équilibre à chaque température correspond une concentration, nous avons opté à contrôler la température afin de maîtriser l'évolution du réacteur.

En reprenant le système d'équations (VI.19) nous constatons que celui ci est de la forme:

$$T(k+1) = f\left(T(k), C_a(k)\right) + g\left(T(k), u(k)\right)u(k)$$

Dans cette équation, qui a beaucoup de similitude avec l'équation (VI.3), que nous avons étudié dans le cadre de la commande auto-ajustable (cf.VI.1.2.1), Nous constatons l'existence d'un terme qui traduit l'interaction entre température et concentration. Ainsi nous avons opté à une stratégie visant à compenser l'effet de cette interaction. Pour cela, les réseaux du type RBF seront utilisés.

Pour cela, nous avons généré des exemples entrées-sorties en annulant l'effet de la température sur la concentration dans le système défini dans l'équation (VI.19). Ceci donne pour le CSTR un comportement d'un système de premier ordre.
Les sorties désirées d'un tel modéle peuvent aussi étre obtenues à partir de l'équation (VI.19):

$$T(k+1) = T(k) - \Delta t\left(T(k) - T_{ref}(k)\right)$$

Notre objectif etant de découpler le systeme et de le ramener à la concentration désirée, deux réseaux gaussiens sont utilisés. Le premier, noté RBF1, a pour rôle de compenser l'effet de la concentration sur la température. Il est à deux entrées dont l'une est récurrente, une couche cachée Gaussienne de 52 neurones et une seule sortie. Le deuxième réseau, noté RBF2, a pour rôle d'agir, afin de ramener le système vers la concentration désirée. Il a deux entrées récurrentes, une couche cachée Gaussienne de 47 neurones et une sortie. Ces réseaux sont reliés à un élément linéaire qui effectue la somme de leurs sorties pour l'injecter au système. La figure (VI.26) montre le schéma complet du système commandé.

Les deux réseaux sont entraînés avec des concentrations instantanées et des concentrations de référence dans l'intervalle 0 à 0.3. Le coefficient de dispersion des gaussiennes σ est fixé à 0.4 pour tous les neurones cachés du premier réseau (RBF1) et 0.5 pour le second réseau(RBF2).

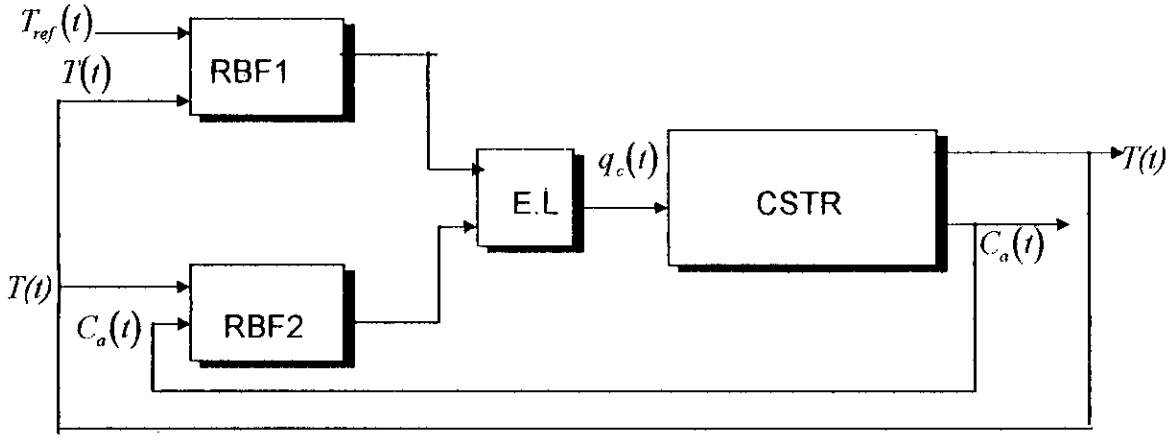


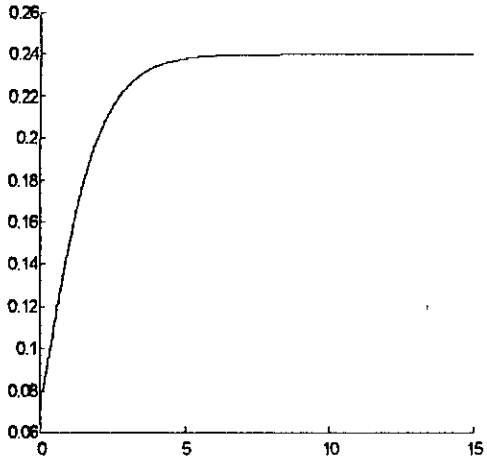
Fig VI.26 Schéma de commande du CSTR

Au court de l'entraînement des deux réseaux, le critère à optimiser est l'erreur entre la concentration désirée et celle obtenue. Les réseaux doivent délivrer la commande q_c inconnue qui permettra de minimiser cet écart. Nous avons utilisé, pour la détermination des paramétres des réseaux RBF, la méthode du gradient. Cette méthode nécessite le calcul du Jacobien du système. Pour cela, nous avons utilisé le modèle neural LBF identificateur du système que nous avons établi en (V.3). A partir de la sortie du réseau émulateur, nous avons "rétro-propagé" l'erreur à travers le calcul des dérivées partielles jusqu'à la sortie des réseaux contrôleurs RBF (cf.VI.1.3).

Nous avons effectué sur le système plusieurs simulations dans cette zone de fonctionnement. Des concentrations de références avec différentes conditions initiales sont injectées au réseau. Le neuro-contrôlleur arrive à ramener le système vers la concentration désirée avec une bonne précision.

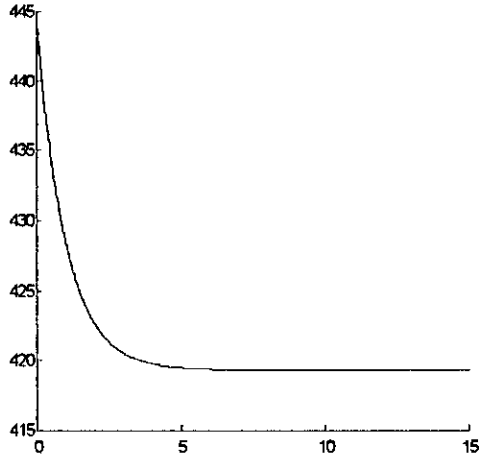
La figure (VI.27) montre la réponse du système pour une référence $C_{ref} = 0.24$, et des états initiaux $C_a(0) = 0.08$; $T(0) = 444$. Nous remarquons l'augmentation du flux refroidissant au début afin de faire baisser la température.

Concentration



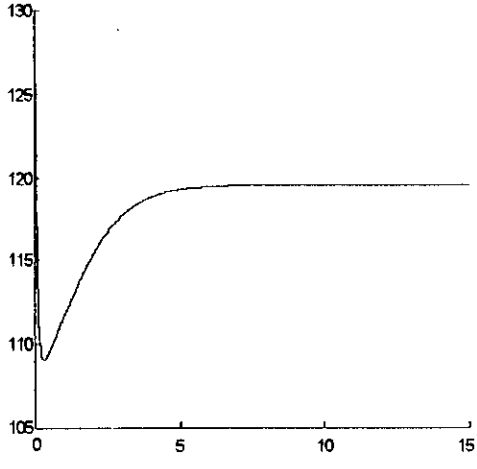
Temps(s)

Température



Temps(s)

Commande

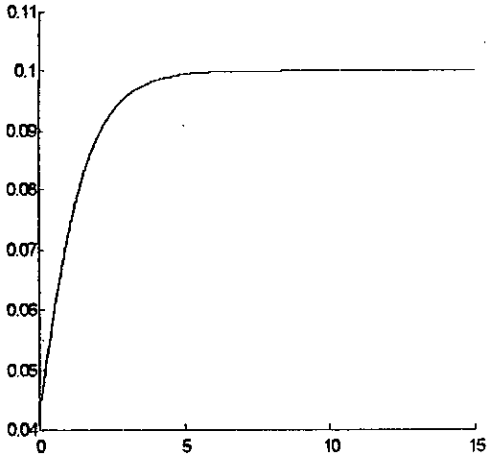


Temps(s)

Fig.VI.27 La réponse du système pour $C_{ref} = 0.24;\ C_a(0) = 0.08;\ T(0) = 444$

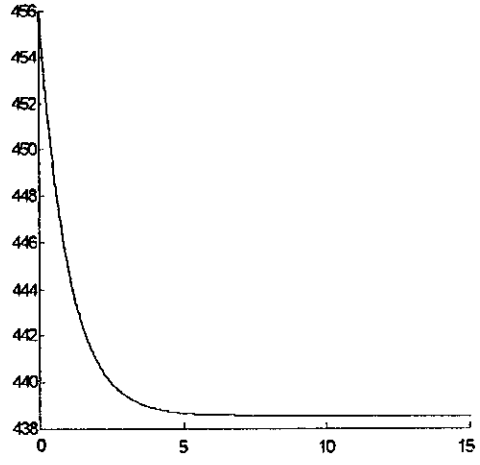
Nous remarquons sur la figure (VI.28) que pour des concentrations initiales basses par rapport à la référence, la commande agit sur la température, qui est élevée, afin de la faire baisser et permettre ainsi à la concentration d'augmenter.

Concentration



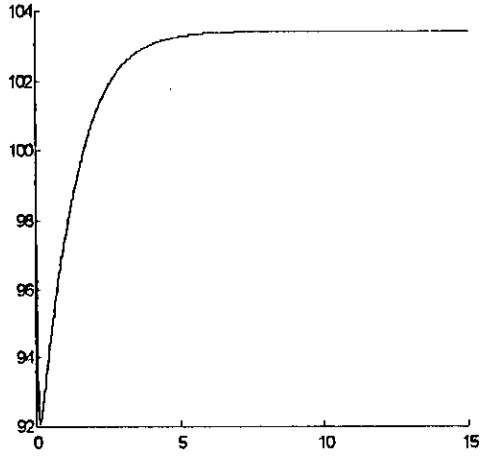
Temps(s)

Température



Temps(s)

Commande



Temps(s)

Fig VI.28.Réponse du système pour C_{ref}=0.1; $C_a(0)$=0.045 *T(0)*=456.

Nous avons testé le régulateur sur sa capacité de faire baisser la concentration. Nous povons voir sur la figure (VI.29) le passage de la concentration de $C_a(0)$=0.2 à C_{ref}=0.04. On remarque l'effet de la commande qui fait augmenter la température favorisant le déroulement de la réaction dans le sens opposé afin de faire diminuer le produit.

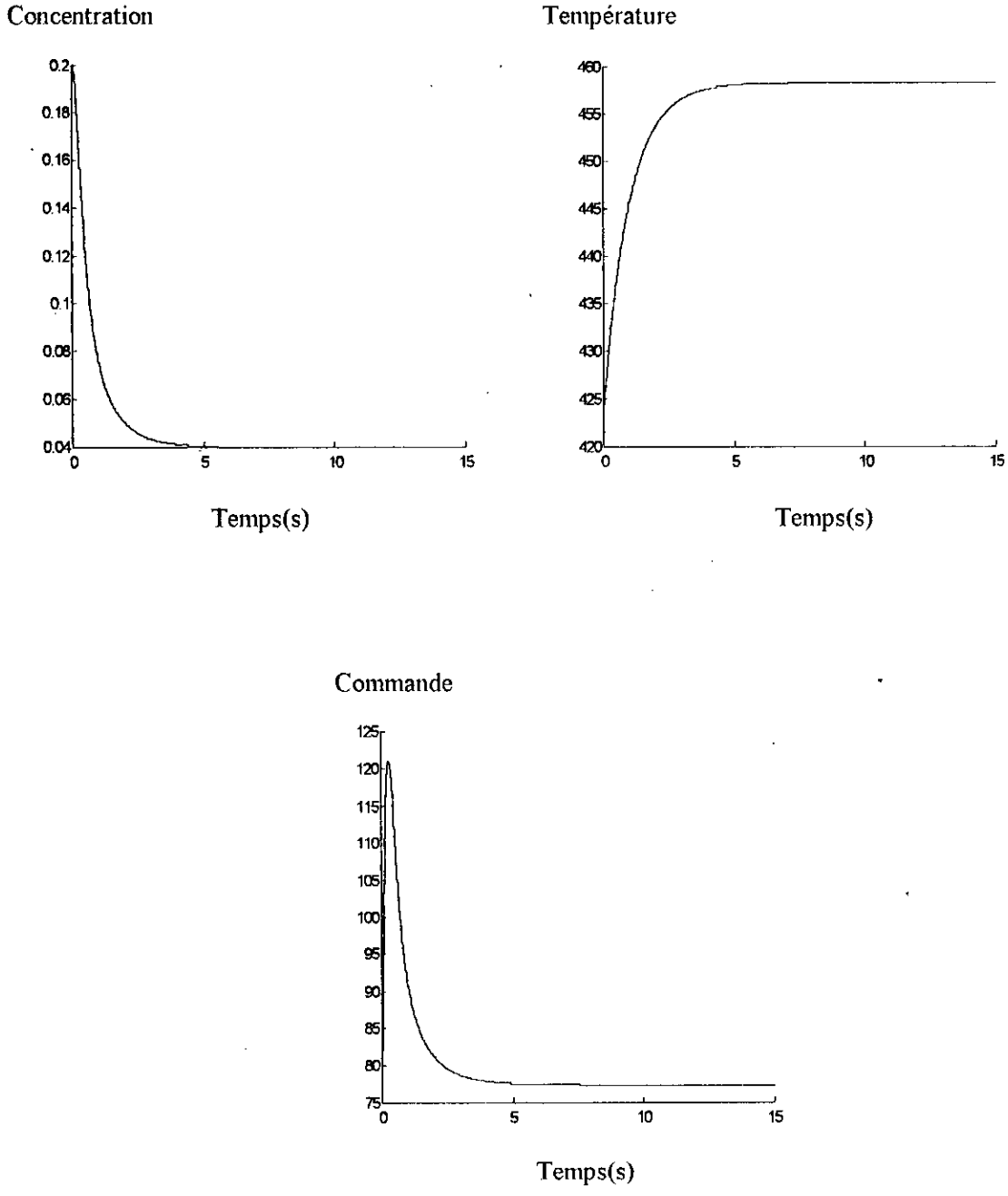


Fig.VI.29 réponse du système pour C_{ref} =0.04; $C_a(0)$ =0.2 T(0)=424.

Afin de voir le comportement du contrôleur, lorsque les conditions initiales sont à l'extérieur de la région d'entraînement du neurô-controleur, nous avons effectué une simulation pour $C_{ref}=0.24$ avec $C_a(0)=0.4$. Nous remarquons que le réseau génère une commande qui ramène le système vers la concentration désirée (fig VI.30).

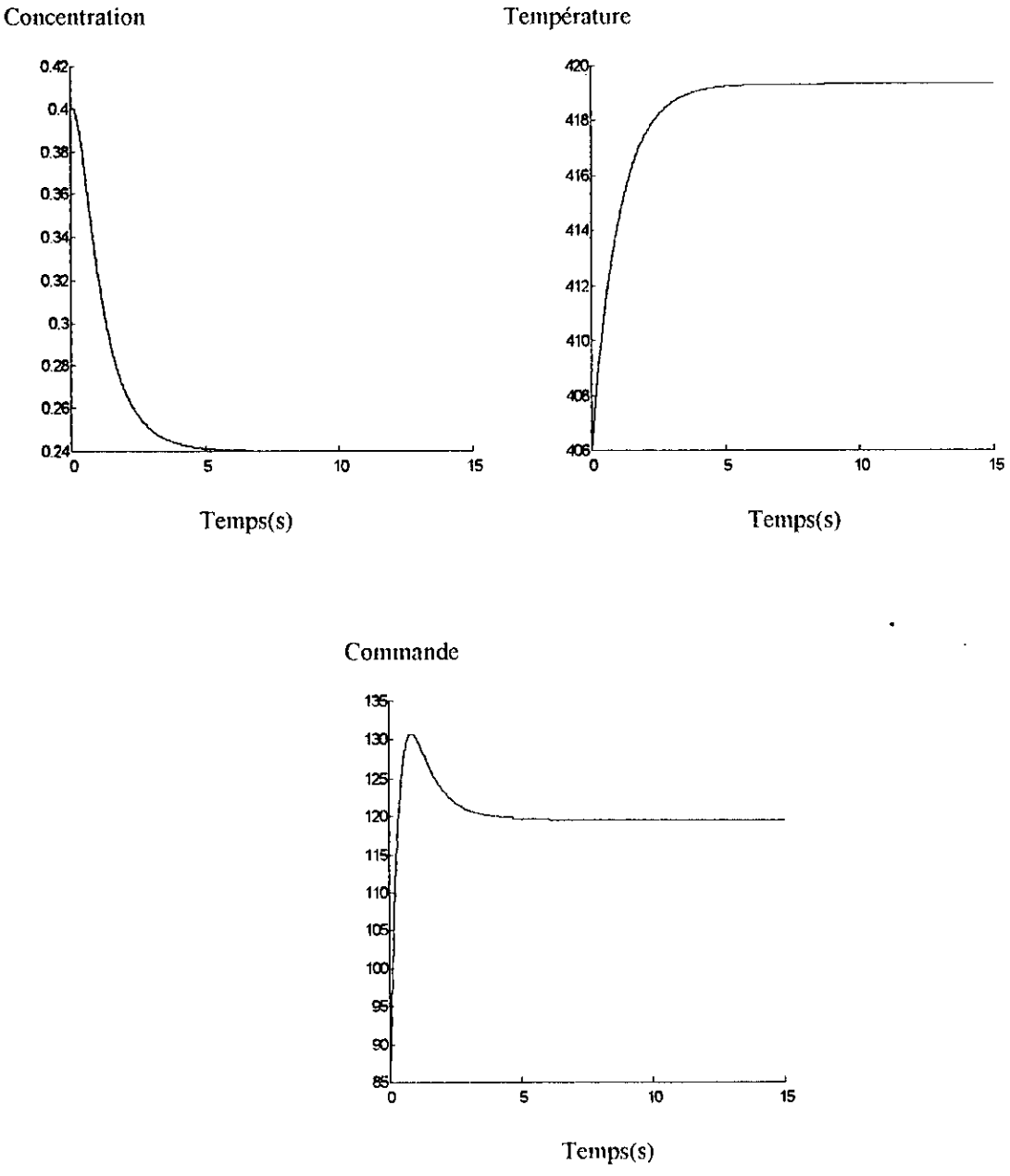


Fig.VI.30 Réponse du système pour $C_{ref}=0.24$; $C_a(0)=0.4$ $T(0)=406$

Nous avons testé le comportement du réseau vis à vis aux perturbations, que nous avons présentées comme une augmentation ou diminution brusque de la température ou de la concentration.
Nous remarquons que le réseau arrive à compenser la perturbation et ramener le système à la concentration désirée (Fig.VI.31).

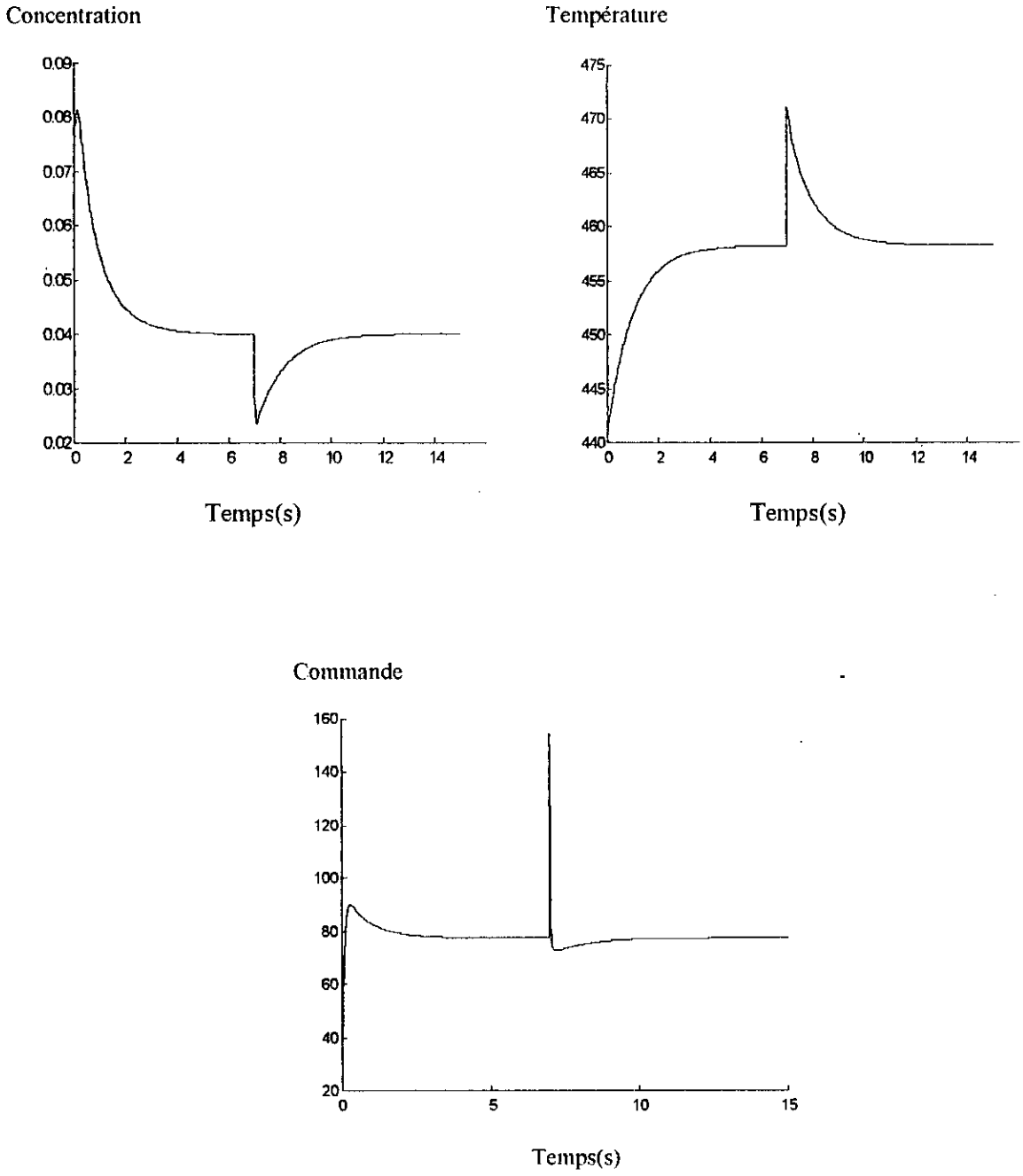


Fig.VI.31. Réponse du système et l'action de commande pour $C_{ref}=0.04$; C_a (0)=0.07; $T(0)=442$ avec une perturbation sur la température à t=7s

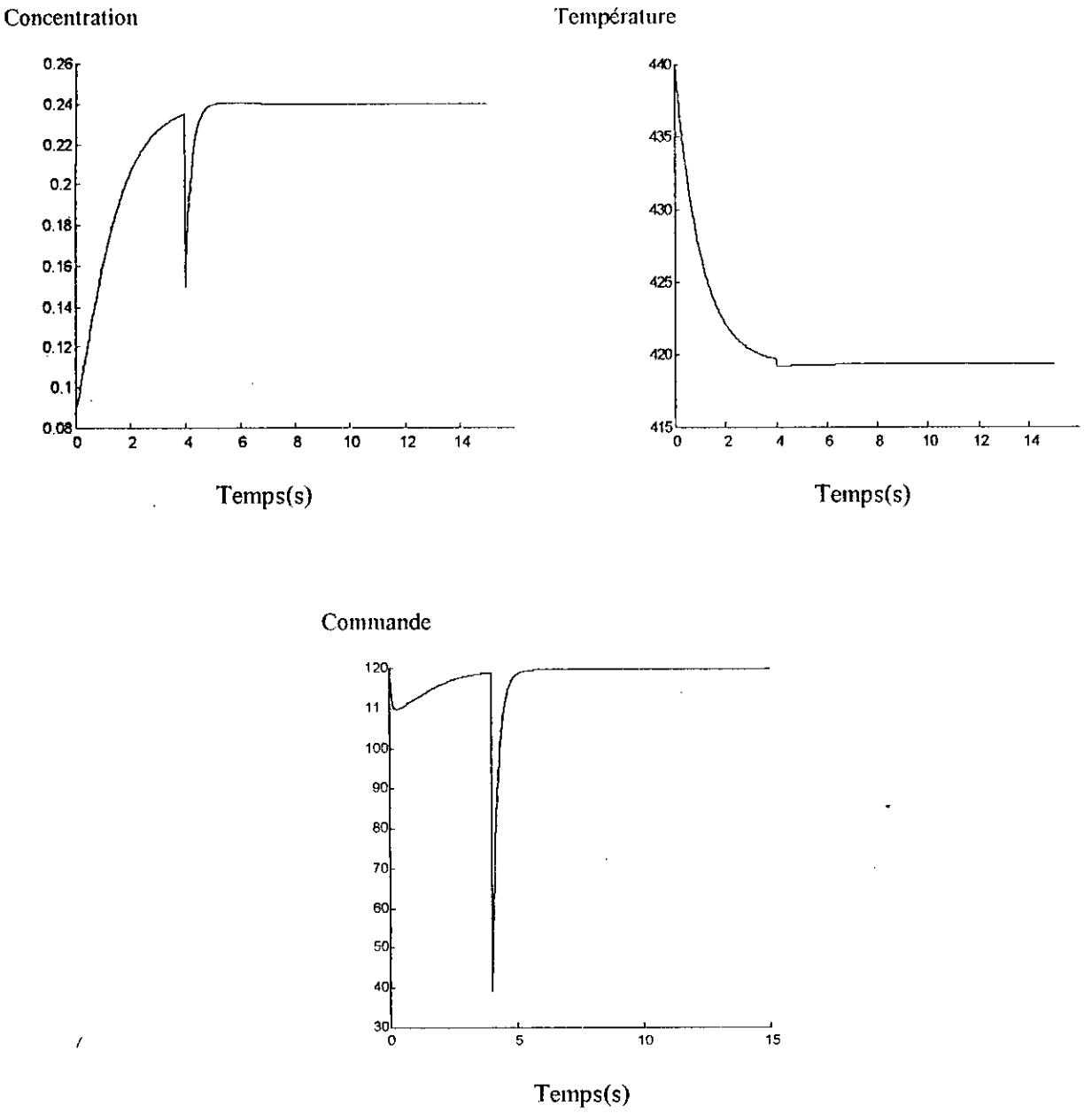


Fig. VI.32. Réponse du système et action de commande pour $C_{ref}=0.24$; C_a (0)=0.09; $T(0)=440$ avec perturbation sur la concentration à $t=4.2\ s$.

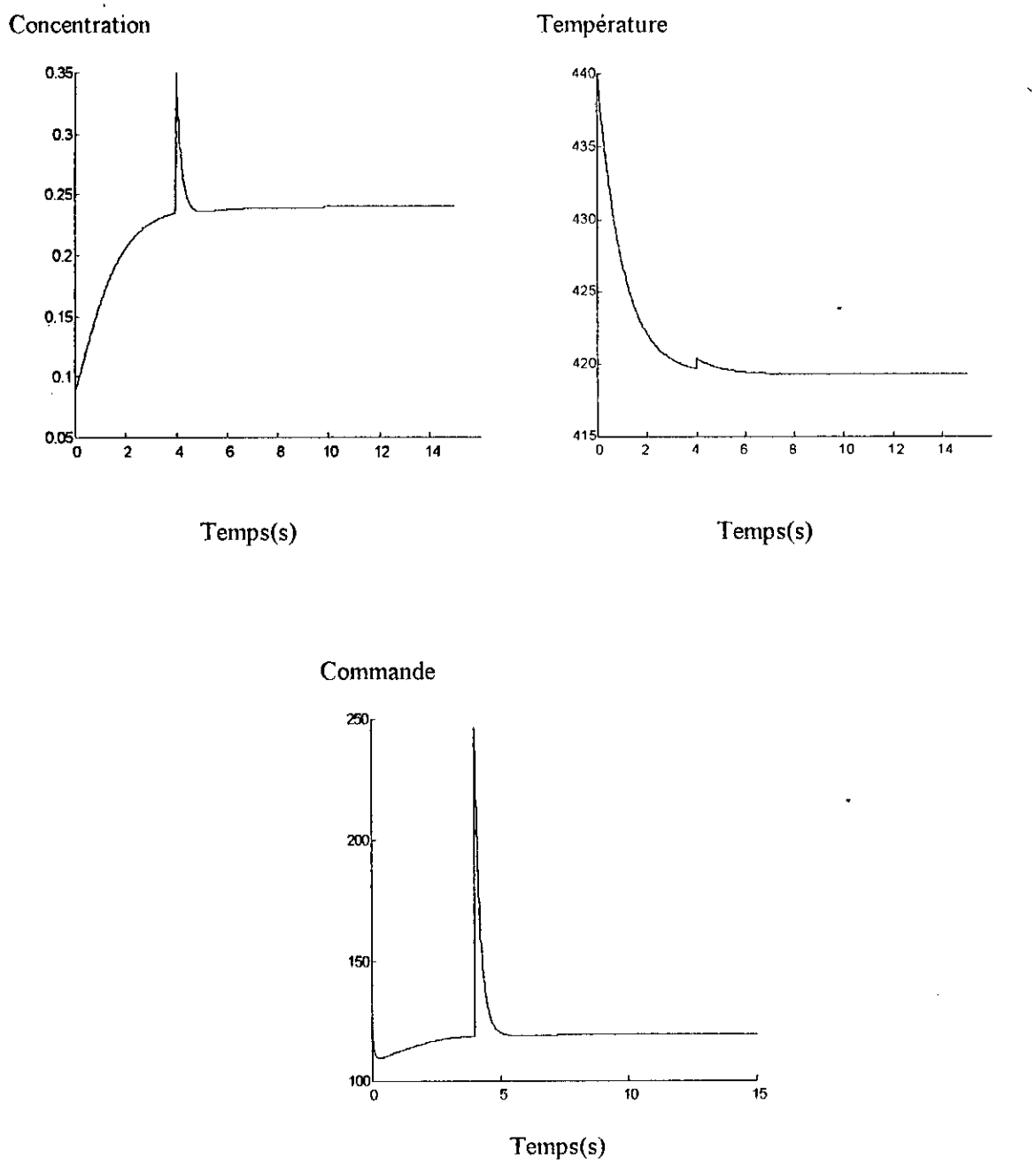


Fig. VI.33 Réponse du système et action de commande pour $C_{ref}=0.24$; $C(0)=0.09$; $T(0)=440$ avec perturbation sur la concentration à $t=4.2s$.

Nous constatons que le contrôleur, à base des réseaux RBF, a montré de bonnes performances, notamment en généralisation.

D'autre part, comme nous l'avons constaté dans notre étude en boucle ouverte, Ce système est très dépendant des conditions initiales. Ainsi, il est impossible de commander le système dans toute sa zone de fonctionnement *[0,1]* avec le même contrôleur RBF ayant les mêmes poids synaptiques et les mémés centres des Gaussiennes dans tout cet étendu.
Ceci nous a poussé à réfléchir à élaborer une commande nous permettant de conduire ce système suivant la région où il se trouve dans tout son étendu de fonctionnement.

VI.4.4 Commande avec Classification par Régions de Fonctionnement

Dans cette partie nous avons élaboré une technique permettant de commander le système pour les différentes valeurs de concentration de *0* à *1*. Cette technique permet de basculer dans la loi de commande suivant la région où le système se trouve.
Pour cela, suivant les capacités du contrôleur neural, la zone de fonctionnement du système est partagée en trois classes.
Un réseau **ART2** ***(Artificial Resonnace Theory)***, à deux entrées et trois sorties, est d'abord utilisé. Ce réseau a été entraîné avec les concentrations instantanées par lesquelles le système passe. Le réseau doit à chaque instant classifier le couple constitué par les valeurs de concentration et celles de référence. L'entraînement est commencé avec une seule classe, dont les poids correspondent à la première concentration avec une référence de même classe. Le coefficient de vigilance ρ est fixé à 0.17; ceci, afin de permettre la formation de trois classes distinctes.
L'entraînement est arrêté lorsqu' on remarque la formation de classes dont les contours ne changent plus notablement pendant le fonctionnement du ART2 (cf.II.3.2.1).

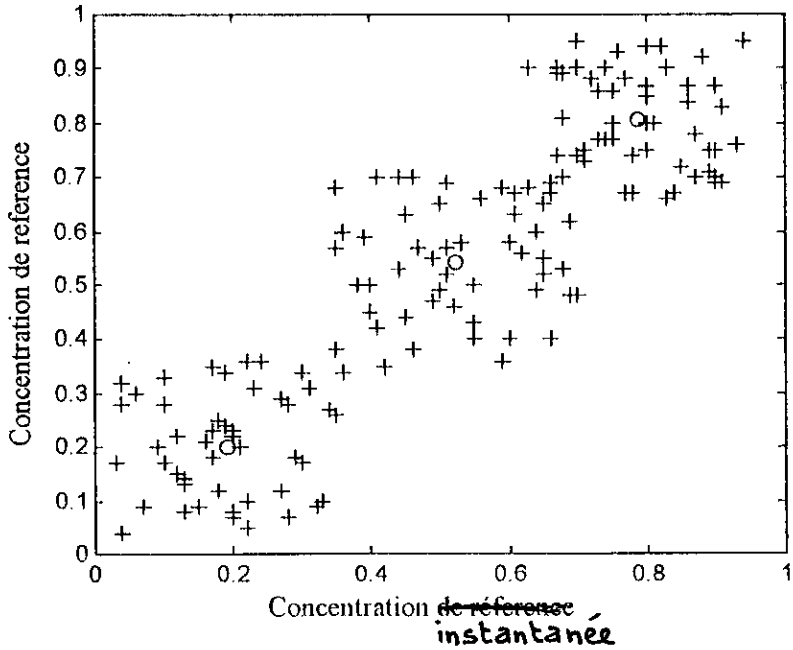


Fig. VI.34 Le regroupement *(clustering)* constitué par le ART2
+ Point de fonctionnement du système.
o Les centroïdes (poids synaptiques du réseau).

Nous remarquons, sur la figure (VI.34), que trois nuages de concentration de données, constituant les trois classes, se sont formés. Les centroides de ces régions, constituant les poids synaptiques à lesquels le réseau ART2 a abouti, sont montrés sur la même figure. Nous remarquons trois regroupements, respectivement autour de 0.22, 0.54 et 0.83. Les intervalles des concentrations gravitent donc autour des trois centroïdes (fig.VI.34).

Ainsi trois neuro-contrôleurs, identiques à celui utilisé en (VI.3.38), sont entraînés dans chacune de ces régions. Pour leur entraînement, les modèles RBF utilisés pour l'identification du système dans chacune des trois classes sont utilisés. Les réseaux RBF1 RBF2 contrôleurs obtenus pour chacune des deux dernières classes sont respectivement à 40 et 43 neurones cachés pour ANN2 et 30 et 33 pour ANN3.
La structure de cette stratégie de commande est montrée sur la figure (VI.35).

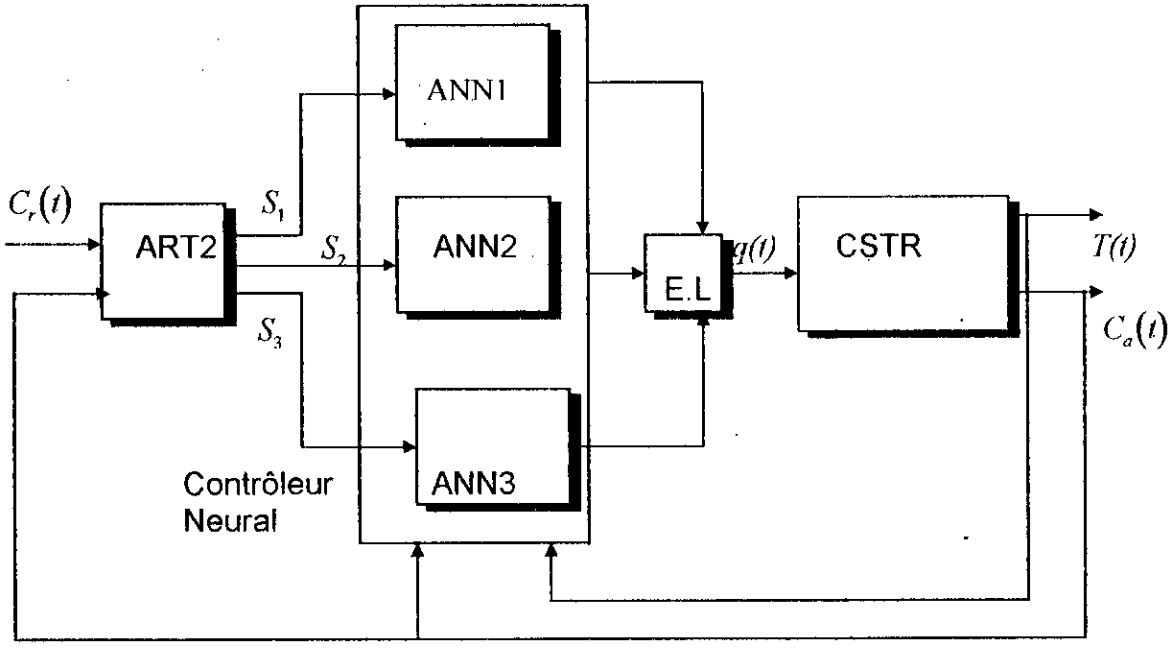
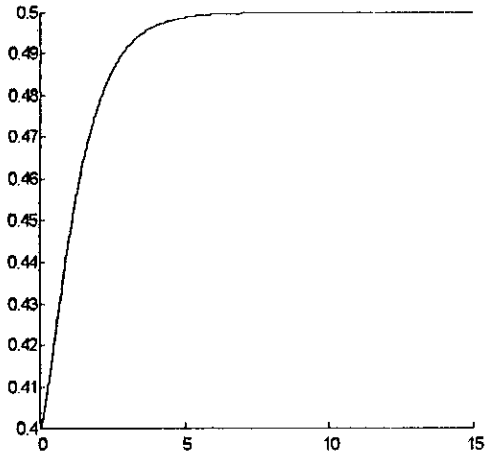


Fig.VI.35 Structure de la commande du système sur toute sa région de fonctionnement.

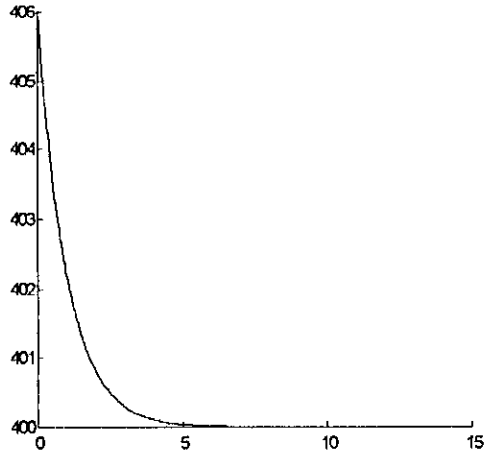
A chaque instant, les sorties du système sont classées par le ART2 et le neuro-contrôleur adéquat est actionné suivant les sorties du ART2. Ce réseau est le seul qui reçoit une entrée active de la part du ART2.
Nous avons testé le régulateur sur des références relevant de chacune des trois régions. Les résultats sont sur les figures (VI.36), (VI.37) et (VI.38).

Concentration



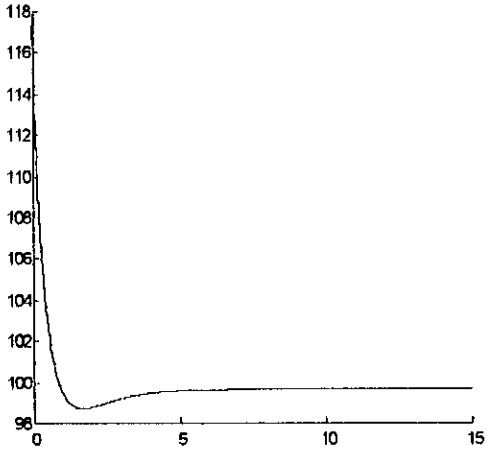
Temps(s)

Température



Temps(s)

Commande



Temps(s)

Fig.VI.36 Réponse du système pour C_{ref}=0.5; $C_a(0)$=0.4 *T(0)*=406.
Fonctionnement dans la seconde classe.

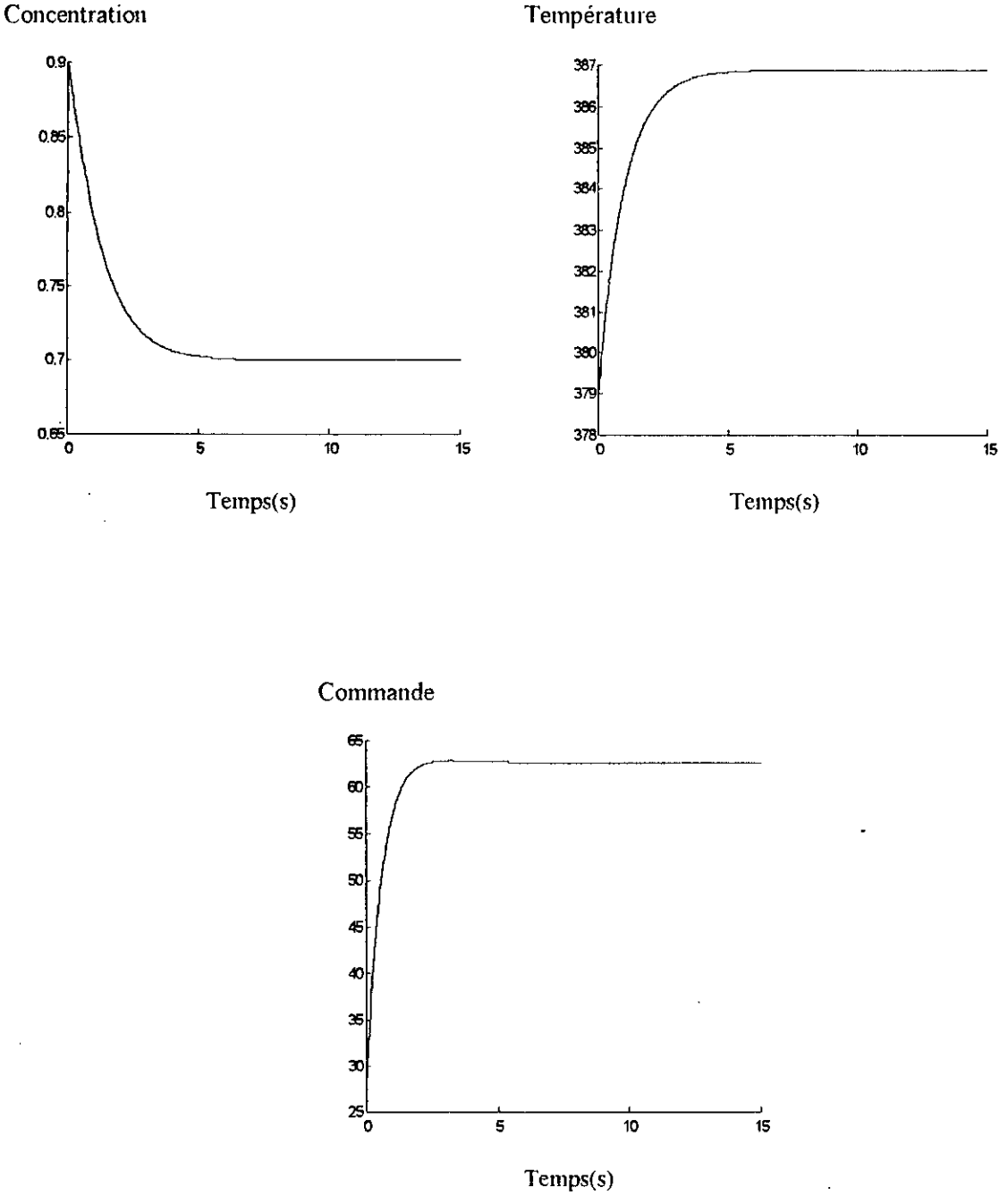
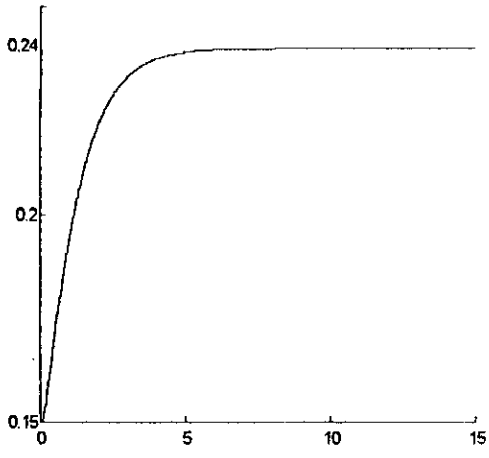


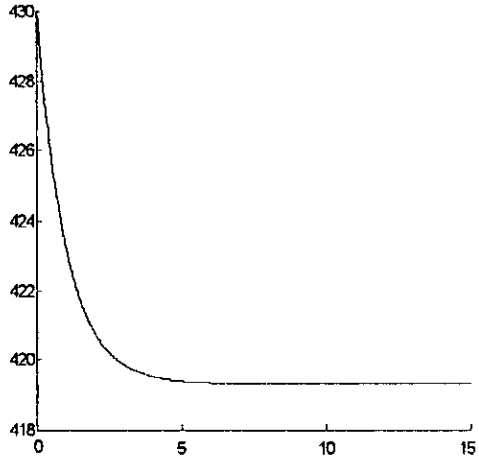
Fig.VI.37 Réponse du système pour C_{ref}=0.7; $C_a(0)$=0.9; *T(0)*=379.
Fonctionnement dans troisième classe.

Concentration



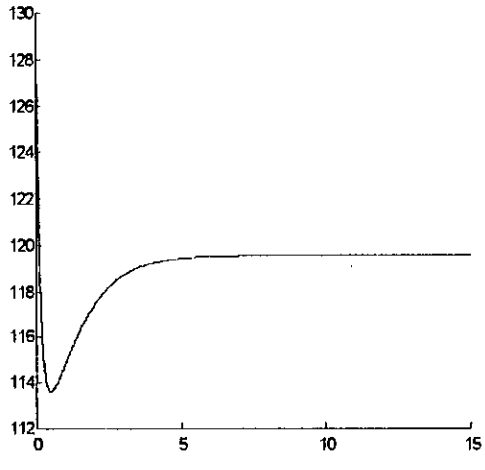
Temps(s)

Température



Temps(s)

Commande



Temps(s)

Fig.VI.38 Réponse du système pour $C_{ref}=0.24;\ C_a(0)=0.15;\ T(0)=430.$
Fonctionnement dans la première classe.

Nous remarquons sur les courbes ci dessus la capacité du régulateur à ramener la sortie à la valeur désirée sur les trois régions couverte par le ART2.

En effet le réseau ART2 permet de générer l' action de commande nécessaire selon l'état de ces sorties S_1, S_2, S_3, qui dépendent des concentrations instantanées et des références. Dans chacun de ces cas, le ART2 détecte la région où le système évolue, et actionne le réseau régulateur adéquat.

Afin de mieux constater les capacités de ce neuro-contôleur, Nous avons testé le système pour des références et des états initiaux éloignés entre eux. Ceci nous permettra de voir les performances du ART2 et son aptitude à pouvoir conduire la concentration de sa valeur initiale à la valeur désirée en parcourant les différentes classes.

Dans la figure (VI.39.a), nous présentons l'évolution du système pour une concentration initiale $C_a(0)=0.1$ et une référence $C_{ref}=0.8$. Il est physiquement impossible de générer une seul commande, qui peut conduire le système directement vers cette concentration. Ainsi, le réseau ART2 doit gérer la "contribution" de chacun des trois régulateurs, pour amener la concentration à sa valeur finale désirée. Le réseau effectue la classification des états instantanée du système, à chaque instant, et assure le basculement entre les trois régions au moment adéquat, suivant l'évolution de la concentration. La figure (VI.39.b) montre les sorties du ART2, qui gouvernent l'intervention de chaque régulateur, ainsi que la commande générée à la sortie du neuro-contrôleur.
La figure (VI.40.a.) Montre la réponse du système, pour une référence $C_{ref}=0.6$ avec une concentration initiale $C_a(0)=0.04$. Nous remarquons les instants de "passation " entre les deux régulateurs neuraux, ANN1 et ANN2, suivant les sorties S_1, S_2, S_3 du ART2, sur la figure (VI.40.b.)

Nous avons testé le régulateur vis à vis à de brusques augmentations de températures causées par des perturbations. Ceci, afin de voir la capacité du réseau ART2 à agir contre les perturbations.
Nous remarquons sur la figure (VI.41), lors de la chute de la concentration, le passage à la deuxième classe et l'action du régulateur ART2 sur le régulateur ANN2. Sur la figure (VI.42), l'augmentation de la température a causé le retour du système vers la première région de fonctionnement. Pour palier ce problème, le réseau ANN1 est actionné, afin de ramener la concentration vers la seconde classe. Nous remarquons lors de ce passage l'activation du réseau ANN2 et l'inhibition de ANN1 par le ART2 (fig.VI.42.b.).

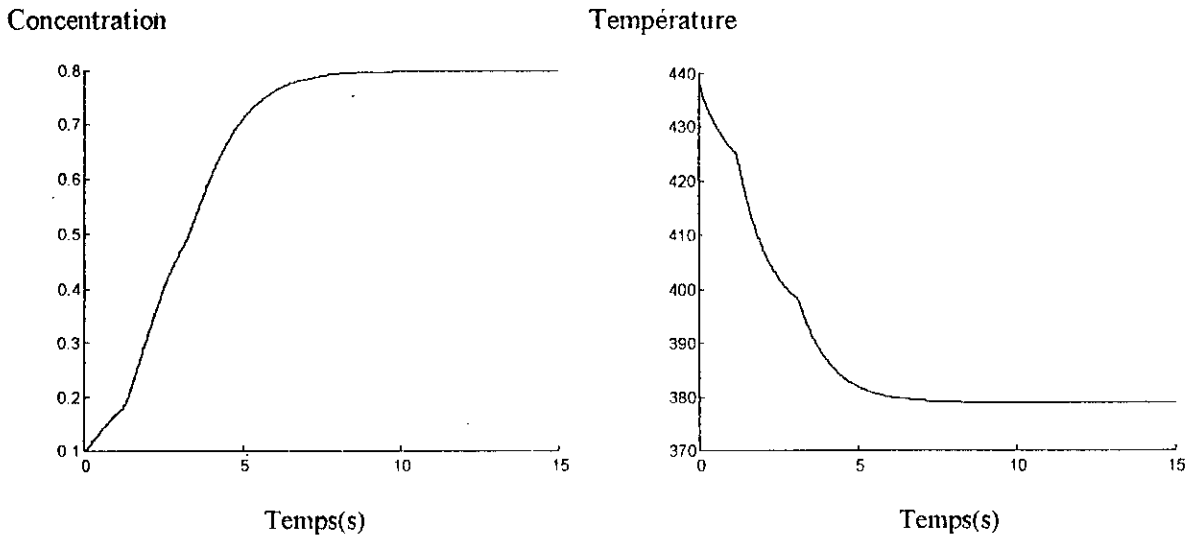


Fig.VI.39. a. La réponse du système pour $C_{ref}=0.8;\ C_a(0)=0.1;\ T(0)=438$

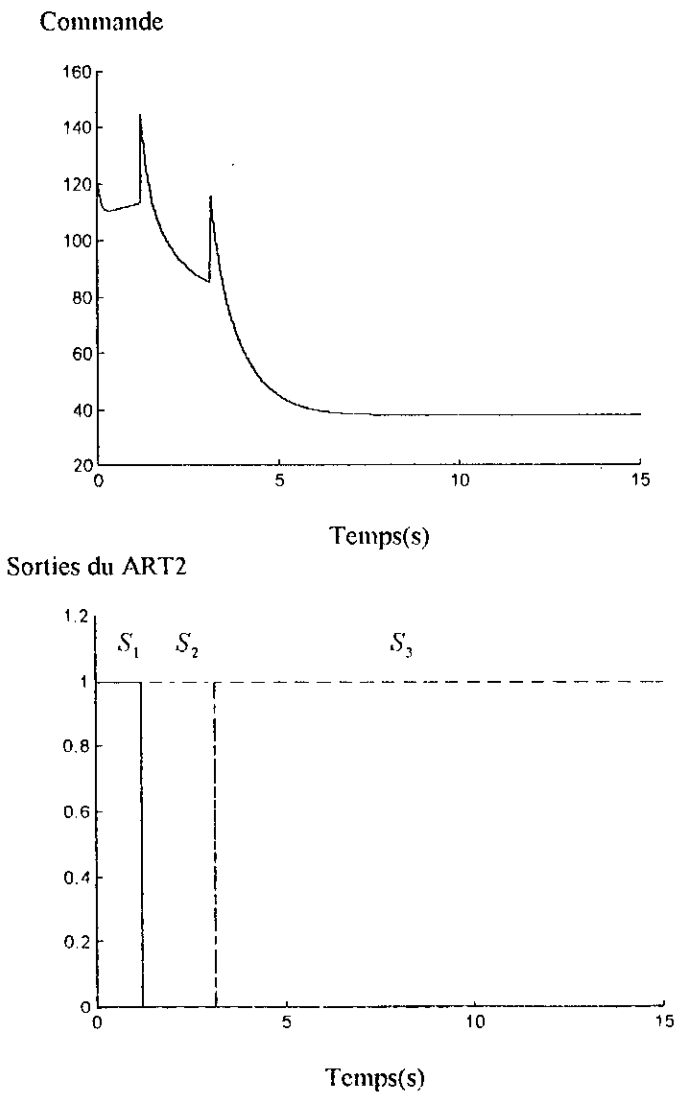


Fig.VI.39.b. Les sorties du ART2 et la commande générée par le neuro-contrôleur

——— Sortie S_1

– · – · – Sortie S_2

····· Sortie S_3

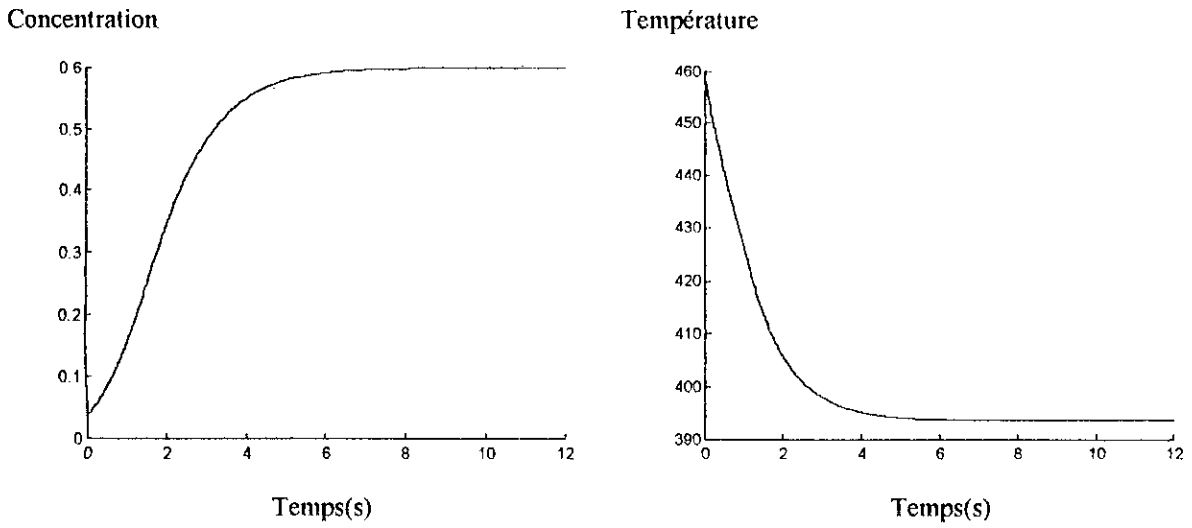


Fig.VI.40. a. La réponse du système pour $C_{ref}=0.6;\ C_a(0)=0.04;\ T(0)=457$

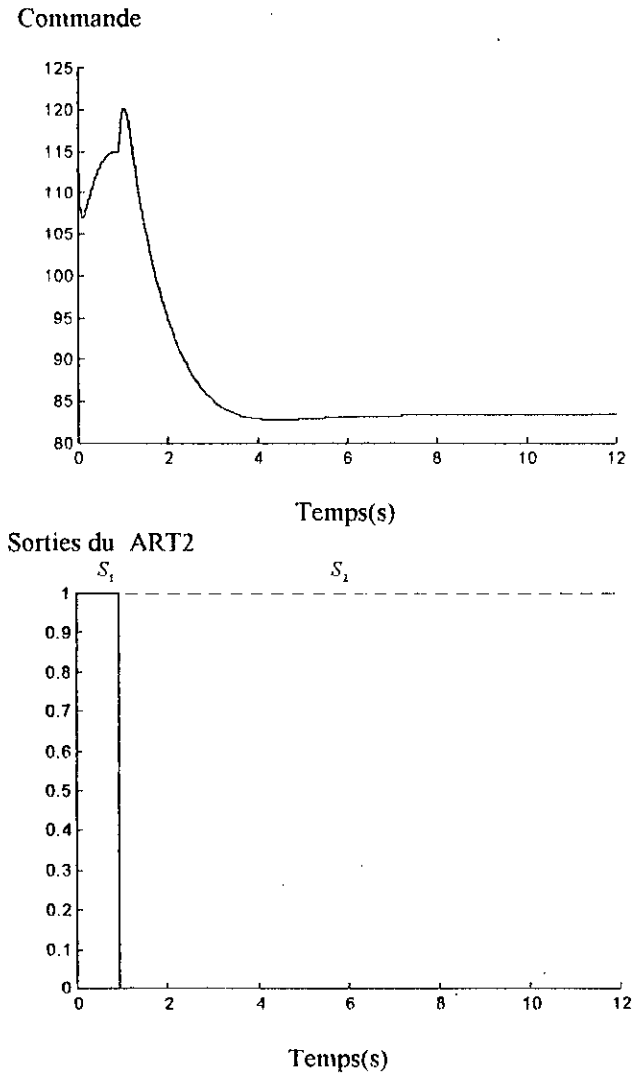


Fig.VI.40.b. Les sorties du ART2 et la commande générée par le neuro-contôleur

——— Sortie S_1

····· Sortie S_2

— · — · — Sortie S_3

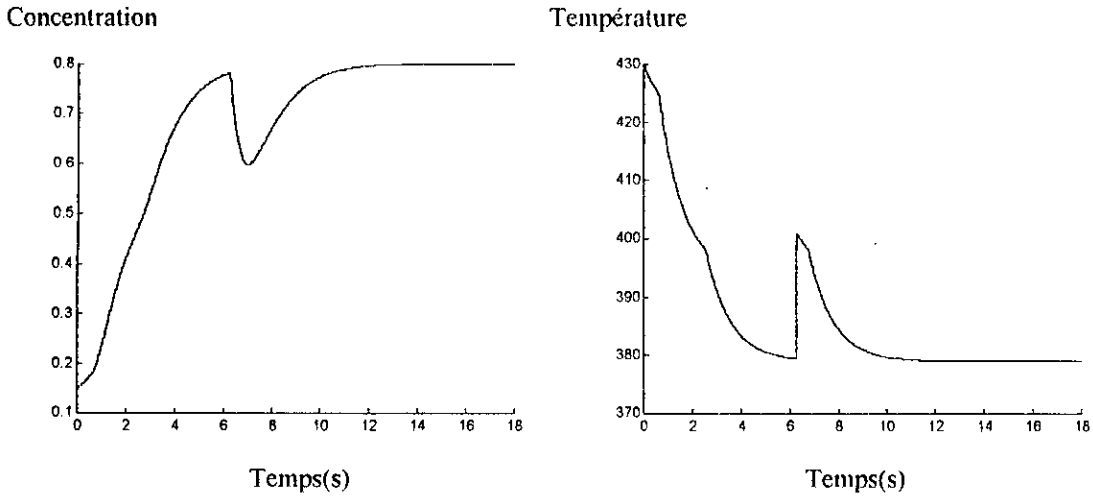


Fig.VI.41.a. Les réponses du système à une augmentation brusque de la température à $t=6s$. pour $C_{ref}=0.8$; $C_a(0)=015$; $T(0)=430$.

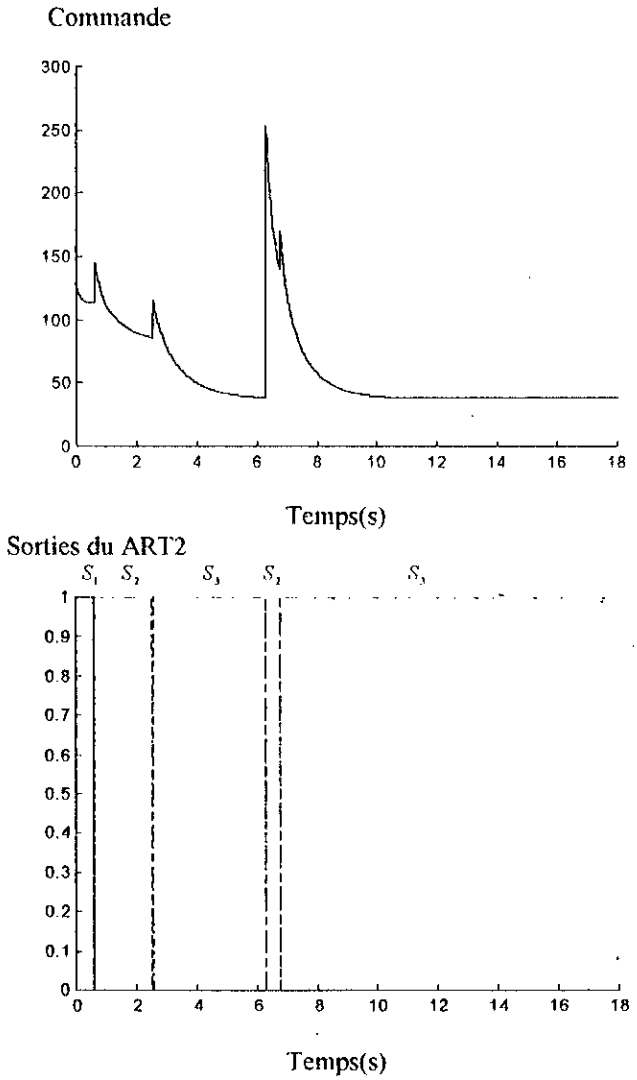


Fig.VI.41.b. Les sorties du ART2 et la commande générée par le neuro-contôleur.

——— Sortie S_1

– – – – Sortie S_2

····· Sortie S_3

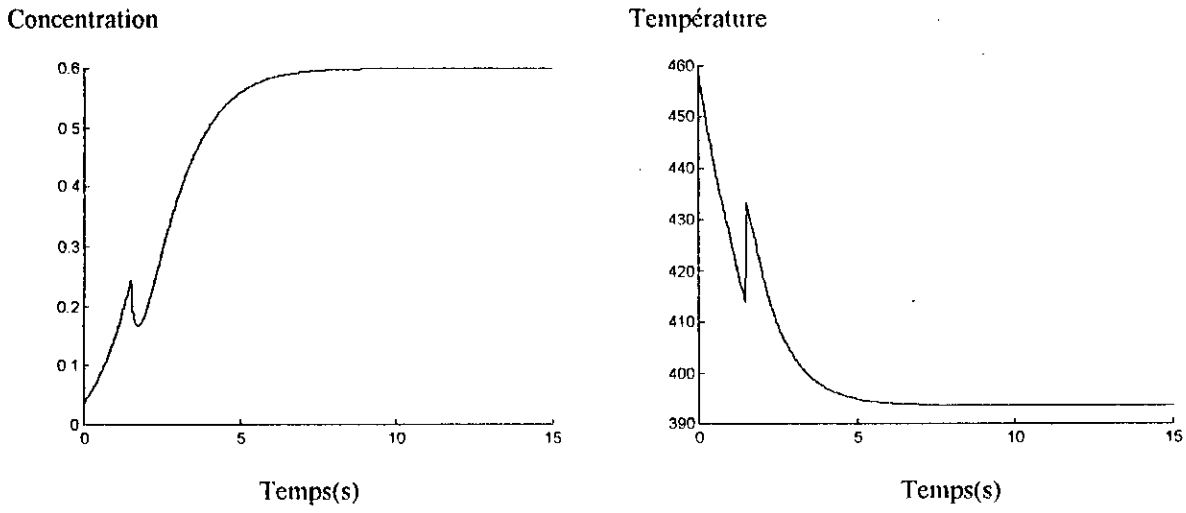


Fig.VI.42.a Les réponses du système pour $C_{ref}=0.6;\ C_a(0)=0.04\ T(0)=458$ avec introduction d'une augmentation brusque de la température à $t=1.5s$.

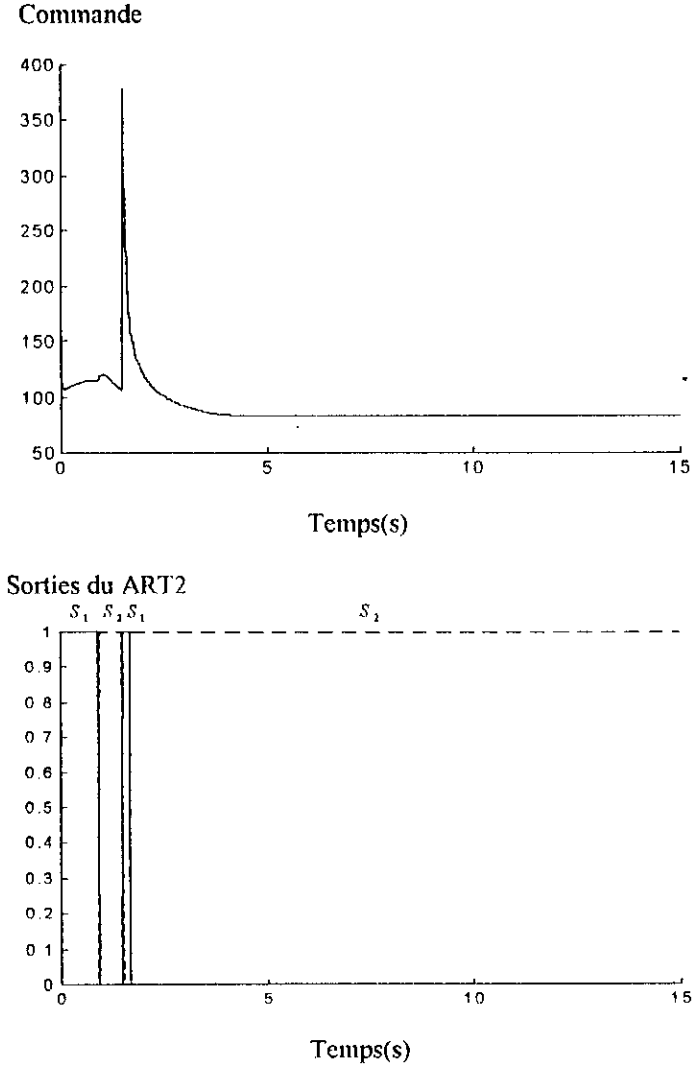


Fig.VI.42.b. Les sorties du ART2 et la commande générée par le neuro-contôleur.

——— Sortie S_1

····· Sortie S_2

— · — · — Sortie S_3

Nous constatons, à travers ces simulations, l'aptitude de ce régulateur à conduire le système dans les différentes régions de fonctionnement. Ceci grâce au réseau classificateur ART2 qui gère le "basculement" entre les trois régulateurs neuraux suivant les différentes régions de l'espace des états où le système se trouve.

VI.4.5 Commande avec Modèle de Référence

Dans cette partie, nous allons entraîner un réseau LBF pour appliquer au système une commande avec modèle de référence.
Pour cela, un modèle de référence, que la concentration doit suivre, est imposé. Ce modèle est décrit par l'équation suivante:

$$C_a(k+1) = 0.9C_a(k) + 0.1r(k) \tag{VI.20}$$

Pour cela, un réseau de neurones LBF multicouches à trois entrées, dont deux sont récurrentes, à deux couches cachées de 20 et 17 neurones et deux sorties, est utilisé.

Ce réseau est entraîné avec une séquence en entrée:

$$r(k) = 0.4\left[\sin(2\pi k/7) + \cos(2\pi k/13) + \sin(2\pi k/25)\right] + 0.15 \tag{VI.21}$$

Les grandeurs de commande que le réseau doit générer sont inconnues; et le signal d'erreur est mesuré à la sortie du système, non celle du réseau. En (VI.4.3), pour surpasser ce problème, nous avons utilisé la technique du réseau émulateur. Cette fois, afin d'éviter l'utilisation de ce réseau, nous avons choisi d'effectuer l'entraînement avec la méthode d'optimisation aléatoire ROM. En effet, avec cette méthode l'apprentissage se résume à l'optimisation du critère définissant l'écart entre la sortie du réacteur et celle du modèle de référence, et nous évitera la nécessité de calculer le Jacobien où de connaître les sorties au préalable, que le réseau doit générer. La figure (VI.43) présente le schéma d'apprentissage du réseau

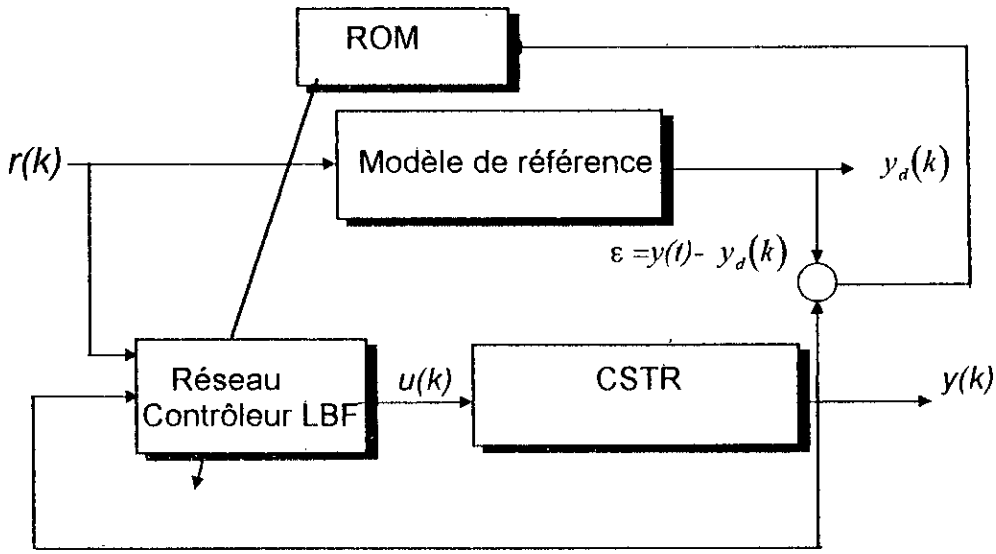


Fig.VI.43 La structure utilisée pour l'apprentissage du neuro-contrôleur par la ROM.

Avec la ROM à variance dynamique (cf. IV.2.1.4), où la valeur initiale de la variance est égale à 1, l'entraînement a nécessité 200000 itérations.

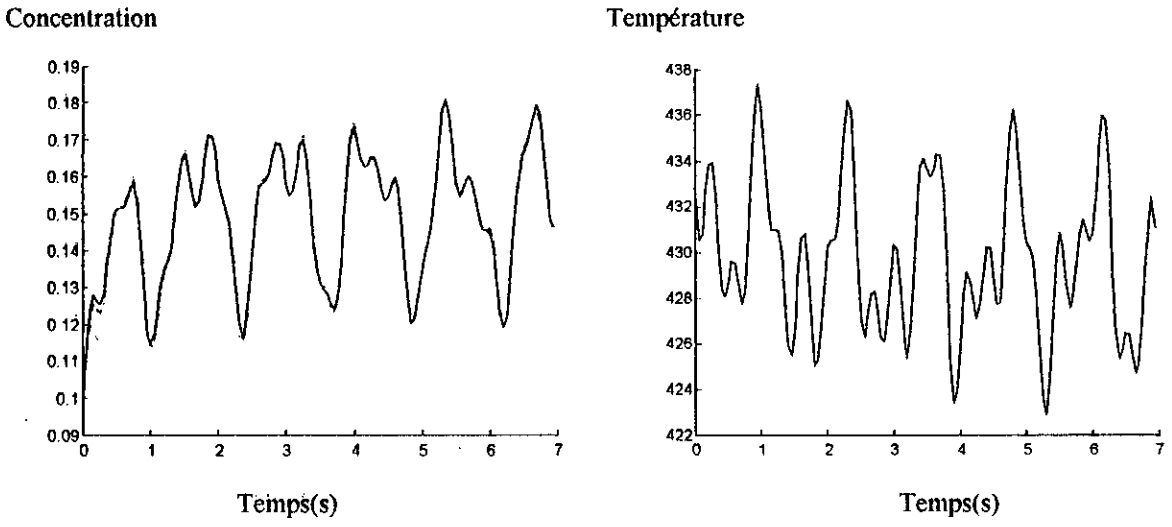


Fig.VI.44.a. La réponse du système après entraînement du réseau contrôleur pour une commande avec modèle de référence.
——— La concentration du produit.
..... La sortie du modèle de référence.

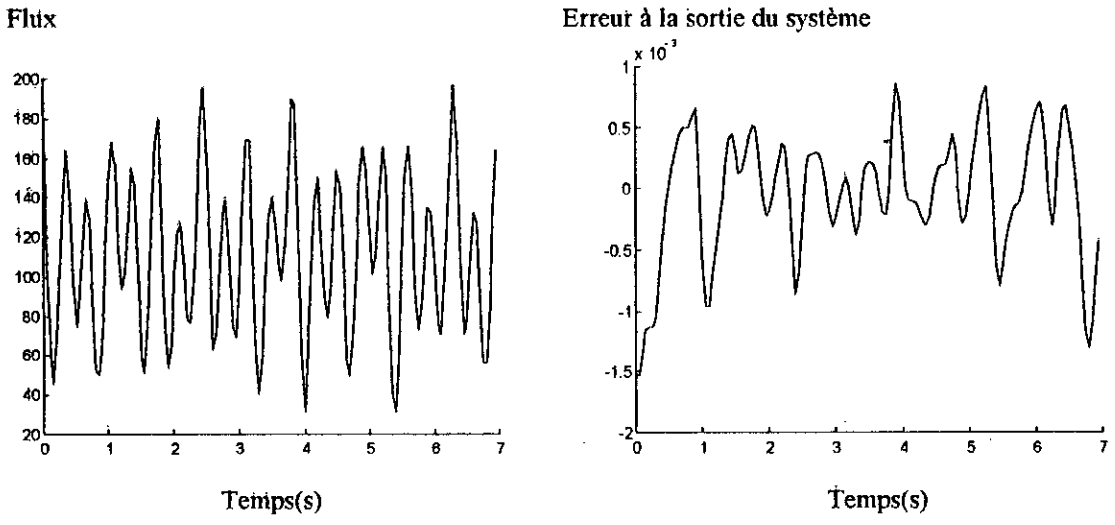


Fig.VI.44.b. La réponse du système après entraînement du réseau contrôleur pour une commande avec modèle de référence.

La figure (VI.44.a) présente l'évolution de la concentration et celle de la température après entraînement du réseau contrôleur. Nous remarquons que le réacteur suit le modèle de référence avec une bonne précision. Sur la figure (VI.44.b), on peut voir la commande générée par le réseau et l'évolution de l'erreur à la sortie du système.

Ainsi grâce à la méthode ROM, nous avons réussi à faire entraîner le réseaux LBF à conduire le système pour suivre le modèle de référence, sans être obligé d'identifier le système ou d'approximer son Jacobien. Ce qui constitue l'avantage de cette méthode d'entraînement par rapport à la Backpropagation.

Conclusion

Après une étude sur les possibilités de commande par réseaux de neurones, nous avons proposé dans ce chapitre plusieurs stratégies de commande neurales. Ces stratégies ont été utilisées dans nos applications pour la commande d'un bras de robot puis celle d'un réacteur chimique.

Dans la commande du bras de robot, les réseaux RBF Gaussiens, à travers leur caractéristique de calcul et de réadaptation locaux, nous ont permis de les utiliser pour construire un régulateur auto-ajustable, afin de faire face aux modifications et erreurs d'identifications des paramètres du bras de robot utilisé. En effet, grâce à la classification des entrées, qui est effectuée dans la couche cachée, ce réseau arrive à se réadapter plus rapidement; Ce qui nous justifie notre utilisation de ces derniers pour un apprentissage en temps réel.
Pour la réadaptation des poids, Il est impossible de connaître les valeurs des commandes que le réseau doit générer en sortie. La méthode d'optimisation Aléatoire ROM, basée sur l'optimisation d'un critère sans obligatoirement savoir les sorties désirées, nous a permis de surmonter ce problème et d'obtenir des résultats satisfaisants. Au court de notre utilisation de cette méthode, nous avons, cependant, été contraint à l'adapter pour un entraînement en Data Learning et en temps réel.

La méthode ROM a été, aussi, utilisée dans une stratégie de commande différente appliquée au réacteur chimique. Cette méthode, qui ne nécessite pas la connaissance des sorties désirées du réseau, nous a permis de surmonter la nécessité de l'approximation du Jacobien du système, posé dans le cas de l'utilisation des autres méthodes. Ce qui a rendu possible d'effectuer une commande avec modèle de référence directement avec un seul réseau du type LBF.

Pour la commande du réacteur chimique, dans tout l'étendu de son fonctionnement, nous avons mis au point une stratégie, utilisant les réseaux ART2, qui sont à apprentissage non-supervisé. Cette stratégie peut constituer un outil intéressant pour la commande de systèmes, dont le comportement peut varier d'un intervalle à l'autre, où dont l'objectif de la commande est différent suivant l'intervalle où l'on se trouve. La décision sur l'action de la commande à adopter se fait automatiquement par le réseau ART2, et particulièrement plus rapidement que dans le cas des procédures classique utilisées. Celle ci nous permet d'autre part de concevoir des structures de commande entièrement neurale.

CONCLUSION GENERALE

CONCLUSION GENERALE

Pour ce travail, nous avons mené une large recherche bibliographique qui nous permis d'explorer les différents types de réseaux de neurones mis en oeuvre à ce jour.
Depuis l'apparition du premier neurone artificiel mis au point par Mc Culloch et Pitts en 1943, l'évolution des réseaux de neurones reste marqués par deux travaux dont l'importance est toujours valable jusqu'aujourd'hui.

- Le modèle neural entièrement connecté de J. Hopfield conçu en 1982, qui a résolu le problème de stabilité par l'utilisation du formalisme de fonction de Lyapunov et notamment grâce à l'analogie avec la notion des spins des systèmes magnétiques.

-L'algorithme d'apprentissage Backpropagation, qui a été mis au point par un groupe de chercheur de l'université de Stanford, aux U.S.A, Rumelhart, Hinton et williams; qui a, enfin, permis d'exploiter la structure fort puissante des réseaux Multicouches.

Nos applications se sont effectuées en reconnaissance de formes (caractères latins et chiffres arabes) et en identification et commande de systèmes, à savoir la commande d'un réacteur chimique puis celle d'un bras de robot. Pour cela nous avons utilisé les réseaux à apprentissage supervisé et ceux à apprentissage non-supervisé.

En reconnaissance, le réseau de Hopfield, qui est basé sur la règle d'apprentissage de Hebb, est sensible au bruit. Celle ci devient, d'autre part, inefficace lorsque les entrées sont non-orthogonales entre elles, notamment quand le nombre de prototypes à mémoriser est très important.
Les BAM, qui sont une forme généralisée du réseau de Hopfield, peuvent aussi bien être utilisées en tant que mémoire hétéro-associative ou mémoire autoassociative. Ainsi, ceux ci peuvent remplacer le réseau de Hopfield avantageusement. En effet, nous avons constaté dans nos applications que ces derniers nécessitent un nombre de poids plus réduits et peuvent être plus sélectives vis à vis au bruit.
Le réseau basé sur la règle de projection OLAM permet une capacité de stockage plus importance et n'exige pas des exemples à mémoriser d'être mutuellement orthogonaux.
Le réseau de Hamming se distingue particulièrement par rapport aux autres, par sa capacité de mémorisation largement plus importante, et surtout par sa robustesse aux bruits. C'est le réseau qui nous a donné les meilleurs résultats. Nous constatons que ce réseau est bien adapté pour les problèmes de traitement de signal et particulièrement pour le filtrage de signaux bruités et la reconnaissance.

En identification de système, c'est la propriété d'approximation des réseaux qui est directement mise en oeuvre pour construire des modèles pour les systèmes dynamiques non-linéaires.

Nous avons relevé l'avantage qu'offre la caractéristique de calcul et de d'adaptation paramétrique locaux, dont les réseaux RBF sont dotés, pour l'identification de systèmes. Cette caractéristique, que l'en ne retrouve pas dans les réseaux LBF (dotés de fonctions Sigmoïdes), permet un apprentissage précis sur plusieurs régions de l'espace des états.

Pour les réseaux LBF, les méthodes basées sur le gradient du second ordre, bien que plus consommatrices du temps de calcul, et plus compliquées à l'implémentation, constituent un moyen plus puissant pour l'entraînement de ces réseaux. C'est ainsi que nous avons constaté

que celles ci convergent après un nombre d'itérations, au plus égal à celles correspondant à un entraînement avec la Backpropagation sous toutes ces formes.

Les réseaux dynamiques, par leur structure interne récurrente, peuvent être utilisé pour la construction de modèles pour des systèmes dynamiques avec un nombre de neurones récurrents qui est très réduits, comparativement aux réseaux statiques. ces réseaux peuvent, cependant montrer des problèmes d'instabilité. Pour cela le choix des paramètres en apprentissage est souvent très délicat.

En commande, les caractéristiques d'adaptabilité et de mémorisation associative des réseaux à apprentissage supervisé sont exploités. Il est, cependant, important de noter que c'est aux aptitudes d'approximation de ces réseaux que l'on fait aussi appel pour associer les actions de commandes appropriées aux grandeurs mesurées sur les systèmes.

Dans la commande du réacteur chimique, Pour résoudre le problème du calcul du Jacobien, en commande avec modèle de référence, notre utilisation de la méthode d'optimisation aléatoire ROM a montré de bons résultats. Celle ci permet d'entraîner le système d'une manière plus souple par rapport aux autres méthodes utilisées nécessitant l'approximation du jacobien et sans être obligé de l'identifier.

En commande adaptative, il n'existe en général pas de garantie pour la stabilité du processus. Notre utilisation des réseaux RBF pour la commande d'un bras de robot a montré que ces réseaux peuvent être adaptés pour un entraînement en temps réel, nécessaire pour un régulateur auto-ajustable. Cette réadaptation s'est faite sans générer des perturbations importantes sur le système, grâce à la caractéristique d'adaptation local de ces réseaux. Les réseaux LBF, par contre, ne sont pas très adaptés à un tel apprentissage. L'adaptation des poids de ces réseaux en temps réel risque de concentrer le fonctionnement sur une zone créant ainsi l'effet de *non-entrainnement,* ce qui peut déstabiliser le système. Pour cet apprentissage en temps réel du régulateur à base de réseaux RBF Gaussiens, nous avons adapté la méthode ROM pour être utilisée à un tel apprentissage.

Notre utilisation du ART2, qui est un réseau à apprentissage non-supervisé, dans la commande par température du réacteur chimique suivant la région de fonctionnement a été très efficace. En effet, à travers cette technique de commande proposée dans cette thèse, nous avons montré la possibilité de commander des systèmes complexes, dont le fonctionnement varie d'une région à l'autre entièrement par réseaux de neurones. La structure, que nous avons élaborée, est aussi intéressante en implémentation matérielle et bénéficie de la capacité de traitement parallèle et distribué des réseaux de neurones. Elle remplace les procédures classiques, généralement utilisées pour ce types de systèmes, en permettant du régulateur de fournir une action beaucoup plus rapide. Cette structure nous permet d'introduire les réseaux de neurones à apprentissage non-supervisé en automatique. Ce qui n'est pratiquement pas fait encore d'après nos recherches bibliographiques.

A la fin, nous terminons avec quelques idées représentant les perspectives à lesquelles ce travail peut s'ouvrir.

En reconnaissance, l'extension des réseaux à apprentissage non supervisé, comme celui de Hamming, par exemple, sur des images analogiques, moyennant les transformations nécessaires, serait très intéressante. De plus, il serait très intéressant d'exploiter les techniques

de compression de signaux, tout en gardant leurs caractéristiques principales. Nous signalons que ces techniques de représentations existent déjà. Une utilisation de ces méthodes, comme forme de représentation, des signaux à traiter par les réseaux de neurones, serait très bénéfiques et permettrai d'accélérer le traitement.

En identification et commande de systèmes, la structure de commande neurale *par classification suivant les régions de fonctionnement*, que nous avons proposée, ouvre la voie à l'utilisation des réseaux à apprentissage non supervisé dans le domaine de l'Automatique. Cette structure de commande peut être développée sous plusieurs formes. Celle ci peut remplacer plusieurs méthodes de commande classiques, qui sont basées sur le basculement ou celles nécessitant une loi de commande dont l'action doit varier suivant l'état du système.

Enfin, nous mentionnons qu'à travers ce travail, nous avons constaté que les réseaux de neurones c'est une famille d'outils très diversifiés. Chaque modèle a ces propres caractéristiques et peut être destiné à des opérations spécifiques. Nous pensons qu'il serait intéressant de construire des systèmes de traitement, où de commande complexes, entièrement par réseaux de neurones. Ainsi, des modèles différents de réseaux peuvent être assemblées pour travailler ensemble dans une structure organisée.

REFERENCES BIBLIOGRAPHIQUES

[Ant90] Antony N. M. and. Farrel J.A. *Associative Memories Via Artificial Neural Networks.* IEEE Control Systems Magazine (1990), .

[Ahm93] Ahmed K. and SutharsananS. I. *Identification and Decentralized Adaptive Control Using Dynamical Neural Networks With Application to Robotic Manipulator.* IEEE Trans. on Neural Networks, Vol. 4, pp 919-929. (1993)

[Asr93] Asriel U. Levin and Kumpati .Narendra S. *Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilisation.* IEEE Transactions on Neural Networks. Vol 4. NO 2, March.. (1994).

[Asr94] Asriel U. Levin and Kumpati Narendra S. *Identification Using Feedforward Neural Networks.* (1994).

[Ast89] Astrom K.J. and Wittenmark B. *Adaptive Control* Adison Wisley Publishing Company.(1989)

[Bab89] Baba N. *A New Approach for Finding the Global Minimum of Error Function of Neural Networks.* (1989)

[Bar89] Bark A. pearlmuter. *Learninfg State Space Trajectories in Reccurent Neural Networks.* Neural Computation 1, 263-269 (1989)

[Bau88] Baum E. B. *Complete Representations For Learning From Exemples.* Springer-Verlag New York. (1988)

[Bav87] Bavarian B. *Introduction to Neural Networks for Intelligent Control.* IEEE Systems Magazine.(1988).

[Bel96] Bellouchrani, A., Cichochi A. and Meriaim A. *A Blind Identification and Separation Technique via Multi-layer Neural Networks.* ICONIP 96' Hong Kong. (1996)

[Bil90] Billings S. A. and Jamaluddin B. .*A Comparisson of The Backpropagation And Recursive Prediction Error Algorithms for Training Neural Networks.* Mechanical Systems and Signal Processing, pp 233-255. (1990).

[Bil92] Billings S. A. and Chen S. *Neural Networks and System Identification.* Neural Networks for Control and Identification Edited by K. Warwik, G. W. Irwin, K. J. Hunt. Peter Perinus. (1992)

[Bla91] Blayo F. *Réseaux de Neurones Formels Supervisés.* Ecole de Printemps Neurosciences et Sciences de l'Ingénieur. Villard de Lans (Isère, France), NSI'91. (1991)

[Bur93] Burrons T. L. *Feed-forward and Reccurent Neural Networks forSystem Identification.* Technical Report CUED/F-INFENG/TR158. Cambridge University Enginneering Department. England December. (1993)

[Cha95] Chao-Che K. and K. wang Y. Lee. *Diagonal Reccurent Neural Networks for Dynamic Systems Control.* IEE Trans. on Neural Networks Vol **6** pp 144-156. (1995)

[Che90] Chen F.C. *Back-Propagation Neural Networks for Nonlinear Self-Tuning Adaptive Control.* IEEE Control Systems Magazine. Vol. 10, No. 3, April. (1990)

[Coh83] Cohen M. A. and Grosberg S. *Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks.* IEEE Transactions on Systems, Man and Sybernetics, Vol. SMC-13, NO.5, September/October (1983).

[Chu90] Chu S. R. Shoureshi R. and Tenorio M. *Neural Networks for Systems Identification.* IEEE Control Systems Magazine. Vol. 10, No.3, April. (1990).

[Dem91] Demongeot J. *Neural Networks: From Formal Neuro-Computing to Real Neural-Modelling.* Ecole de Printemps Neurosciences et Sciences de l'Ingénieur. Villard de Lans (Isère, France), NSI'91. (1991)

[Ebe95] Eberhardt R., P.M. Hélène, Puzenat Didier and Royet J.P. *Is It Possible to Determinate Odors with Common Words?.* Reaserch Report N95-31. Laboratoire de l'Informatique du Parallélisme. Ecole Normale Supérieur de Lyon. (1995)

[Eil88] Eilbert J.L. Guez A. and M.Kam. *Neural network architecture for Control.* IEE Control Systems Magazine (1988).

[Erg95] Ergenzinger S.and Thomsen E. *An Accelerate Learning Algorithme for Multilayer Perceptron: Optimisation Layer by layer.* IEEE Transactions on Neural Networks. Vol 6, NO. 1, Junuary.(1995).

[Fre92] Freeman, J. D. and M. Skapura. *Neural Networks, Algorithms, Applications, and Programming Technique*. Adison Wesley company(1992)

[Fu 87] K. S. Fu R. C Gonzalez C. S. G. Lee. *Robotics: Control, Sencing, vision, and Intelligence.* McGraw-Hill Company. (1987)

[Fuk83] Fukushima K., S. Miyakey and T. Ito. *Neocongnitron, A Neural Network Model for a Mechanism of Visual Pattern Recognition.* IEEE Transactions on Man, and Sybernetics, SMC, Vol. 13 No. 5, PP 826-834, September-October.(1983)

[Fuk87] Fukushima K. *Neural Network Model forSelective attention in Visual Pattern Recognition and Associative recall.* Applied Optics, Vol. 26 No. 23, pp 4985-4992. Desember.(1987)

[Fuk92] Fukuda T. and T. Shibata. *Theorie and Application of Neural networks for Industrial Control Systems.* IEEE Transactions on Industrial Electronics Vol. 39 No 6, December. (1992)

[Gra89]. Gras L.C.J.M and H. Nijmeijer. *Decoupling in Nonlinear Systems: from Linearity to Nonlinearity.* IEE Proceedings. Vol. 136 No 2, March (1989)

[Gil87] Giles C.L. and T. Maxwell. Learning, Invariance, and Generalisation in Higher-order Neural Networks. In [Pen 92]. pp 94-100. IEEE Computer Society Press Tutorial. (1992)

[Gue95] Guenfaf L. *Etude de Differentes Strategies de Commande Adaptative: Application à un Robot Manipulateur* Thése de Magister à l'ENP Departement de G. Electrique. Automatique. (1995)

[Ham95]Hamzi B. et S Labiod *Identification et commande des systemes dynamiques par réseaux de neurones.*Thése d'ingéniorat en automatique à l'Ecole Nationale Polytechnique d'Alger.(1995).

[Has92] Hashimoto, H, T. Kubota, M. Satoand F. Harashima. *Visual Control of Robotic Manipulator Base on Neural Networks.* IEEE Transaction on Industrial Electronics, Vol 39, NO 6, December. (1992)

[Hér94] Hérault J. *Réseaux Neuronaux et Traitement du Signal.* Traité Des Nouvelles Technologies ed. HERMES (1994).

[Hua90] Huang S.C. and Hunang Y.F. *Learning Algorithms for Perceptron Using Back-Propagation with Selective Updates.*IEEE Control Systems Magazine. Vol. 10, No.3, April. (1990)

[Hus93] Hush D.R. *Progress in Supervised Neural Networks.* IEEE Signal Magazine. Junuary(1993).

[Hun92] Hunt K. J. D. Sbarbaro and P.J. Gawthrop *Neural Networks for Control Systems.* Automatica Vol 6 pp 1112-1992.

[Isi89] Isidori A. *Nonlinear Control Systems: An Introduction.* Springer Verlag. (1989)

[Jer93] Jervis T. *Connectionist Adaptive Control.* Philosophy Doctorat Theses at The University of Combridge. (1993).

[Jut91] Juten C. *Apprentissage sans superviseur et incrémental. Réseaux à Architecture Evolutive* Ecole de Printemps Neurosciences et Sciences de l'Ingénieur. Villard de Lans Isère, France. NSI'91. (1991)

[Kar93] Karayannis N.B and A. N. Venetsanopoulos. *Artificial Neural Networks, Learning, Performance, Evaluation and application.* Kluwer Academic Publishers.(1993).

[Kar93] Karakasoglu A. and S. I. Sudharsanan. *Identification and Decentralized Adaptive Control Using Dynamical Neural Networks With Application to Robotic Manipulators.* IEEE Transactions on Neural Networks, Vol 4, NO 6, Novembre. (1993).

[Kha96] Khalil, K. H. *Adaptive Feedback Control of Nonlinear Systems Represented by Input-Output Models.* IEE Transactions on Automatic Control, Vol 41, NO 2, February. [1996]

[[Khe94] Khemaissia S. and A. s. Morris. *Review of Networks and Choice of Radial Basis function Networks for System Identification.* Technologies Avancées N 6, pp 55- 85.(1994).

[Koh77] Kohonen T. *Associative Memories* Printice Hall. (1977).

[Koh88] Kohonen T. *An Introduction to Neurl Computing.* Neural Networks Vol 1 pp 3-16 Pergamon Press(1988).

[Kol89] Kollias, S. and D. Anastassiou. *An Adaptive Least Squares Algorithme for Efficient Traning of Artificial Neural Networks.* IEEE Transactions on Circuits and Systems Vol 36 NO. 8, August. (1989).

[Kos92] Kosko B. *Neural Networks and Fuzzy Systems, Adynamical Machine Approach To Machines Systems.*Printice Hall International. .(1992)

[Kun93] Kung S. Y. *Digital Neural Networks* PTR Printice Hall. .(1993)

[Lee88] Lee S. and Kil M.R. *Multilayer Feedforword Potential Function Network.* Second International Conference on Neural Networks Vol 1 pp 161-171.

[Lee91]Lee S. and Kil M.R. *A Gaussian Potential Function Network With Hierarchically Self-Organizing Learning.* Neural Networks. Vol. 4, pp 207-224.(1991)

[Lig95] Lightbody G. and Prof.G.W. Irwin. *Neural Model Reference adaptive Control.* IEEE Proc., Control Theory Appl., Vo 142, No 1, Junuary.(1995).

[Lip87] Lippmann R. *An Introduction To Computing With Neural Nets.* IEEE Acoustics Speech, and Signal Processing Magazine. pp 4-22. (1987)

[Mul91] Muller B. and J. Reinhart. *Neural Networks.*pergamon press.(1991).

[Nag90] Nagata S. Sekugushi, M. and Asakawa, K. *Mobil Robot Control by a Structured Hierarchical Neural Networks.* IEEE Control Systems Magazine.Vol. 10, No 3. (1990)

[Nar90] Narendra K. S. and K. Parthasarathy.*Identification and Control of Dynamical Systems Using Neural Networks.* IEEE Transactions on Neural Networks. Vol 1, NO.1, March.(1992)

[Nar92] Narendra K. S. *Intelligent Control Using Neural Networks.* IEEE control Systems. April.(1992)

[Nav90] Naveen V. B., P.A. Minderman,Jr.T. McAvoy and N.S. Wang. *modeling Chemical Process Systems via Neural Computation.* IEEE Control Systems magazine. Vol 10 NO. 3, pp 24-30. April.(1990).

[Ner92] Nerrand O., Roussel R.P., Personnaz L. and Dreyfus G. *Neural Networks and Non-linear Adaptive Filtering: Unifying Con cepts and New Algorithms.* Neural Computation. Vol. 5 No 2, pp. 165-199. (1992)

[Ngu90] Nguyen D, and B. Widrow. *Neural Networks for Self-Learning Control Systems.* IEEE Control Systems Magazine. Vol 10 NO 3, pp 18-23. April. (1990).

[Pat96] Patterson Dan W. *Artificial Neural Networks. Theory and Applications.* Printice Hall.(1996)

[Pan90] Panos, J. A. *Neural Networks in Control Systems.* IEEE Control Systems Magazine. Vol 10, No. 3. (1990)

[Pen93] Penkaj M. & Benjamin W. artificial Neural Networks: *Concepts and Theory.* IEEE Computer Society Press Tutorial. (1992)

[Per86] Personnaz, l.L. Guyon. *Collective Computationnal Properties of Neural Networks,* Phys. Rev.A, Vol. **34** N 5 pp 4217-4228. (1986).

[Pol96] Polycarpou, M. M. *Stable Adaptive Neural Networks Control Scheme for Nonlinear Systems.* IEEE Transactions on Automatic Control, Vol. 41, NO. 3, March. (1996)

[Pra78] W. K. Pratt. *Digital Image Processing.* Wiley Interscience publications. 1978

[Psa88] Psaltis, D., A. Sideris and A. Yamamura. *Neural Controllers.* Proc. IEEE Conf on Neural Networks, San Diego. vol IV.(1988).

[Puz95] Puzenat D. Reaserch Report N95-31. Laboratoire de l'Informatique du Parallélisme. Ecole Normale Supérieur de Lyon. (1995)

[Qin92] Qin, S. H.T. Su and T. G McAvoy. *Comparaison of Four Net Learning methods for Dynamical Systems Identification.* IEEE Transactions on Neural Networks. Vol 3, No 1, January. (1992)

[Ren 95] Renders, J.M. *Algorithmes Génétiques et Réseaux de Neurones. Applications à la Commande de Processus.* Hermes.(1995).

[Riv 95] Rivals I. *Modélisation de Processus par Réseaux de Neurones: Application au pilotage d'un Véhicule Autonome.* Thése de Doctorat en Physique à l'Université Pierre et Marie Curie. Paris,6. (1995).

[Rum 86] Rumelhart D. E., G.E. Hinton and R.J. williams. *Learning Internal Representation by Error Propagation.* In D.E Rumelhart, G.E. hinton and R. J. Williams. Parallel Distributed Processing. Vol. 1 , pp 318-362. combrige, M.A, MST Press.(1986).

[Saa 94a] Saad, M., L. A. Dessaint, P. Bigras and K. Al-Haddad. *Adaptive Versus Neural Adaptive Control: Application to Robotics.* International Journal of Control and Signal Proceeding, Vol 8, pp 223-236. (1994)

[Saa 94b]Saad M., L.A. Dessaint, P.Bigras and K.Al Haddad. *Adaptive Versus Neural Adaptive Control: Application to Robotics.* International Journal of Adaptive Control and Signal Processing, Vol. 8, 223-236.(1994).

[Sag 92] Saga K., Sugasaka T., Sekugushi S., Nagata S. and Asakawa K. *Mobil Robot Control by Neural Networks Using Self-Supervised Learning.* IEE Transactions on Industrial Electronics. Vol. 39, N. 6, December.(1992)

[San 92] Sanner R.M. *Gaussian Networks for Direct Adaptive control.* IEEE Trans. on Neural Networks Vol. 3 pp837-863. (1992).

[Sas 89] Sastry S., and Isidory S. *Adaptive Control of Linearizable Systems.* IEEE Transactions on Automatic Control. Vol 34, NO 11. November. (1989).

[Sba93] Sbarbaro-Hofer D.,.D.Neumerkel, and K. Hunt *Neural Control ofa Steel Rolling Mill.* IEEE Control Systems pp 69-75. June (1993)

[Sim90] Simpson P.K. *Artificial Neural Networks.* Pergamon Press. (1990)

[Son93] Sontag E.D. *Neural Networks for control.* In [Tre 93], pp 339-380.(1993)

[Sto89] Stoten D.P. *Generalized Manipulator Dynamics, with Regard to Model Reference Adaptive Control.* INT. J. Control, Vol. 50 No. 6, 2249-2268.(1989)

[TRC89] *Neural Networks.* Tecnnical Report. Center of Conputation Reasearch in Economics and Management Science, MIT. (1989)

[Tre93] Tretelman H. L. & Willems J.C. eds *Essays on Control: Perspective in the Theory and its Applications.* Birkhäse, Boston. (1993)

[Urb95] Urbani D. *Méthodes Statistiques de Sélection d'Architectures Neuronales: Application à la conception de Modéles de Processus dynamiques.* Thése de Doctorat en Microélectrinique et Microinformatique à l'Université Pierre et Marie Curie. Paris. (1995).

[Van95] Van der Walt T., Barnard E. and Van Deventer J. *Process Modeling With the Regression Network.*IEEE Transactions on Neural networks. Vol.6, No 1, Junuary.(1995)

[Wer90] Werbos, P.J. *Backpropagation Trough Time: What It Does and how to Do It.* Proceedings of The IEEE, Vol. 78, No. 10, October. (1990)

[Wil89] Williams, R. J. and D. Zipser. *Experimental analysis of Real-time Reccurent Learning Algorithme.* Connection Science, Vol. 1, No 1. (1989)

[Yed95] Yeddou Y. M., Senouci B. D. & Souami M.C. *Edge Detection Using Neural Networks.* AMSE Conference on Systems , SYS'95, BRNO, July 3-5.(1995).

ANNEXES

Algorithme du HSOL: Hierrachical Self-Organized Learning de S. Lee et M. Kill

Les GPFN *(Gaussian potantial Function Networks)* sont des réseaux RBF Gaussiens.
Ce type de réseaux est doté d'une couche cachée, dont les neurones, appelés GPFU (*Gaussian Potantial Function Unit),* ont des sorties gaussiennes. Chacun d'eux définit une classe représentée par une hyper-sphère H_i (cf.4.3.3. p80).

Ainsi, pour une entrée $X = x_1, x_2, ..., x_v$ la sortie du *i éme* neurone est définie par:

$$\Phi_i\left(X, C^i\right) = e^{-d\left(X, C^i\right)/2} \tag{A.1}$$

où: $d\left(X, C^i\right)$ définit la distance entre cet exemple et le centre C^i (la fonction de base) tel que:

$$d\left(X, C^i\right) = \left(X - C^i\right)^T K^j \left(X - C^i\right) \tag{A.2}$$

En plus détaillé, on a:

$$d\left(x, c^j, K^j\right) = \sum_i \sum_k K^i_{jk}\left(x_j - c^i_j\right)\left(x_k - c^i_k\right) \tag{A.3}$$

où x_j est le *j éme* élément du vecteur d'entrée X, c^i_j représente le *j éme* élément du vecteur centre C^i associé au *i éme* élément Gaussien (GPFU) de la couche cachée.
K^i_{jk} est le *(j,k) éme* élément de la matrice K^i qui, en jouant sur les variances σ^i_j et σ^i_k (qui sont aussi appelées déviations marginales), module la forme de la gaussienne à la sortie du neurone tel que:

$$K^i_{jk} = \frac{h^i_{jk}}{\sigma^i_j \sigma^i_k} \tag{A.4}$$

où h^i_{jk}, appelé coefficient de corrélation, est défini par:

$$\begin{cases} h^i_{jk} = 1 & \text{si } j = k \\ \left|h^i_{jk}\right| \le 1 & \text{si } j \ne k \end{cases} \tag{A.5}$$

La sortie du réseau est constituée par des neurones linéaires:

$$y_j = \sum_{i=1}^{m} \lambda_{ji} \Phi_i \tag{A.6}$$

avec $\Phi_i = \Phi_i\left(X, C^i\right)$

La méthode d'apprentissage présentée par M. Kill et S. Lee réadapte, en plus des poids synaptiques et des vecteurs des centres, le vecteur des déviations marginales et le vecteur des coefficients de corrélations de chaque neurone [Lee88, Kil91].

Nous présentons ci -dessous les étapes de cet algorithme:

1. Au début, choisir dans la base d'apprentissage un exemple aléatoirement. Générer un *GPFU (Gaussian Potential Function Unit)* pour le représenter. Le vecteur de centres (moyennes) et celui des poids le reliant à la couche de sortie sont initialisés respectivement aux valeurs de l'exemple d'entrée et à ses sorties désirées. La covariance initiale σ_0 est fixée à une valeur arbitraire positive. Initialiser le rayon effectif r_0 de la classe représentée par ce neurone à une valeur de préférence grande.

2. Injecter le *p éme* exemple et poser *p=p+1*.

3.

◆Si l'erreur associée au *k éme* neurone à la sortie du réseau $\left|d_k^p - y_k^p\right|$ *(avec k=1,..n)* est supérieur au seuil toléré:

•Trouver un neurone caché qui a la même classe que cette sortie. Un neurone est défini ayant la même classe que cette sortie, lorsque le poids synaptique le reliant avec cette sortie est de même signe qu'elle.

-Si ce GPFU existe, vérifier si le rayon r_i de l'hyper-sphére H_i de la classe représentée par ce neurone couvre l'entrée injectée au réseau, ce qui veut dire que la sortie de ce GPFU est inférieur à G_i tel que:

$$G_i = e^{-\left(r_i^2/2\right)} \tag{A.7}$$

-Si c'est le cas, appliquer la réadaptation des paramètres du neurone (équ.A.9; A.10; A.11; A12).
-Sinon, générer un nouveau neurone (GPFU) *i* dont le vecteur centres est égal au nouvel exemple en entrée, et le vecteur des poids est égal aux sorties désirées que doit générer cette entrée; avec la matrice forme $K^i = 1/\sigma_0^2 I$ (*I*:matrice identité) et le rayon effectif $r_i = r_0$.

◆Si l'erreur associée au *k éme* neurone à la sortie du réseau $\left|d_k^p - y_k^p\right|$ est inférieur au seuil toleré, appliquer directement l'adaptation des paramètres du réseau (équ. A.9; A.10; A11; A.12).

4. Si le réseau est saturé, ce qui veut dire que ce lui ci a atteint un extremum et l'erreur n'évolue presque plus, réduire les rayons des GPFN afin d'affiner la classification par le rajout de nouveaux GPFN qui seront nécessaire.

$$r_l^{new} = \begin{cases} r_l^{old}.r_d & \text{pour } l = 1,...m \text{ si } r_l > r_{\min} \\ r_l^{old} & \text{pour } l = 1,...,m \qquad \text{sinon} \end{cases} \tag{A.8}$$

Où $r_{\min}$ est la borne inférieure des rayons, et r_d le taux de décrementation des rayons.

5. Si tous les exemples sont présentés *(p=M)*, aller à l'étape **6.**, sinon revenir à l'étape **2.**

6. Si le réseau montre des performances satisfaisantes arrêter l'entraînement, sinon aller à l'étape **2.**

S. Lee et M.Kill ont proposé des techniques pour détecter automatiquement la saturation du réseau. Ceci, en introduisant de nouvelles variables qui mesurent le taux de saturation du réseau à chaque itération. Ainsi à partir des valeurs que peuvent prendre ces variables, on peut déceler la saturation et donc décider de réduire les rayons [Lee91].

La réadaptation des paramètres à l'étape **3.** se fait par la méthode du gradient. L'application de cette méthode à la présentation du *p éme* exemple en entrée donne pour $i=1,..m;\ j=1,..n;\ p=1,..M$:

Pour le poids reliant le *i éme GPFU* au *j éme* élément de sortie:

$$\Delta\lambda_{ji} = -\frac{\partial E_p}{\partial \lambda_{ji}} = \left(d_j^p - y_j^p\right)\Phi_i \tag{A.9}$$

où y_j^p est la *j éme* sortie du réseau pour le *p éme* vecteur d'entrée et d_j^p est sa valeur désirée.

Pour le *j éme* élément du vecteur moyenne du *i éme* GPFU:

$$\Delta c_j^i = -\frac{\partial E_p}{\partial c_j^i} = \sum_{l=1}^{v} K_{jl}^i \left(x_l - c_l^i\right)\Phi_i \sum_{k=1}^{n}\left(d_k^p - y_k^p\right)\lambda_{ki} \tag{A.10}$$

Pour la déviation marginale standard σ_j^i:

$$\Delta\sigma_j^i = -\frac{\partial E_p}{\partial \sigma_j^i} = \sum_{l=1}^{v} k_{jl}^i \frac{\left(x_j - c_j^i\right)\left(x_l - c_l^i\right)}{\sigma_j^i}\Phi_i \sum_{k=1}^{n}\left(d_k^p - y_k^p\right)\lambda_{ki} \tag{A.11}$$

Pour le coefficient de corrélation h_{jk}^i:

$$\Delta h_{jk}^i = -\frac{\partial E_p}{\partial h_{jk}^i} = -\frac{1}{2}\frac{\left(x_j - c_j^i\right)\left(x_k - c_k^i\right)}{\sigma_j^i \sigma_k^i}\Phi_i \sum_{l=1}^{M}\left(d_l^p - y_l^p\right)\lambda_{li} \tag{A.12}$$

Le Reacteur Chimique CSTR (Continuous Stirred Tank Reactor)

le model du CSTR est decrit par les equations differentielles non linéaires suivantes:

$$\dot{C}_a(t) = \frac{q}{v}\left(C_{a0} - C_a(t)\right) - k_0 C_a(t)\exp-\left(E/RT(t)\right)$$
$$\dot{T}(t) = \frac{q}{v}\left(T_0 - T(t)\right) + k_1 C_a(t)\exp-\left(E/RT(t)\right) + k_2 q_c(t)\left(1 - \exp-\left(k_3/q_c(t)\right)\right)\left(T_{c0} - T(t)\right) \quad \text{(B.1)}$$

$C_a(t)$ est la concentration du produit résultant
$T(t)$ est la température à l'interieur du CSTR.
$q_c(t)$ est le débit du réactif refroidissant qui represente la commande à appliquer.
C_{a0} représente la concentration de l'alimentation en entrée.
T_0 représente la temperature du produit à entrée.
T_{c0} représente la température du refroidissant.

q	Taux d'écoulement du processus	100 l/min
v	Volume du réacteur	100 l
k_0	Constante définissant le taux (la vitesse) de réaction.	$7.2 \times 10^{10}\ \text{min}^{-1}$
T_0	Temperature du produit en entrée.	350 K
T_{c0}	Température du refroidissant.	350 K
ΔH	Chaleur de la réaction.	$-2 \times 10^5 cal/mol$
C_p, C_{pc}	Chaleurs spécifiques	$1\ cal/g/l$
ρ, ρ_c	Densités des liquides	$10^3\ g/l$
h_a	Coefficient de transfert de chaleur	$7 \times 10^5\ cal/\text{min}/K$

$$k_1 = -\frac{\Delta H k_0}{\rho C_p}$$
$$k_2 = \frac{\rho_c C_{pc}}{\rho C_p v}$$
$$k_3 = \frac{h_a}{\rho_c c_{pc}}$$

Avec:
C_{a0}, la concentration d'alimentation à l'entrée. qui est fixée à $C_{a0} = 1\ mol/l$

Recurrent Backpropagation de F.J. Pineda

Nous rappelons le modèle proposé par F.J. Pineda qui a éte présenté dans (IV.5.1) Celui-ci est défini par (cf. ch IV p83;84) :

$$\tau \frac{dz_i}{dt} = -z_i + f(u_i) + I_i \tag{C.1}$$

$$u_i = \sum_j w_{ij} z_j \tag{C.2}$$

Les équations (C.1) (C.2) donnent à l'équilibre:

$$\begin{aligned} z^*_i &= f(u^*_i) + I_i \\ &= f\left(\sum_j w_{ij} z^*_j \right) + I_i \end{aligned} \tag{C.3}$$

L'erreur à minimiser est définie par:

$$E = \frac{1}{2} \sum_{k=1}^{m} E^2_k \tag{C.4}$$

tel que:

$$E_k = \begin{cases} d_k - z^*_k & k \in T \text{ (champs de sortie)} \\ 0 & \text{sinon.} \end{cases} \tag{C.5}$$

En appliquant l'algorithme de Backpropagation, on a:

$$\Delta w_{pq} = -\alpha \sum_i \frac{\partial E_i}{\partial w_{pq}} = \sum_i \frac{\partial z^*_i}{\partial u^*_i} \frac{\partial u_i}{\partial w_{pq}} \tag{C.6}$$

Or des l'équations (C.1), (C.2) et (C.3) on tire:

$$\frac{\partial z^*_i}{\partial w_{pq}} = f(u^*_i) \left(\delta_{ip} z^*_q + \sum_{j=1}^{n} w_{ij} \frac{\partial z^*_j}{\partial w_{pq}} \right) \tag{C.7}$$

où δ_{ip} représente le symbole de Kröneker.

le coté droit de l'équation (C.7) indique que les dérivés des sorties de tous les neurones par rapport à ce poids doivent être calculés. Pour contourner cette complexité, R. Rohwer a proposé une version matricielle, où on pose [Kar93]:

$$L_{ij} = \delta_{ij} - f'\left(u^*_i\right)w_{ij} \tag{C.8}$$

$$\sum_j L_{ij}\frac{\partial z^*_j}{\partial w_{pq}} = \delta_{ip} f'\left(u^*_i\right)z^*_q$$

Ainsi, on tire:

$$\frac{\partial z^*_k}{\partial w_{pq}} = \left(L^{-1}\right)_{kp} f'\left(u^*_p\right)z^*_q \tag{C.9}$$

Ce qui résoud le problème du calcul des dérivées.

D'autre part, l'équation (C.9) nécessite une inversion matricielle. Ce qui n'est pas toujours évident.
Pineda et Almeida ont proposé pour cela, une méthode qui consiste à poser:

$$v^*_p = \sum_k E_k\left(L^{-1}\right)_{kp} \tag{C.10}$$

d'où, en utilisant (C.7), on obtient une relation entre v_i et E_i tel que:

$$v^*_i - \left(\delta_{pi} - f'\left(u_p\right)w_{pi}\right) = E_i \tag{C.11}$$

L'équation (C.11) constitue l'état d'équilibre de l'équation différentielle suivante:

$$\dot{v}_i = -v_i + \sum_p v_p f'\left(u_p\right)w_{pi} + E_i \tag{C.12}$$

L'équation (C.11) est identique à l'équation (C.1). Ainsi cette dernière modélise un second réseau semblable au premier. Ce qui revient donc pendant l'entraînement à devoir obtenir d'abord l'état d'équilibre de ce deuxième réseau, pour calculer l'adaptation des poids du réseau principale à chaque étape.

Le Forced-Teached Real-Time Learning Algorithm de Williams & Zipser

A partir du RTRL, que nous avons étudié au chapitre IV (cf.IV.5.1), nous allons introduire cet algorithme qui consiste à Forcé le réseau à suivre les valeurs désirées.
Pour cela, on remplace pendant l'entraînement, les sorties des neurones leurs valeurs désirées .
On a, donc:

$$z_j(t) = \begin{cases} x_j(t) & j \in A \\ d_j(t) & j \in \mathrm{T} \\ y_j(t) & j \in \Omega - \mathrm{T} \end{cases} \qquad \text{(D.1)}$$

D'où l'on obtient:

$$\frac{\partial z_j}{\partial w_{pq}} = \begin{cases} 0 & j \in A \\ 0 & j \in \mathrm{T} \\ \dfrac{\partial y_j(t)}{\partial w_{pq}} & j \in \Omega - \mathrm{T} \end{cases} \qquad \text{(D.2)}$$

A partir de là, on a:

$$\frac{\partial y_j(t+1)}{\partial w_{pq}} = f'\left(u_j(t)\right)\left[\sum_{l \in \Omega - \mathrm{T}} w_{jl}\, \frac{\partial y_l(t)}{\partial w_{pq}} + \delta_{pj}\, z_q\right] \qquad \text{(D 3)}$$

et donc:

$$p^j_{pq}(t+1) = f'\left(u_j(t)\right)\left[\sum_{l \in \Omega - \mathrm{T}} w_{jl}\, p^l_{pq}(t) + \delta_{pj}\, z_q\right] \qquad \text{(D.4)}$$

L'algorithme du Forced Teached RTRL consiste, ainsi en l'application des instructions suivantes, Pour chaque étapes t :

1. Calculer l'état du réseau à l'instant t, en utilisant les valeur de $z_j(t)$ définies par l'équation (D.1)
2. Calculer les $p^j_{pq}(t)$ par l'équation (D.4)
3. Réadapter les poids:

$$\Delta w_{ij}(t) = \alpha \sum_{k \in \mathrm{T}} E_k(t) p^j_{pq}(t)$$

4. Remettre à zéro toutes les valeurs de $p^j_{pq}(t)$ pour lesquels il existe des sorties désirées $d_j(t)$ $j \in \mathrm{T}$.
5. Préparer les états forcés du réseaux à l'instant t pour l'étape suivante.
Aller à **1.** tant que $t < t_1$.