Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri de Tizi-Ouzou

Faculté de Génie Electrique et d'Informatique

Département d'Automatique

MEMOIRE DE MAGISTER

Spécialité **: Automatique**

Option : **Automatique des Systèmes Continus et Productique**

Thème: **Réseaux de neurones et espace d'état pour l'identification et la prédiction**

Présenté par : **AMOURA Karima**

Devant le jury d'examen composé de :

Président :	Kamel HAMMOUCHE	Maître de Conférence 'A',	UMM-TO
Rapporteur :	Saïd DJENNOUNE	Professeur,	UMM-TO
Examinateur :	Ahmed MAÏDI	Maître de Conférence 'B',	UMM-TO
Examinateur :	MELAH Rabah	Maître de Conférence 'A',	UMM-TO
Invité :	WIRA Patrice	Professeur,	UHA

Remerciements

Je tiens à adresser mes plus vifs remerciements à mon promoteur Mr S.Djennoune (professeur à l'UMMTO) et Mr P. Wira (Professeur à l'université de Haute Alsace) pour leurs rigueurs scientifique, leurs conseils, leurs disponibilité sans faille et pour tout le temps qu'ils m'ont accordé. Merci également pour m'avoir encouragé dans les moments difficiles.

Je tiens à exprimer ma reconnaissance à Mr D.Ould Abedeslam (Maître de conférences à l'Université de Haute Alsace) et Mr Jean-Philippe Urban (Professeur à L'Université de Haute-Alsace) pour leurs aides et pour leurs accueils au sein du laboratoire MIPS.

Je remercie également les membres du jury, Mr K.Hmmouche (maître de conférences à l'UMMTO) pour sa patience, pour son aide et qui me fait l'honneur de présider ce jury, Mr A.Maidi (maître de conférences à l'UMMTO) et Mr R.Mellah (maître de conférences à l'UMMTO) qui ont pris la peine de lire avec soin ce mémoire pour juger sont contenu.

Je tiens par ailleurs à remercier l'ensemble des membres du laboratoire MIPS à l'Université de Haute Alsace pour leurs accueils, leurs sympathies tout au long de mon stage. Je remercie chaleureusement mes collègues du laboratoire L2CSP pour leur bonne humeur et pour leurs soutiens.

Enfin un remerciement particulier à mes parents, à mes frères, ma sœur et mes amis.

Avant-propos

Le travail présenté dans ce mémoire a été effectué au sein du Laboratoire de Conception et de Conduite des Systèmes de Production (L2CSP) de l'Université Mouloud Mammeri de Tizi-Ouzou.

Le thème a été proposé par Patrice WIRA, Djaffar OULD ABDESLAM respectivement professeur, maître de conférence à l'Université de Haute-Alsace, Mulhouse, France.

Ce projet rentre dans le cadre d'un programme d'une coopération franco-algérienne : l'accord-programme n°09 MDU 783, intitulé « Techniques de commande intelligente : application aux systèmes électriques », liant le Laboratoire MIPS (Modelisation Intelligence Processus Systems) de l'Université de Haute Alsace, le Laboratoire ICEPS (Intelligent Control & Electrical Power Systems) de l'Université Djillali Liabes, Sidi Bel Abbès et le Laboratoire L2CSP (Conception et de Conduite des Systèmes de Production) de l'Université Mouloud Mammeri de Tizi-Ouzou.

Ce travail a été dirigé par le professeur Patrice WIRA de l'université de Haute Alsace, un stage d'un mois a été effectué au sein du laboratoire MIPS-TROP, UHA dans le cadre de ce magister.

Sommaire

Chapitre III : Applications

Introduction générale

Introduction

Les premiers travaux sur les réseaux de neurones remontent aux années 1940 avec les travaux de deux bio-physiciens, Warren McCulloch et Walter Pitts. En effet, ils inventent en 1943 le premier neurone formel et explorent les possibilités de ce modèle avec l'objectif de mémoriser des fonctions booléennes simples. La popularité de ces travaux fera que le modèle formel proposé portera par la suite leurs noms (on parle aujourd'hui encore du neurone de McCulloch-Pitts). Jusqu'au 1985 les fonctions réalisables étaient limitées aux fonctions linéaires. Depuis, un progrès considérable a été accompli avec la mise au point de l'algorithme d'apprentissage basé sur le calcul du gradient (Werbos, 1974) pour les réseaux multicouches mis en évidence par Marvin Minsky. En effet, aujourd'hui, avec l'apparition de différentes structures neuronales (appelées architectures) et le développement d'algorithmes performants pour l'apprentissage de ces réseaux ; et grâce aux résultats théoriques et pratiques obtenus au cours des dernières années, les réseaux de neurones sont devenus des outils de plus en plus utilisés dans divers autres domaines (bancaire, financier, militaire, météorologique, aéronautique, médecine…ect). L'utilisation des réseaux de neurones s'est avérée extrêmement réussie et appropriée, particulièrement dans le domaine de l'Automatique pour la résolution des problèmes de modélisation, d'identification (Widrow et Lehr, 1990) et de commande. Le perceptron multicouche a reçu une attention considérable au regard de ses capacités d'identification des systèmes non linéaires. L'application de ce type de réseaux reste cependant limitée aux modélisations statiques de processus. Pour remédier à ce problème, de nouvelles architectures de réseaux de neurones ont été mises au point. Elles sont appelées "réseaux de neurones récurrents" où "réseaux de neurones dynamiques" et offrent la possibilité de prendre en compte de l'aspect temporel des données. Historiquement, le premier réseau récurent est le réseau de Hopfield (1982). Depuis, plusieurs architectures sont apparues, comme le réseau de Jordan (Jordan 1986) et le réseau d'Elman (Elman 1990). Cependant, pour modéliser un processus, disposer de plusieurs modèles ou architectures n'est pas suffisant. Il s'agit également de connaître ou de savoir déterminer les performances optimales qui peuvent être obtenues par un modèle.

Notre travail consiste à étudier un type de réseaux de neurones artificiels dynamiques appelés réseaux de neurones à espace d'état, appelés en anglais "State Space Neural Networks" ou SSNN. Les SSNN font partie des techniques de l'intelligence artificielle et trouvent des applications directes dans le domaine de l'Automatique. En effet, ce type de réseau présente des particularités intéressantes qui n'existent pas chez d'autres types de réseaux de neurones. Il permet notamment de s'approcher de l'Automatique classique avec son architecture qui s'approche d'une représentation d'état. Dès lors, ces poids représentent les matrices de la représentation d'état. D'autres particularités intéressantes nous incitent à l'étudier ainsi qu'à en évaluer ses performances et ses limites. L'objectif de ce travail consiste précisément à évaluer le potentiel et les performances optimales que l'on peut attendre de cette technique dans le cadre de tâches de modélisation, d'identification et de prédiction appliquées à des systèmes dynamiques. Les techniques d'apprentissage appropriées sont alors la méthode du gradient et la méthode de Levenberg Marquardt (Werbos 1974).

Le chapitre I de ce présent mémoire est consacré à une présentation des réseaux de neurones artificiels. Nous commençons par donner une brève présentation de l'évolution historique de cet axe de recherche. Puis, nous présentons quelques définitions sur les réseaux de neurones biologiques et mathématiques et les grands principes de fonctionnement liés (Touzet 1992), (Borne et al. 2007). Nous présentons notamment les principales architectures présentes dans la littérature. Les différentes techniques d'apprentissage seront également abordés (Wilamowski 2009), (Rivals 1995).

Le chapitre II introduit le réseau de neurones à espace d'état, désormais appelé SSNN et en anglais "State Space Neural Network" (Zamarreno et al. 2000), (Henriques et al. 2001). Le SSNN est un réseau de neurones du type récurrent. Il possède une architecture bien spécifique qui sert à la modélisation et la commande numérique des processus. Ce réseau de neurones est proche de la représentation d'état bien connue en Automatique. Quelques notions de base de la représentation d'état des systèmes sont également récapitulées dans ce chapitre.

Le chapitre III est un chapitre de résultats. Il propose les résultats obtenus en utilisant le SSNN pour modéliser différents systèmes dynamiques. Les essais ont été effectués à l'aide de différents systèmes. Le SSNN a été utilisé avec plusieurs méthodes d'apprentissage. La première est la méthode de Levenberg Marquardt qui est largement utilisée pour l'apprentissage des réseaux bouclés (dynamiques) et non bouclés (feedforward). La seconde est la méthode du gradient. Les essais effectués concernent notamment la modélisation et l'identification des différents systèmes linéaires du premier ordre, du quatrième ordre et du troisième ordre avec deux entrées de commande. Les deux méthodes d'apprentissage ont été mises en œuvre et évaluées. Par la suite, nous étudions les performances d'une identification d'un système linéaire du deuxième ordre en ajoutant à ce système un bruit (gaussien) additif sur la sortie, en utilisant un SSNN. Nous abordons par la suite les systèmes non linéaires. Dans ce cas, le SSNN est utilisé comme un prédicteur à un pas. A cet effet, l'évaluation des performances porte sur un système du quatrième ordre. D'autres essais ont été effectués pour reproduire le comportement de ce système, notamment lorsque la non linéarité du système varie. Enfin, le SSNN est comparé avec le perceptron multicouche qui est le réseau de neurones le plus souvent utilisé et apprécié pour ses capacités d'identification des processus dynamiques. Ces derniers tests comparatifs se basent sur deux exemples, un système linéaire d'ordre trois et un système non linéaire d'ordre deux.

Le chapitre IV conclue ce mémoire. Nous y dresserons un bilan final de notre travail et nous y donnerons quelques perspectives de continuité.

Chapitre I

I-1-Introduction

Le cerveau humain est un système d'une extrême complexité capable de résoudre des problèmes très difficiles et complexes. Il est composé de plusieurs éléments différents, mais un des constituants les plus importants est le neurone, il contient environ 100 milliards de neurones et chaque neurone est de fortes complexités, et en dépit des immenses progrès réalisés au cours des vingt dernières années dans le domaine de la neurobiologie, les spécialistes des neurones biologiques commencent à peine de comprendre quelques uns de leurs mécanismes internes et des questions aussi fondamentales que celle du codage des informations par le cerveau restent l'objet de nombreuses recherches (Dreyfus 2002) (Touzet 1992). Les neurones artificiels ressemblent à leurs congénères biologiques dans leur fonctionnement et leur architecture. Aujourd'hui, les réseaux de neurones artificiels (RNA) constituent une technique de traitement de données bien comprise et bien maîtrisée. Ces techniques s'intègrent parfaitement dans les stratégies de commande. D'un point de vue industriel on les retrouve implantées dans diverses industries, par exemple, dans les milieux financiers pour évaluer le risque financier, en pharmaceutique, pour le diagnostic médical, dans le domaine bancaire pour la détection de fraudes sur des cartes de crédit, en aéronautique pour la programmation de pilotes automatiques, dans le domaine de la prévision météorologique pour la prédiction de la vitesse du vent, dans diverses autres industries pour la surveillance et diagnostic de pannes (Xu 2002)...ect. Les applications sont nombreuses et partagent toutes un point commun essentiel à l'utilité des réseaux de neurones.

Ce chapitre est consacré à la présentation des réseaux de neurones artificiels. Nous commençons par donner une brève présentation de l'évolution historique de cet axe de recherche, ensuite nous montrons les ressemblances entre le neurone biologique et le neurone mathématique en présentant quelques définitions sur le neurone biologique et mathématique, les réseaux de neurones biologiques et mathématiques et leur principe de fonctionnement. Nous présentons ainsi quelques principales architectures que l'on retrouve dans la littérature en énonçant quelques propriétés fondamentales des réseaux de neurones artificiels, puis nous abordons les différentes techniques d'apprentissage.

I-2-Historique

L'origine de l'inspiration des réseaux de neurones artificiels remonte à *1890* où **W. James**, célèbre psychologue américain, introduit le concept de mémoire associative. Il propose ce qui deviendra une loi de fonctionnement pour l'apprentissage des réseaux de neurones, connue plus tard sous le nom de loi de **Hebb**. Quelques années plus tard, en *1943,* les travaux des deux bio-physiciens, **J. Mc Culloch** et **W. Pitts** montrent qu'un réseau de neurones discret, sans contrainte de topologie, peut représenter n'importe quelle fonction booléenne et donc émuler un ordinateur. C'est ensuite que **D. Hebb**, physiologiste américain, présente en *1949* les propriétés des neurones par le conditionnement chez l'animal. Ainsi, un conditionnement de type pavlovien tel que, nourrir tous les jours à la même heure un chien, entraîne chez cet animal la sécrétion de salive à cette heure précise même en l'absence de nourriture. La loi de modification des propriétés des connexions entre neurones qu'il propose, explique en partie ce type de résultats expérimentaux.

Les premiers succès de cette discipline remontent à *1957,* lorsque **F. Rosenblatt** développe le modèle du Perceptron. En *1960,* l'automaticien **Widrow** développe le modèle Adaline (Adaptative Linear Element). Dans sa structure, le modèle ressemble au Perceptron, cependant la loi d'apprentissage est différente. Celle-ci est à l'origine de l'algorithme de rétropropagation du gradient très utilisé aujourd'hui avec les Perceptrons Multi Couches. En 1969, **Minsky et Papert** publient le livre "Perceptrons" dans lequel ils utilisent une solide argumentation mathématique pour démontrer les limitations des réseaux de neurones à une seule couche. Ce livre aura une influence telle que la plupart des chercheurs quitteront le champ de recherche sur les réseaux de neurones.

Quelques années plus tard et plus précisément en 1982, le physicien **J. J. Hopfield** propose des réseaux de neurones associatifs et l'intérêt pour les réseaux de neurones renaît chez les scientifiques. C'est ensuite en 1985 que la rétro-propagation de gradient apparaît. C'est un algorithme d'apprentissage adapté au Perceptron Multi Couches. Sa découverte est réalisée par trois groupes de chercheurs indépendants. Dès cette découverte, nous avons la possibilité de réaliser une fonction non linéaire d'entrée/sortie sur un réseau, en décomposant cette fonction en une suite d'étapes linéairement séparables. Enfin, en 1989 **Moody et Darken** exploitent quelques résultats de l'interpolation multi variables pour proposer le Réseau à Fonction de base Radiale

(RFR), connu sous l'appellation anglophone «Radial Basis Function network » (RBF). Ce type de réseau se distingue des autres types de réseaux de neurones par sa représentation locale.

I-3-Définition et principe de fonctionnement des réseaux de neurones artificiels

I-3-1-Définition d'un neurone biologique

L'élément de base du système nerveux central est le neurone. Le neurone est une cellule composée d'un corps cellulaire, dendrites, axone et synapse.

- Le corps cellulaire contient le noyau du neurone ainsi que la machine biochimique nécessaire à la synthèse des enzymes. Sa taille est de quelques microns de diamètre
- Les dendrites se sont de fines extensions tubulaires qui se ramifie autour du neurone et forme une sorte de vaste arborescence. Les signaux envoyés au neurone sont captés par les dendrites.
- L'axone : c'est le long de l'axone que les signaux partent du neurone. Contrairement aux dendrites qui se ramifient autour du neurone, l'axone est plus long est se ramifie a son extrémité ou il se connecte aux dendrites des autres neurones.
- Une synapse est une jonction entre deux neurones, et généralement entre l'axone d'un neurone et un dendrite d'un autre neurone.

Le corps cellulaire se ramifie pour former les dendrites, c'est par ces dernières que l'information est acheminée de l'extérieur vers le corps cellulaire où elle est traitée. L'information traitée par le neurone chemine ensuite le long de l'axone (unique) pour être transmise aux autres neurones (Hérault 1994) (Touzet 1990).

I-3-2-Définition d'un neurone artificiel

Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones amonts. A chacune de ces entrées est associé un poids (paramètre) *w*, représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals.

Les réseaux de neurones formels sont issus d'une tentative de modélisation mathématique du cerveau humain, cette modélisation consiste à mettre en œuvre un système de réseau de neurones

artificiels qu'il existe pour chaque élément composant le neurone biologique, un élément correspondant dans le neurone artificiel, comme le montre clairement la figure I. 1.

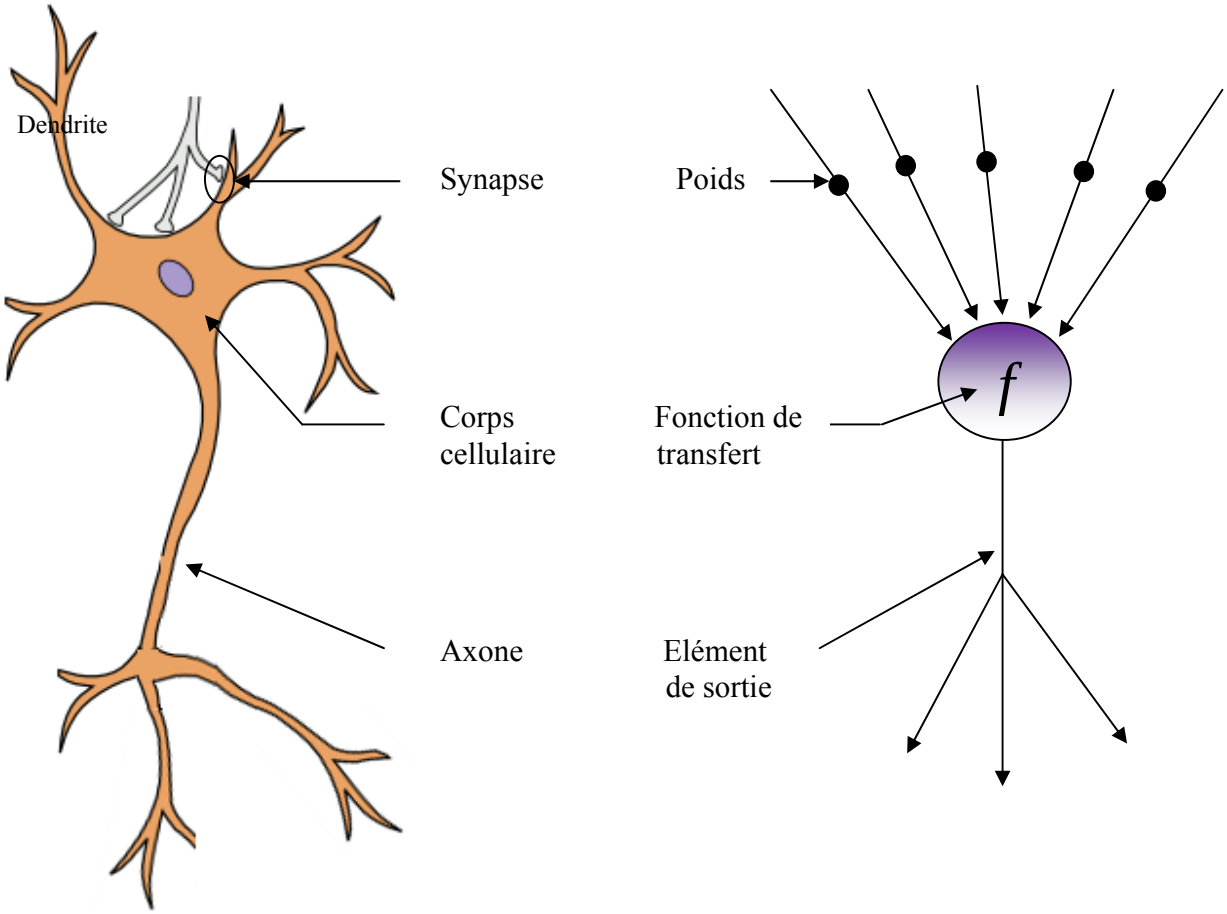

Figure. I.1 : *neurone biologique et neurone artificiel*

I-3-3-Définition d'un réseau de neurones

Dans le monde biologique plusieurs structures neuronales existent et chaque structure est dotée d'une fonction particulière. Le neurone reçoit en continu des entrées, ces informations sont interprétées au niveau du corps cellulaire du neurone. La réponse à ces signaux est envoyée a travers l'axone qui fait synapse sur des milliers d'autres neurones (Touzet 1992).

Pour les réseaux de neurones artificiels, Kohonen propose la définition suivante : "Les RNA sont des réseaux massivement connectés en parallèle d'éléments simples (habituellement

adaptatifs) et leur organisation hiérarchique. Ils sont sensés interagir avec les objets du monde réel de la même manière que les systèmes nerveux biologiques"

D'une autre manière, un réseau de neurones artificiels (RNA), est un ensemble de neurones formels associés en couches (ou sous groupes) et fonctionnant en parallèle, chaque neurone dans une couche est connecté à tous les neurones de la couche suivante. Dans un réseau de neurones l'information se propage, de la couche d'entrée à la couche de sortie, en passant par aucune où plusieurs couches intermédiaires dites couches cachées. La figure. I.2 montre un RN biologique et un RN Artificiel d'une couche d'entrée, d'une couche cachée et d'une couche de sortie.

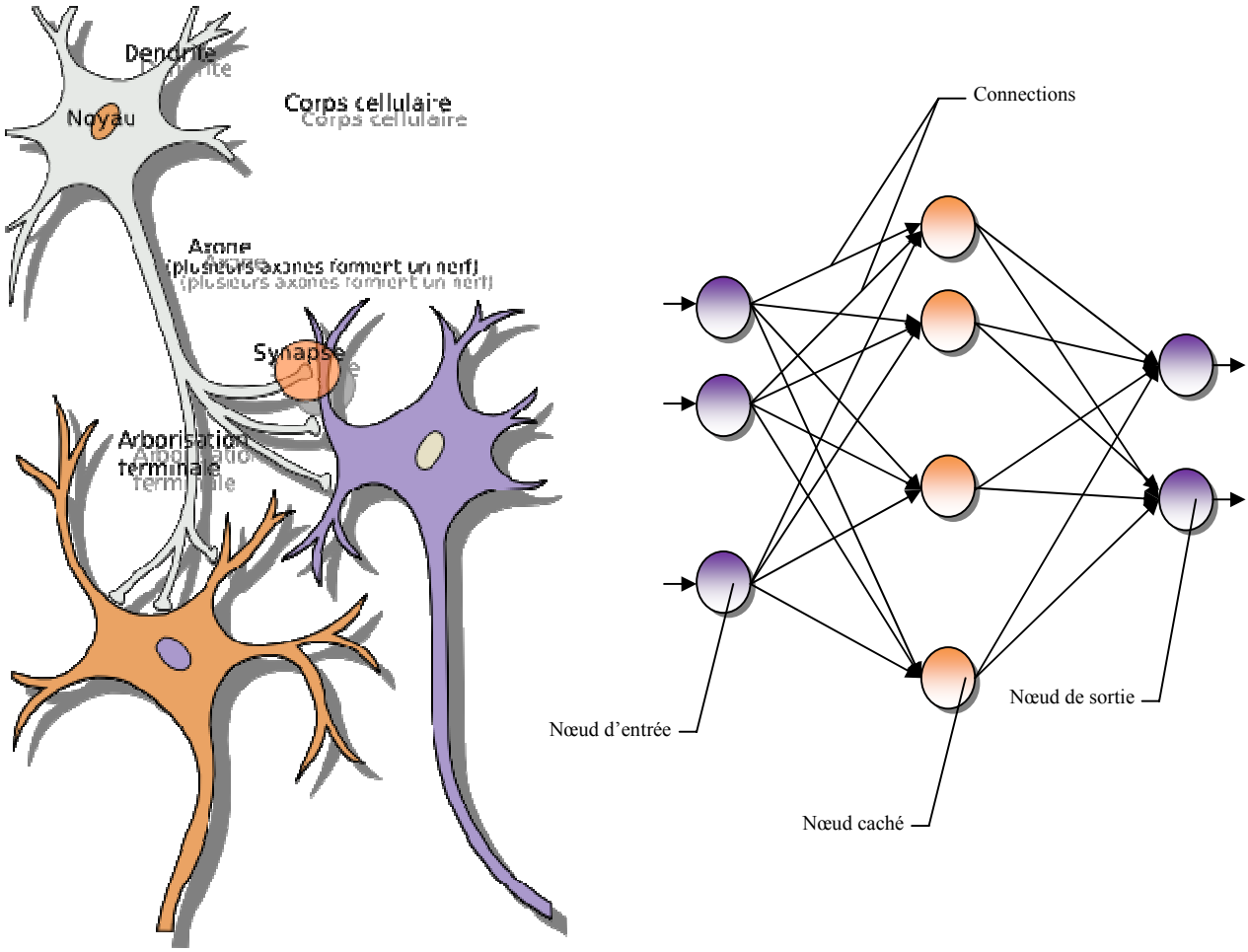

Figure. I.2 : Réseau *de neurones biologiques (à gauche), réseau de neurones artificiels (à droite)*

I-3-4-Principe de fonctionnement d'un neurone formel

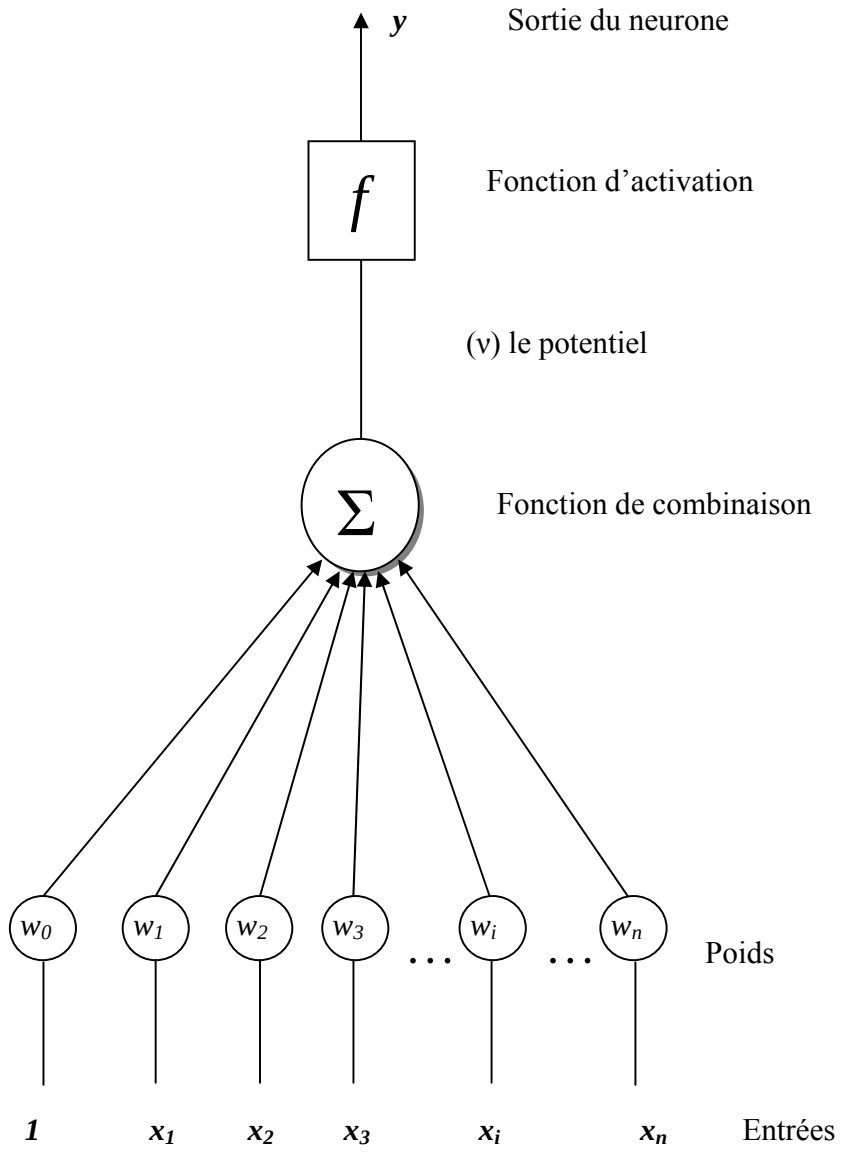

Figure. I.3 : neurone mathématique

Un neurone réalise une fonction non linéaire, paramétrée, à valeurs bornées entre les entrées et la sortie. Autrement dit un neurone réalise une fonction non linéaire d'une combinaison des entrées $\{x_i\}$ pondérées par les paramètres (où poids) $\{w_i\}$. La combinaison linéaire est

appelée potentiel (ν), à laquelle s'ajoute un terme constant ou « biais » (Borne et al. 2007). Ce principe est illustré sur la figure. I.3

Le potentiel d'un neurone est donc :

$$\nu = w_0 + \sum_{i=1}^{n-1} w_i x_i$$

La sortie du neurone sera donc :

$$y = f(\nu) = f(w_0 + \sum_{i=1}^{n-1} w_i x_i)$$

Tel que n est le nombre d'entrées du neurone et f la fonction d'activation.

- **La fonction d'activation** :

La fonction d'activation (où fonction de transfert) permet d'introduire un seuil et une saturation. Il existe différentes formes des fonctions d'activation la plupart sont continues, les plus courantes sont les sigmoïdes unipolaire et bipolaire ainsi que la tangente hyperbolique. Voici quelques exemples de fonction d'activation courante.

- **Sigmoïde unipolaire**

$$f_1(x) = \frac{1}{1+e^{-x}}$$

$$f_1'(x) = f_1(x)[1 - f_1(x)]$$

- **Sigmoïde bipolaire**

$$f_2(x) = \frac{1-e^{-x}}{1+e^{-x}} = 2[f_2(x) - 0.5]$$

$$f_2'(x) = \frac{1}{2}[1 + f_2(x)][1 - f_2(x)]$$

- **Tangente hyperbolique**

$$f_3(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$f_3'(x) = \frac{4}{(e^{x} + e^{-x})^2}$$

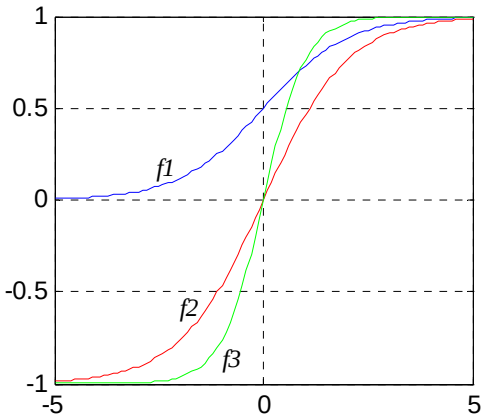

Figure. I.4 : fonctions d'activations

Les autres fonctions d'activations souvent utilisées sont récapitulées dans le tableau. I.1.

Non de la fonction	Relation entrée/sortie	Icône	Matlab
seuil	y=0 si x≤0 y=1 si x≥0		hardlim
Seuil symétrique	y= -1 si x<0 y=1 si x≥0		hardlims
linéaire	y=x		purelin
Linéaire saturée symétrique	y= -1 si x<-1 y=x si -1≤x≤1 y=1 si x>1		satlins
Linéaire positive	y=0 si x<0 y=x si x≥0		poslin

Tableau. I.1 : fonctions d'activations

I-4-Classification des réseaux de neurones

Historiquement, les réseaux de neurones ont été utilisés pour la classification et l'approximation des fonctions sur des données statiques, la sortie ne dépend que de l'entrée actuelle (fonctionnement combinatoire). Cependant il existe des applications où il faut prendre en compte l'aspect dynamique du problème lorsque la sortie dépend de l'entrée à l'instant *t* ainsi que les entrées et les sorties aux instants précédents.

On peut distinguer essentiellement deux types de réseaux de neurones formels : les réseaux non bouclés (*feedforward neural networks*) et les réseaux bouclés où récurrents (*feedback,* où *recurent neural networks (RNN)*)

I-4-a- Réseau non bouclé (feed-forword)

Un réseau de neurones non bouclé est constitué d'une ou plusieurs couches cachées, contenant chacune un certain nombre de neurones. Il réalise une (ou plusieurs) fonction algébrique de ses entrées par composition des fonctions réalisées par chacun de ses neurones. Dans un tel réseau, le flux de l'information circule des entrées vers les sorties sans "retour en arrière". Tout neurone dont la sortie est une sortie du réseau est appelé "neurone de sortie". Les autres, qui effectuent des calculs intermédiaires, sont appelés "neurones cachés" (Wilamowski 2009).

Un réseau de neurones non bouclé réalise donc une fonction non linéaire ψ de ses entrées paramétrée par les coefficients W du réseau :

$$O(k) = \psi(I(k);W) \quad \text{..... (I-1)}$$

Où $O(k) \in \Re^{N0}$ est le vecteur de sorties à l'instant *k*, $I(k) \in \Re^{NI}$ est le vecteur des entrées et $\psi : \Re^{NI} \rightarrow \Re^{N0}$ est la fonction non linéaire réalisée par les neurones du réseau.

Il existe plusieurs types de réseaux de neurones non bouclés : réseaux à une seule couche (perceptron) tel que l'Adaline (Ould abdeslam 1992), réseaux de neurones à plusieurs couches (figure I.7) et réseaux de neurones non bouclés complètement connectés (figure I.6).

La figure I.5 représente la forme canonique d'un réseau de neurones non bouclé et la figure I.6 montre une des architectures des réseaux non bouclé qui est le réseau de neurones complètement connecté.

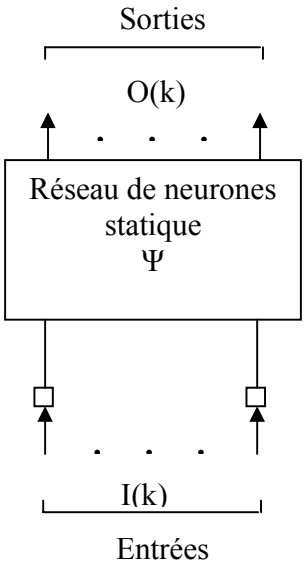

Figure. I.5 : Forme canonique d'un réseau de neurones non bouclé

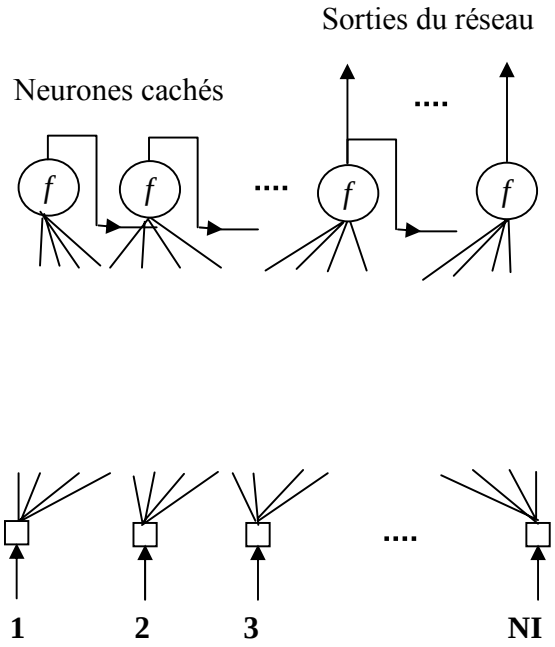

Figure 1.6 : réseau de neurones non bouclé complètement connecté

- **Réseaux de neurones à couches** : Appelés aussi "Perceptrons multicouche" où "MLP pour Multi-Layer Perceptrons". Dans une architecture de réseaux à couches les neurones cachés sont organisé en couches, les neurones d'une même couche n'étant pas connectés entre eux, de plus les connections entre deux couches non consécutives sont éliminés comme il est montré sur la figure. I.7. Une autre architecture de réseau très fréquemment utilisée est les réseaux de neurones à une seule couche cachée.

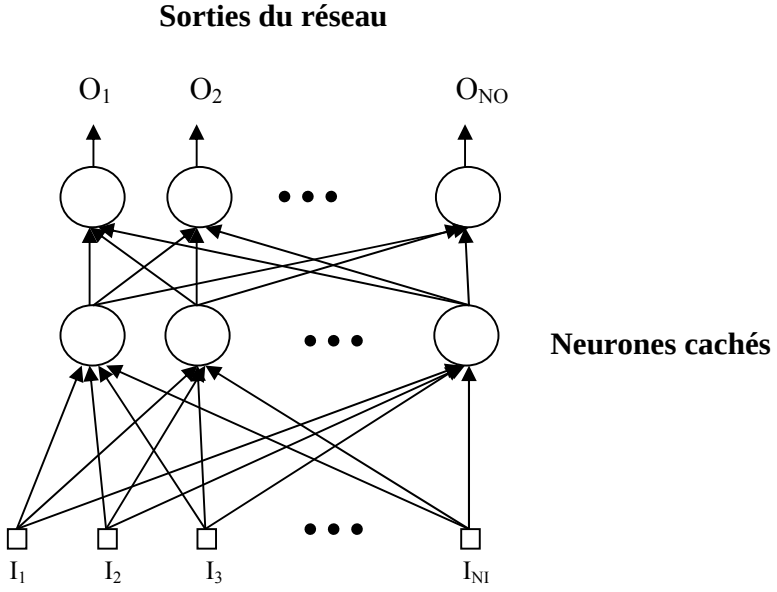

Figure. I.7 : Réseau de neurones à NI entrées, une couche cachée, NO sorties

I-4-b- Réseau bouclé (récurrent)

Ce type de réseau possède des connexions sous forme de boucles, contrairement au réseau non bouclé ou les connections sont orientées de l'entré vers la sortie du réseau. Le réseau récurrent admet tout type de connexion, c'est à dire d'un neurone à n'importe quel autre, y compris lui-même. Il est possible dans ce réseau de trouver au moins un chemin qui revient à son point de départ en suivant le sens des connections, on appelle un tel chemin *le cycle*.

Ces réseaux sont assez intéressants car leur fonctionnement est séquentiel et adopte un comportement dynamique. Ils sont utilisés essentiellement pour modéliser les systèmes récursifs, comme ils permettent des applications des problèmes réels (par exemples industriels).

Tout réseau de neurones bouclé est un système dynamique non linéaire que l'on peut mettre sous forme d'une représentation d'état appelée forme canonique.

$$\begin{cases} S(k+1) = f(S(k), I(k), W) \\ O(k) = g(S(k), I(k), W) \end{cases} \quad \text{(I-2)}$$

Où $O(k) \in \Re^{N0}$ est le vecteur de sorties à l'instant k, $I(k) \in \Re^{NI}$ est le vecteur des entrées et les fonctions, f *et* g sont des fonctions non linéaires qui peuvent être réalisées par deux réseaux de neurones non bouclés.

La figure I.8 représente la forme canonique d'un réseau de neurones bouclé.

Les modèles les plus performants et les plus connus sont, l'architecture de Jordan, c'est un multicouche, les cellules de la première couche se répartissent en deux couches : couche de contexte et couche d'entrée. La dernière couche est rebouclée sur la couche de contexte (Jordan 1986), l'architecture est montrée sur la figure I.9. et celui d'Elman (Elman 1990), (Saul et al. 2000) sont architecture est légèrement différente de celle de Jordan, il s'agit d'une structure multicouche, où les boucles de rétroaction relient la couche cachée avec les cellules d'entrées, il y a autant de cellules cachées que de cellules de contexte et chaque cellule cachée est reliée à une seule cellule de contexte, on trouve ce modèle dans plusieurs travaux de recherche (Narendra et al 1990). L'architecture du modèle d'Elman est montrée sur la figure I.10. L'algorithme d'apprentissage pour ces deux types de réseaux est la retropropagation du gradient. Il existe d'autres algorithmes d'apprentissages qui peuvent être appliqués pour des réseaux récurrents tel que : "real time recurrent learning" et "time dependent recurrent back-propagation" (Hertz et al. 1991), (Haykin et al. 1997), Backpropagation through time (Williams et al.1990).

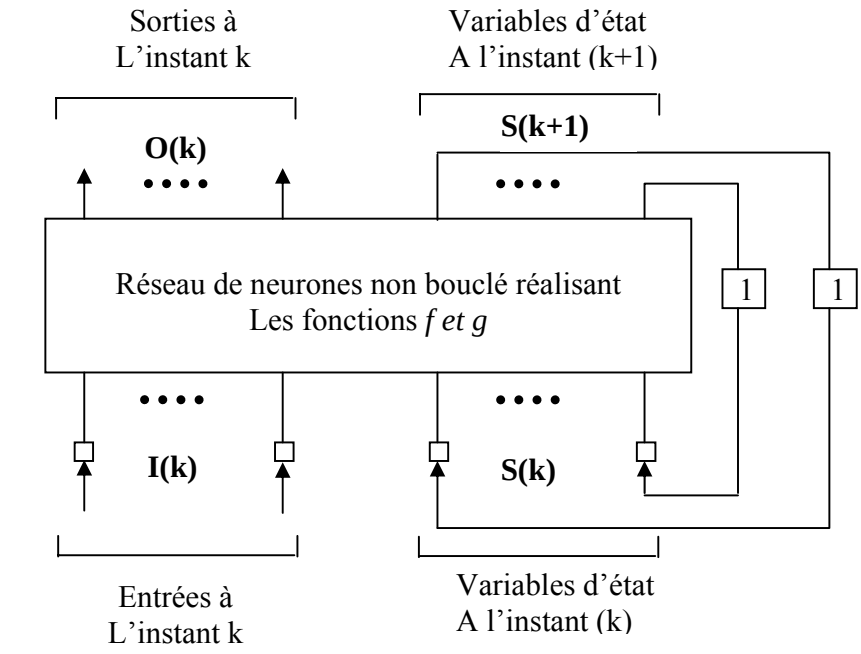

Figure. I.8 : Forme canonique d'un réseau de neurones bouclé.

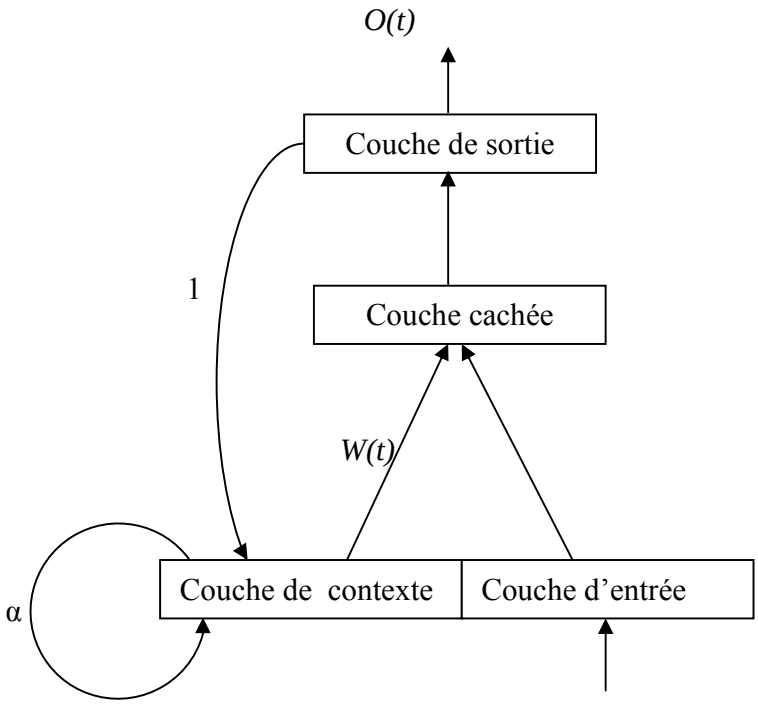

Figure. I.9 : Architecture de Jordan

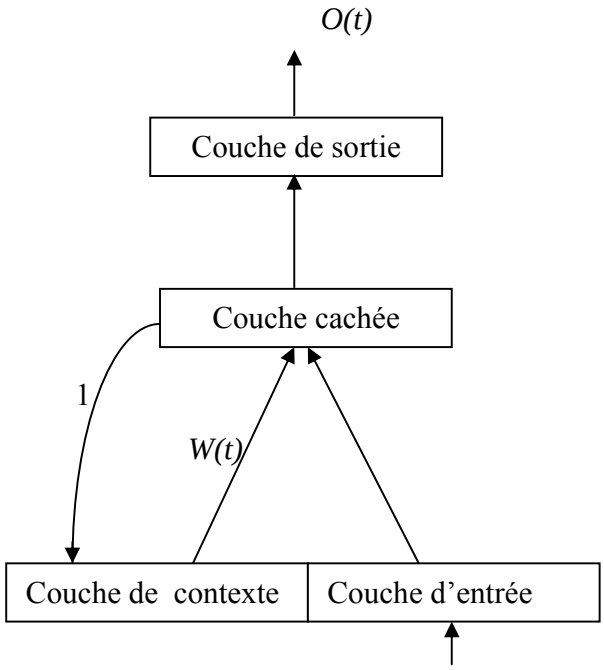

Figure. I.10 : Architecture d'Elman

- **Passage d'une modélisation d'état à une modélisation entrée /sortie**

On parle d'une modélisation entrée-sortie lorsque on définit comme vecteur d'état le vecteur composé des sorties de processus aux instants précédents, (Kotta et al. 2006), (Chen et al 2004). Cette forme peut s'écrire comme suit :

$$y(k+1) = h(y(k), y(k-1)....y(k-n), u(k), u(k-1)....u(k-m)) \quad \text{..... (I-3)}$$

Où *h* est une fonction non linéaire.

y(k), u(k) sont sorties et les entrées du processus à l'instant *k* .

La modélisation d'état d'un processus est la suivante :

$$\begin{cases} x(k+1) = f(x(k), u(k)) \\ y(k) = g(x(k)) \end{cases} \quad \text{..... (I-4)}$$

à partir de (1-3)

$$\begin{cases} y(k+1) = g(x(k+1)) \\ y(k+1) = g(f(x(k),u(k))) \\ ..... \\ y(k+m-1) = g\big(f(...f(x(k),u(k)),u(k+1))...u(k+m-1)\big) \end{cases} \quad \text{..... (I-5)}$$

Dans le cas d'un modèle linéaire d'ordre n, observable, on peut résoudre ce système pour *m=n,* et l'état *x* peut donc être calculé à partir de *n-1* valeurs de l'entrée *(u(k), …, u(k+n-2)),* et de *n* valeurs de la sortie *(yp(k), …, yp(k+n-1)).* Tout modèle d'état linéaire observable possède donc une représentation entrée-sortie (Rivals 1995).

Le modèle entrée/sortie correspondant à un processus d'ordre n=3 est :

$$y(k+1) = h(y(k), y(k-1), y(k-2), u(k), u(k-1)) \text{ ..... (I-6)}$$

Le modèle neuronal correspondant à ce processus est :

$$\hat{y}(k+1) = \Psi_{RN}(y(k), y(k-1), y(k-2), u(k), u(k-1), W) \text{ .....(I-7)}$$

La figure. I.11 montre un réseau de neurones entrée-sortie (multicouche, réseau non bouclé) correspondant à un système d'ordre *n*=3

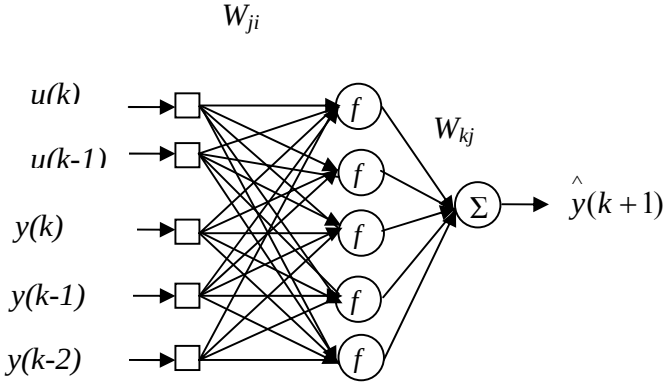

Figure. I.11 : réseau non bouclé entrée-sortie

Dans le cas d'un modèle d'état non linéaire d'ordre n_x, le modèle entrée/sortie correspondant est :

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

Où p est un entier compris entre n_x qui est l'ordre du modèle d'état, et $2n_x+1$.

I-5-Propriétés des réseaux de neurones

a) - *L'approximation universelle* est une propriété fondamentale des réseaux de neurones bouclés et non bouclés (Thiria 1997).

- Pour les réseaux non bouclés : d'après cette propriété, le comportement de tout système statique peut être rapproché par un réseau de neurones non bouclé.

Propriété : *Toute fonction bornée suffisamment régulière peut être approchée uniformément, avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire.*

- Pour les réseaux bouclés :

  Soit le système dynamique suivant :

$$\begin{cases} x_p(k+1) = \phi\left(x_p(k), u(k)\right) \\ y_p(k) = \gamma\left(x_p(k), u(k)\right) \end{cases} \quad \text{..... (I-8)}$$

ϕ, γ , sont des fonctions non linéaires qui peuvent être approchées par deux réseaux de neurones non bouclés. La propriété d'approximation universelle pour les réseaux bouclés peut s'énoncer de la manière suivante :

- pour tout système dynamique défini par (I-3),
- pour toute précision désirée ε , sur une séquence de taille finie [0 ;T],
- pour des entrées bornées $\{u(k)\}$ et un état initial $x_p(0)$, il existe un réseau de neurones bouclés décrit par (I-2) qui approche le comportement entrée-sortie du système avec la précision ε sur l'intervalle [0 ;T].

b)- *La parcimonie* : le but de la modélisation avec les réseaux de neurones est d'obtenir un modèle qui reproduit au mieux le comportement d'un processus avec un nombre minimum de paramètres à ajustés, (Hornic 1994) a montré que :

Si le résultat de l'approximation (c'est-à-dire la sortie de réseau de neurones) est une fonction non linéaire des paramètres ajustables, elle est plus parcimonieuse que si elle est une fonction linéaire de ces paramètres. De plus, pour les réseaux de neurones à fonction d'activation sigmoïdale, l'erreur commise dans l'approximation varie comme l'inverse du nombre de neurones cachés, et elle est indépendante de nombre de variables de la fonction à approchée, par conséquent, pour une précision donnée, donc pour un nombre de neurones cachés donné, le nombre de paramètres du réseau est proportionnel au nombre de variable de la fonction à approchée.

Ainsi l'avantage des réseaux de neurones par rapport aux approximateurs usuels est d'autant plus sensible que le nombre de variable de la fonction à approchée est grand, donc pour des problèmes a trois variables ou plus il est généralement avantageux d'utiliser les réseaux de neurones.

I-6-Apprentissage des réseaux de neurones

L'apprentissage est vraisemblablement la propriété la plus intéressante des réseaux de neurones, c'est la phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré.

On peut distinguer deux types d'apprentissage principaux : l'apprentissage supervisé et l'apprentissage non supervisé.

- **Apprentissage non supervisé :** L'apprentissage est non supervisé lorsque seules les valeurs d'entrée sont disponibles. Dans ce cas, les exemples présentés à l'entrée provoquent une auto adaptation du réseau afin de produire des valeurs de sortie qui soient proche à des valeurs d'entrée similaires (de même nature). Les réseaux à apprentissage non supervisé les plus étudiés et utilisés sont les « cartes auto-organisatrices » de Kohonen.

- **Apprentissage supervisé :** L'apprentissage est dit supervisé lorsque les exemples sont constitués de couples de valeurs du type (valeur d'entrée, valeur de sortie désirée), qui constituent un ensemble d'apprentissage. L'apprentissage supervisé est le type

d'apprentissage le plus utilisé. Pour ce type d'apprentissage la règle la plus utilisée est celle de Widrow-Hoff (Norgaard et al. 2000), (Borne et al. 2007).

Le but de l'apprentissage est d'ajuster les poids des neurones, cette tâche nécessite des exemples désignés aussi sous l'appellation d'échantillons d'apprentissage ainsi qu'un algorithme d'apprentissage.

Soit un ensemble de N mesures de la sortie $\left\{Y_p^k\right\}$ pour k=1 à N, correspondant à N valeurs du vecteur d'entrée $\left\{X^k = \left[X_1^k X_2^k ... X_n^k\right]\right\}$, l'ensemble des couples $\left\{X^k, Y_p^k\right\}$ est appelé, ensemble d'apprentissage. Afin de déterminer la meilleure valeur du vecteur paramètre W on doit minimiser le coût, qui est une fonction des écarts entre les mesures $\left\{Y_p^k\right\}$ et les valeurs fournies par le modèle $\left\{Y^k(W)\right\}$ (réseau de neurones). On choisit généralement une fonction de coût quadratique telle que :

$$J(W) = \sum_{k=1}^{N} \frac{1}{2}\left(Y_p^k - Y^k(W)\right)^2 \quad \text{..... (I-9)}$$

L'apprentissage des modèles nécessite des algorithmes de minimisation du coût $J(W)$ qui permet d'avoir un vecteur paramètre qui garantit une sortie de réseau égale ou proche de la sortie désirée, on distingue deux types d'algorithme:

- Un algorithme de minimisation itératif recherche le minimum du coût en modifiant les paramètres à chaque itération, à partir du calcul de coût total, parmi ces algorithmes on trouve l'algorithme du gradient et l'algorithme de Newton.
- Un algorithme de minimisation récursif effectue des modification des paramètres calculées à l'aide d'un coût portant sur un nombre limité d'exemple (coût partiel), dans ce cas, une modification des paramètres est effectuée après la présentation de chaque exemple, parmi ces algorithmes on cite, l'algorithme de Windrow Hoff, la méthode des moindres carrés récursifs, on peut également utiliser la récursion de filtre de Kalman étendu.

L'estimation des paramètres peut avoir lieu hors ligne (système non adaptatif) où en ligne (système adaptatif) (Personnaz et al. 2003):

- Dans le cas de l'apprentissage hors ligne, l'expérience qui a fourni les séquences d'apprentissage a été effectuée auparavant, donc l'ensemble d'apprentissage est de taille finie. Dans ce cas l'algorithme d'apprentissage peut indifféremment être itératif ou récursif.
- En utilisant l'apprentissage en ligne, les séquences d'apprentissage sont de taille infinie, puisque pour un système adaptatif les valeurs des paramètres des modules de calcul sont modifiés au cours du temps, pendant l'utilisation du système en fonction de l'environnement. Ainsi en traitement du signal, il est nécessaire de faire des mises à jour des paramètres des filtres en fonction de l'arrivée des nouvelles mesures. Dans ce cas, il est nécessaire d'utiliser un algorithme récursif.

  La figure suivante montre les deux systèmes d'apprentissage en ligne ou hors ligne d'un RNA pour l'estimation d'un processus.

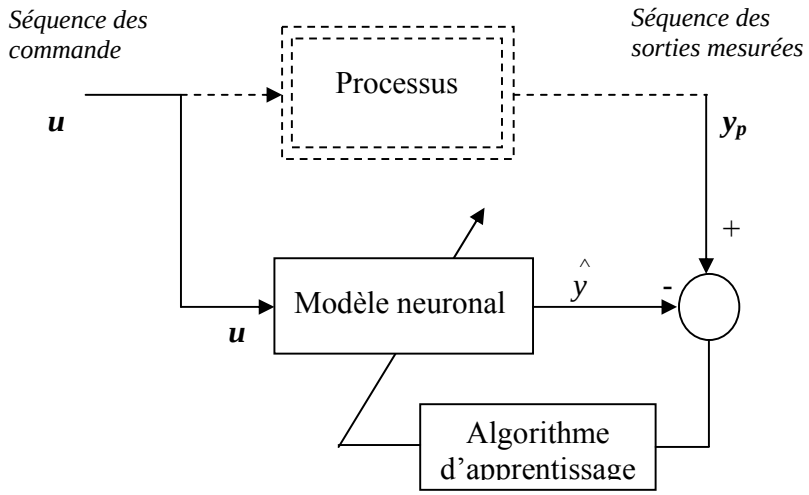

Figure. I.12 : système d'apprentissage en ligne ou hors ligne d'un modèle neuronal

I-7- Conclusion

Dans ce chapitre nous avons présenté les généralités sur les réseaux de neurones artificiels, des définitions de base ainsi que quelques architectures neuronales et les techniques d'apprentissage. Cette étude nous a permis de découvrir les réseaux de neurones artificiels, leur utilisation et leur fonctionnement ainsi que leur apprentissage qui se fait sur un ensemble de couples entrées-sorties afin d'estimer les paramètres du réseau en minimisant l'erreur quadratique moyenne. Le grand avantage des réseaux de neurones réside dans leurs capacité d'apprentissage ce qui permet de régler les problèmes, notamment de modélisation et d'identification des processus statiques ou dynamiques, sans nécessité de règles mathématiques complexes, de plus l'approximation à l'aide de réseaux de neurones nécessite en général moins de paramètres ajustables que les approximateurs usuels. Dans le chapitre qui suit nous allons voir un nouveau type de réseaux dynamiques (récurrents) qui est le réseau de neurones à espace d'état.

Chapitre II

II-1-Introduction

En 1982 J. Hopfield présente pour la première fois un réseau de neurones complètement rebouclé dont il analyse la dynamique. Cette découverte a permis de relancer l'intérêt dans le domaine de la recherche sur les réseaux de neurones artificiels. De nouveaux réseaux de neurones ont pu être mis au point et les appellations d'architectures dynamiques ou de Réseaux Neuronaux Récurrents (RNR) sont apparues. Parmi ces nouvelles architectures de réseaux de neurones récurrents, on compte les réseaux de Jordan (Jordan 1986), (Touzet 1992), les réseaux d'Elman (Elman 1988), les réseaux multicouches dynamiques (avec des retours de la sortie du réseau vers les entrées), (Wen 2002), State-Space Dynamic Neural Network (SSDNN) (Cao et al. 2006). Le réseau de neurones récurrent est considéré actuellement comme faisant partie des meilleures méthodes de traitement séquentiel de l'information. Des études théoriques ont même montré que les réseaux de neurones récurrents sont des approximateurs universels des systèmes dynamiques vu leurs capacités intrinsèques d'introduire la notion du temps.

Ce chapitre introduit un type de réseaux de neurones récurrents bien spécifique qui sert à la modélisation et la commande numérique des processus. Ce réseau de neurones artificiel est le réseau de neurones à espace d'état, appelé en anglais State Space Neural Network (SSNN). Mais avant d'introduire les SSNN, nous rappelons de quelques notions de base de la représentation d'état des systèmes.

Nous nous plaçons dans le cadre général des modèles à temps discret des processus, cadre qui se prête bien à la commande numérique d'une part, et à l'utilisation de réseaux de neurones formels d'autre part.

II-2- Représentation d'état

La représentation d'état des systèmes est un outil très puissant permettant de modéliser le fonctionnement de systèmes linéaires ou non linéaires et de conserver une représentation temporelle des phénomènes. Contrairement à la fonction de transfert qui est une relation entrée/sortie qui n'apporte aucune connaissance sur la structure interne d'un système, la représentation d'état contient des informations accessibles à la mesure et directement liées aux grandeurs physiques des systèmes. De ce fait elle offre des possibilités nouvelles en termes d'analyse.

La représentation d'état d'un système n'est pas unique, bien au contraire pour un système donné il en existe une infinité.

La forme générale des équations d'état en temps continu est la suivante :

$$\begin{cases} \dot{\vec{x}}(t) = f(\vec{x}(t) + \vec{u}(t)) \\ \vec{y} = g(\vec{x}(t)) \end{cases} \quad \text{.....} \quad \text{(II-1)}$$

Où

$\vec{u} \in \Re^n$ est le vecteur entrée (commandes) du système, $\vec{y} \in \Re^m$ est le vecteur de sortie du système et peut être mesurée, le vecteur $\vec{x} \in \Re^s$ est appelé l'état du système et il représente la mémoire du système, c'est-à-dire, l'ensemble des informations donc le système a besoin pour prédire son propre avenir, pour une entrée *u(t)* connue. La première équation de (II-1) est une équation différentielle s'appelle équation d'évolution (où équation d'état), la deuxième s'appelle équation d'observation (où équation de sortie).

Dans le cas des systèmes linéaires *f* et *g* sont linéaires, ainsi la représentation d'état (II-1) devient :

$$\begin{cases} \dot{\vec{x}}(t) = A\vec{x}(t) + B\vec{u}(t) \\ \vec{y} = C\vec{x}(t) \end{cases} \quad \text{.....} \quad \text{(II-2)}$$

Les matrices *A, B et C* sont appelées matrice d'évolution, de commande et d'observation.

I-2-1- Commandabilité

Un système est complètement contrôlable s'il existe un signal de contrôle $u(t)$ qui permet le transfert de l'état initial $x(t_0)$ vers une location désirée $x(t)$, en un temps finie T tel que : $t_0 \leq t \leq T$

Le système (II-2) est complètement commandable si le déterminant de la matrice de contrôlabilité *Pc* est non nul (la matrice P_c est de rang plein n). La matrice *Pc* se calcule de la manière suivante :

$$P_c = \begin{bmatrix} B & AB & A^2B \ldots & A^{n-1}B \end{bmatrix}$$

I-2-2-Observabilité

Un système est complètement observable si et seulement s'il existe un temps fini T tel que la valeur initiale de l'état $x(0)$ peut être déterminé à partir de l'histoire $y(t)$ connaissant la commande $u(t)$.

Le système (II-2) est observable si le déterminant de la matrice d'observabilité Po est non nul. Cette matrice est calculée ainsi :

$$P_o = \begin{bmatrix} C \\ CA \\ . \\ . \\ CA^{n-1} \end{bmatrix}$$

Où bien le système (II-2) est observable si le rang de la matrice P_0 égal à n.

I-2-3-Discrétisation et l'influence de la période d'échantillonnage

Il est parfois utile de travailler dans le domaine discret, et considéré que le temps qui régit le système est discret (k), en effet, si l'univers que nous considérons est l'ordinateur. La discrétisation est un moyen qui permet de passer d'une représentation d'état continu à une représentation dite échantillonnée ou discrète du système.

Considérons un système continu linéaire définit par l'équation (II-2). Deux éléments important interviennent dans la discrétisation de ce système :

- Les échantillonneurs qui captent au rythme de la période d'échantillonnage la valeur du signal.
- Les bloqueurs, généralement d'ordre zéro, qui permettent de maintenir la commande u constante sur l'intervalle [kTe, (k+1)Te] comme le montre la figure suivante :

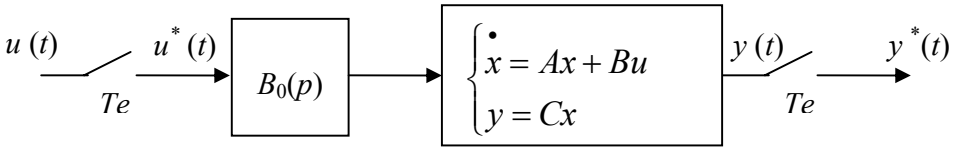

Figure. II.1 : système continu échantillonné

La représentation d'état échantillonnée est alors :

$$\begin{cases} x[(k+1)Te] = A_D(Te)x(kTe) + B_D u(kTe) \\ y(kTe) = C_D x(kTe) \end{cases} \quad \text{.....(II-3)}$$

Où

$$A_D(Te) = e^{ATe}$$

$$B_D(Te) = \int_0^{Te} e^{A\mu} B d\mu$$

$$C_D = C$$

On a intérêt, si on veut restituer au mieux le comportement du système en passant du domaine continu au discret, à bien choisir la période d'échantillonnage qui représente la durée séparant deux prises de mesure.

D'après le théorème de Shannon, si un signal continu possède un spectre de fréquence maximale *Fmax*, il est possible de l'échantillonner sans perte d'information si la période d'échantillonnage *fe* est choisie telle que : *fe>2Fmax.*

Le mauvais choix de la période d'échantillonnage peut générer une perte de stabilité ou perte d'observabilité pour le système échantillonné (Crosnier et al. 2001).

Dans une représentation d'état en temps discret on exprime l'état du système à un instant donné en fonction du signal d'entrée et de son état passé. La forme générale des équations d'état en temps discret est donnée soit par (II-4), soit par (II-5), selon que le système est linéaire ou non linéaire. Ainsi :

- Dans le cas des systèmes non linéaires :

$$\begin{cases} x(k+1) = F(x(k), u(k)) \\ y(k) = G(x(k)) \end{cases} \quad \text{.....(II-4)}$$

Où *F et G* sont des fonctions non linéaires représentatives de l'évolution du système dynamique dans le temps,

- Dans le cas des systèmes linéaires :

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) \end{cases} \quad \text{……..……..(II-5)}$$

On peut remarquer que la modélisation donnée par (II-5) est un cas particulier de la forme générale d'un système non linéaire donnée par (II-4). La modélisation donnée par (II-5) peut être représentée par le schéma de la Figure. II.2.

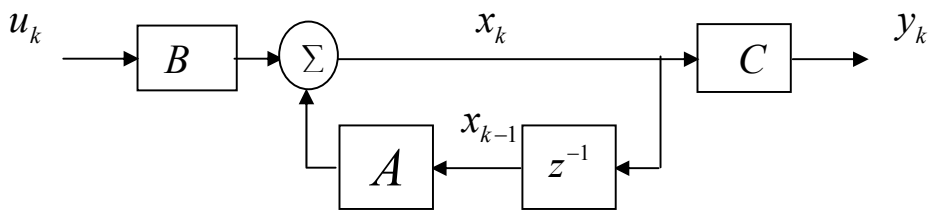

Figure. II.2 : Représentation schématique d'une modélisation d'état en temps discret

II-3- Définition d'un réseau de neurones à espace d'état (SSNN)

Le réseau de neurones à espace d'état (State Space Neural Network, ou SSNN) est un type de RNR dont la structure reflète exactement une modélisation dans l'espace d'état d'un système dynamique non linéaire (Zamarreno et al. 1999). L'idée d'employer les modèles neuronaux à espace d'état pour l'identification des systèmes non linéaires n'est pas nouvelle dans la littérature, il a été employé avec succès dans certain nombre d'applications industrielles. On peut par exemple citer la prévision de la température de l'environnement à court terme, ou la modélisation d'une section de diffusion ou l'extraction du jus brut des betteraves dans une usine de sucre (Zamarreno et al. 2000). Dans ces deux cas précis, l'utilisation du SSNN a facilité l'apprentissage et par conséquent, une base de données d'apprentissage plus petite a pu être employée par rapport à celles d'autres réseaux de neurones. Les résultats obtenus sont tout à fait satisfaisant lorsqu'on les compare à ceux des autres méthodes.

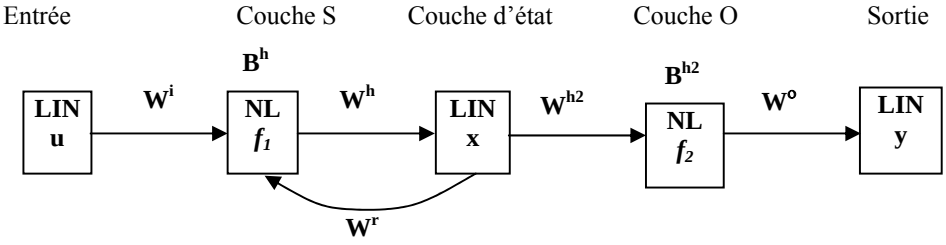

Figure. II.3 : schéma bloc d'un réseau de neurones à espace d'état (SSNN).

LIN : éléments linéaires.

f_1 , f_2 : fonctions non linéaires.

La figure. II.3 représente un SSNN dont la structure reflète exactement une représentation d'état non-linéaire. On peut dire même qu'elle est équivalente à un système non linéaire déterministe dont la représentation d'état est la suivante (Zamarreno et al. 1998) :

$$\begin{cases} \vec{x}(k+1) = F(\vec{x}, \vec{u}) \\ \vec{y}(k) = G(\vec{x}(k)) \end{cases} \quad \text{(II-6)}$$

Où k représente l'instant discret $t=kT$, T étant le pas d'échantillonnage.

$\vec{x} \in \Re^s$ est le vecteur d'état, $\vec{u} \in \Re^n$ est le vecteur d'entrées (commandes), $\vec{y} \in \Re^m$ est le vecteur de sorties.

$F : \Re^s \times \Re^n \to \Re^s$ et $G : \Re^s \to \Re^m$ sont deux fonctions non linéaires statiques.

Chaque bloc dans la figure (II. 3) représente une couche de neurones du SSNN :

- Couche d'entrée : cette couche fournit les entrées de commande à chaque neurone de la prochaine couche, chaque entrée de commande $u(k)$ est pondérée par le poids W^i correspondant et est à injecter à tous les neurones de la prochaine couche.
- Couche cachée (S) : la lettre (S) vient du mot "State", c'est la couche cachée avant la couche d'état, elle représente la fonction non linéaire qui indique le comportement des états.
- Couche d'état : chaque neurone de cette couche représente un état, la sortie du neurone est la valeur de l'état $x(k)$.

- Couche cachée (O) : la lettre (O) vient du mot "Output", c'est la couche avant la couche de sortie. Elle représente la fonction non linéaire qui relie les sorties du réseau aux états.
- Couche de sortie : relie les signaux de la couche cachée (O) à chaque neurone de sortie et ces sorties sont finalement les sortie du SSNN.

Le comportement de ce système est non linéaire (dans le cas général), l'état évolue en fonction de l'état précédent et en fonction des excitations externes (des entrées et des perturbations). Les sorties du système ($y(k)$) sont simplement une fonction des états. Le comportement du système est conditionné au départ par des conditions initiales.

Il a été prouvé que n'importe quelle fonction non linéaire peut être estimée par un réseau de neurones contenant une seule couche cachée composée de neurones dont la fonction de transfert est bouclée (Hornik et al. 1989) Ainsi, deux réseaux de neurones interconnectés peuvent représenter le système d'équation (II-6), l'un représente la fonction qui donne le comportement d'état et l'autre représente la fonction qui relie des sorties aux états. Ainsi le modèle (II-6) peut être représenté comme suit (Zamarreno et al. 2000) :

$$\begin{cases} \hat{x}(k+1) = W^h . f_1(W^r . \hat{x}(k) + W^i . \vec{u}(k) + B^h) \\ \hat{y}(k) = W^0 . f_2(W^{h2} . \hat{x}(k) + B^{h2}) \end{cases} \quad \text{(II-7)}$$

où

- W^h, W^r, W^i, W^0 et W^{h2} sont des matrices de dimensions $s \times h, h \times s, h \times n, m \times h2$ et $h2 \times s$ respectivement·
- B^h et B^{h2} sont deux vecteurs de biais avec h et $h2$, éléments qui ont pour rôle d'augmenter ou d'abaisser l'entrée nette de la fonction d'activation.
- f_1 et f_2 sont deux fonctions non-linéaires dans un cadre le plus général possible, elles sont souvent sigmoïdales.

La figure. II.4 est une réalisation de l'équation (II-7), elle montre clairement les couches du réseau ainsi que les liens entre les neurones.

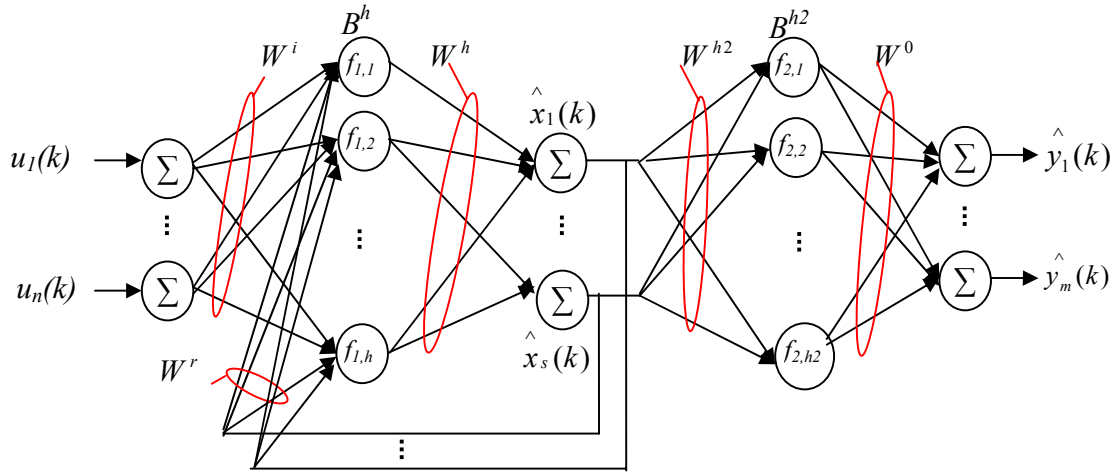

Figure. II.4 : Réseau de neurones à espace d'état

La différence entre ce type de réseaux et les réseaux de neurones classiques, tels que le réseaux multicouche (MLP) qui a été abordé dans le chapitre précédent, c'est que le réseau multicouche apprend et estime une fonction ou une relation entre un espace d'entrée (par exemple l'espace des entrées de commande *u(k)* d'un système) et un espace de sortie (par exemple l'espace des sorties *y(k)* d'un système) sans prendre en compte le comportement interne du système. Par contre, la structure d'un SSNN est identique à la représentation schématique d'une modélisation d'état en temps discret montrée sur la figure II.1. La structure du SSNN possède de ce fait une contrainte imposée par le système à modéliser. Le SSNN cherche à estimer les deux sous fonctions qui représentent les transformations non linéaires de la représentation d'état. Le SSNN se doit d'apprendre donc une relation entre un espace d'entrée (entrées de commande *u(k)*), un espace interne (les variables d'état *x(k)*) et un espace de sortie (sorties *y(k)*) comme le montre la figure (II.2) , il est très difficile d'estimer les sous-fonctions si la grandeur intermédiaire n'est pas accessible,

Un réseau de neurones du type SSNN comporte deux avantages principaux :

1. étant un modèle neuronal, il a la flexibilité de représenter n'importe quelle fonction non linéaire,
2. étant un modèle d'espace d'état, le nombre de connections extérieurs est minimal donc le nombre de paramètres a ajustés est minimal. Les entrées/sorties du modèle neuronal sont les entrées/sorties du système physique.

Il a plusieurs autres avantages à utiliser un SSNN plutôt qu'un multicouche. On peut citer la possibilité d'utiliser les connaissances dont on dispose à priori sur le processus pour le choix de la structure du réseau de neurones ou pour le choix des valeurs initiales de certains paramètres. Le réseau de neurones à espace d'état permet également d'obtenir les propriétés du système, tel que la stabilité et le temps de réponse à partir des poids du réseau de neurones.

L'inconvénient principal du réseau SSNN provient de son apprentissage qui est rendu plus difficile notamment si l'état n'est pas accessible à la mesure (Zamarreno 1999).

II-4-Fonctionnement et conditions initiales

L'architecture d'un réseau de neurones à espace d'état est très proche de la représentation d'état qui est très largement utilisée en automatique pour modéliser un système. La représentation d'état permet de déterminer l'état d'un système à un moment donné à partir des informations de l'instant précédent. Comme le montre l'équation (II-7), la première sortie $\hat{y}(0)$ dépend de l'état $\hat{x}(0)$ et une fois que $\hat{y}(0)$ est obtenu, alors $\hat{x}(1)$ peut être calculé en fonction de l'état $\hat{x}(0)$ et de l'entré de commande $\hat{u}(0)$. Ce processus est ainsi répété itérativement. On appelle $\hat{x}(0)$ l'état initial, fixer les valeurs de l'état initial est un choix crucial. Ainsi, si les valeurs initiales de l'état du système sont connues, le problème est réglé et le réseau de neurones peut être initialisé avec ces valeurs. Lorsqu'elles ne sont pas connues, il est possible de prendre des valeurs initiales nulles (Zamarreno et al. 2000).

La figure. II.5 montre le fonctionnement itératif du SSNN. On voit qu'à chaque instant, les signaux d'états sont propagés vers la sortie, et que les états sont mis à jour en fonction des états précédents et des valeurs présentes des entrées u_k.

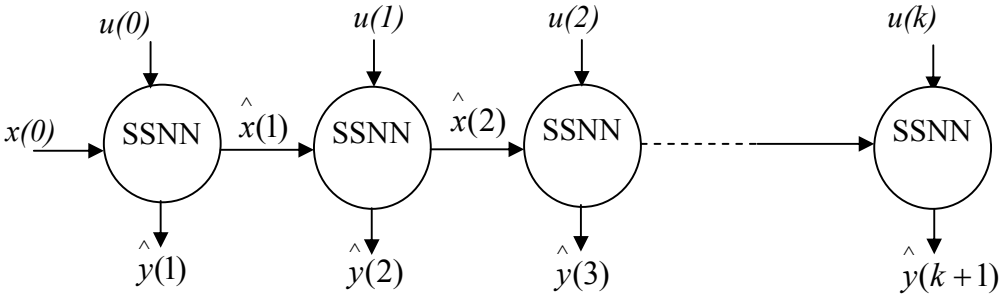

Figure. II.5 : Le principe itératif du SSNN

II-5-Apprentissage des réseaux de neurones à espace d'état

L'apprentissage, c'est-à-dire la détermination des poids neuronaux ou le calcul des paramètres des neurones est une étape fondamentale dans l'utilisation des techniques neuromimétiques. Ces calculs utilisent des ensembles de données issus du système à apprendre pour chercher les meilleurs poids possibles afin que le réseau de neurones reproduise le comportement du système. Plusieurs techniques d'apprentissage peuvent être appliquées au SSNN. Certaines de ces techniques sont basées sur le calcul d'un gradient, tel que : la méthode de backpropagation, fast forward propagation (Pearlmuter 1995), l'apprentissage récurrent à temps réel (real time recurrent learning) (Narendra et al 1990). D'autres nouveaux algorithmes d'apprentissages des SSNN sont apparus, tel que l'algorithme d'apprentissage basé sur l'échange des rôles des états du réseau et la matrice de poids (Pan et al. 2004), l'algorithme Delayed link rule (Zamarreno et al. 1998), la méthode Modified Random Optimisation ou en utilisant la théorie du filtre de Kalman et le filtre du Kalman étendu (Wira 2002).

Le réseau de neurones multicouche a reçu une attention considérable vu ses capacités d'identification. Différentes études ont prouvé sa capacité de prévoir les sorties de beaucoup de processus dynamiques (Åkesson et al. 2006), (Bose et al. 2007). Le SSNN est un enchainement de deux sous réseaux à une seule couche cachée, le premier est un réseau bouclé correspondant à l'équation d'état $\vec{x}(k+1) = F(\vec{x}, \vec{u})$, il est montré sur la figure. II.6 et le deuxième est un réseau non bouclé (statique) correspondant à l'équation de sortie $\vec{y}(k) = G(\vec{x}(k))$ montré sur la figure.II.7.

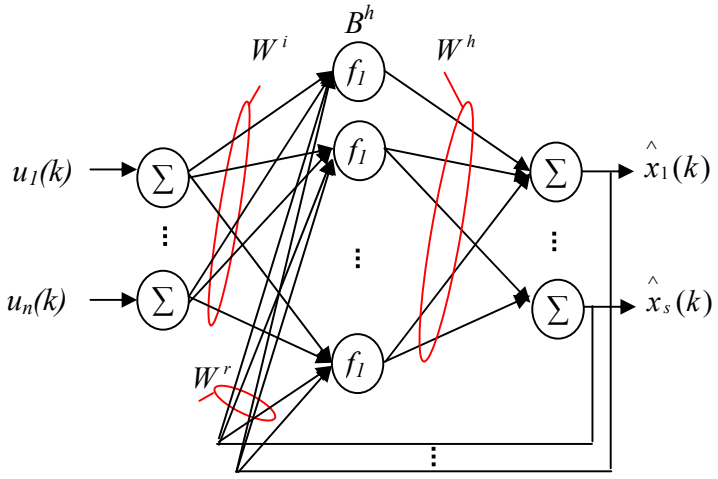

Figure. II.6 : Réseau de neurones dynamique

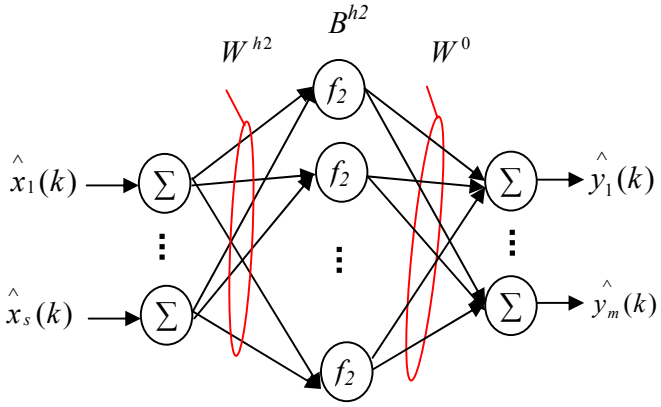

Figure. II.7 : Réseau de neurones statique

On peut envisager deux cas pour la modélisation d'un système en utilisant un réseau de neurones à espace d'état :

Cas ou les variables d'état ne sont pas mesurées, dans ce cas les données utilisées pour l'apprentissage (séquences d'apprentissage) sont la séquence des entrées de commande *u(k)* et la séquence des sorties mesurées $y_p(k)$, (Deng et al. 2009). Dans ce cas l'apprentissage est dit semi dirigé, il peut être réalisé comme pour un réseau non bouclé à travers des différentes copies en utilisant par exemple la rétropropagation à travers le temps (Werbos 1990). La figure suivante montre l'apprentissage semi dirigé dans le cas ou la sortie égale à l'un des états (Rivals 1995), (Lucea 2006).

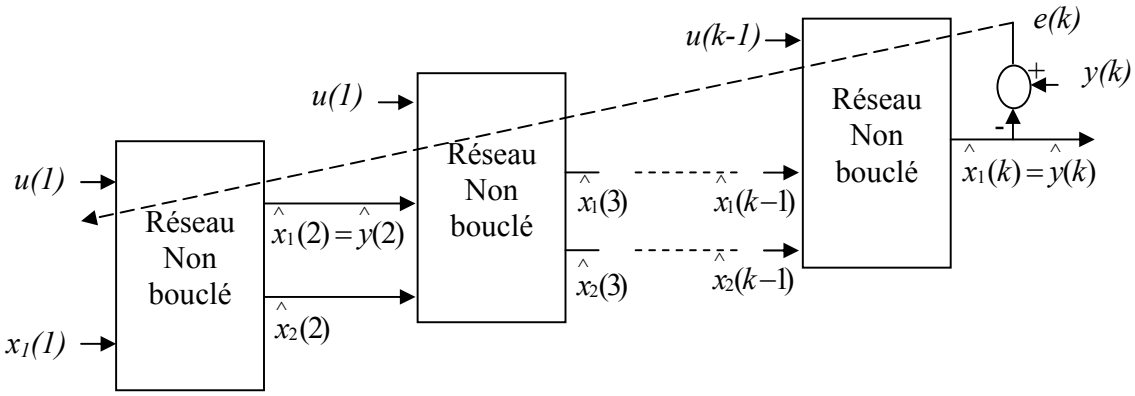

Figure. II.8 : Apprentissage semi dirigé (rétropropagation à travers le temps)

Cas où les variables d'état sont mesurées, c'est le cas que nous allons traiter dans ce travail, les données utilisées pour l'apprentissage (séquences d'apprentissage) sont la séquence des entrées de commande *u(k),* la séquence des variables d'état $x_p(k)$ et la séquence des sorties mesurées $y_p(k)$.

Dans ce cas l'apprentissage est dit dirigé et le système d'apprentissage est montré sur la figure suivante :

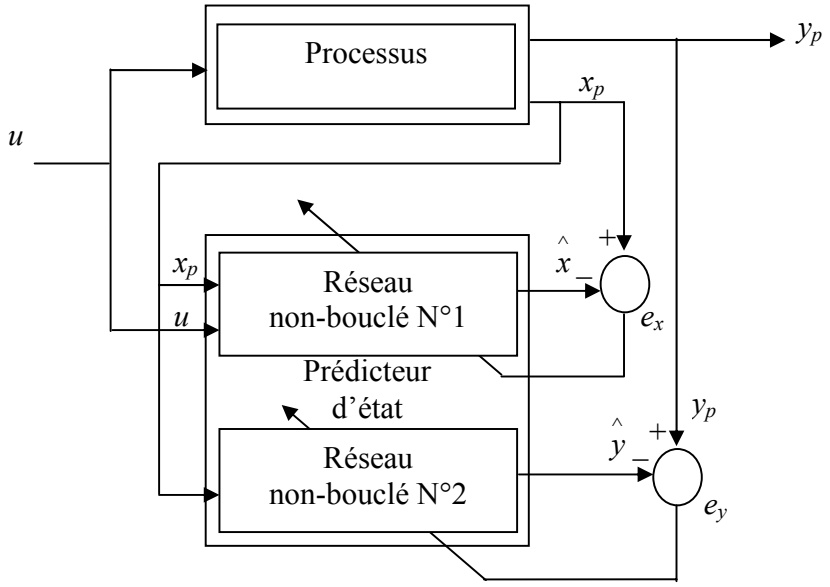

Figure. II.9 : Principe d'apprentissage d'un SSNN

Nous considérons deux sous réseaux distincts le premier correspondant à l'équation d'état pour approcher l'état, le deuxième correspondant à l'équation de sortie pour approcher la sortie, l'apprentissage des deux réseaux est dirigé par l'état *xp* du processus, les erreurs e_x et e_y interviennent dans les deux fonctions de coût à minimiser et chacune de ces erreurs est l'entrée d'un réseau utilisant la rétropropagation pour calculer ses poids.

Pour l'apprentissage de ces deux sous réseaux on peut appliquer les méthodes d'apprentissage de premier ou de deuxième ordre basées sur le calcul du gradient:

Méthodes du premier ordre : les méthodes dites du premier ordre sont basées sur le calcul du gradient de la fonction du coût et le minimum de cette fonction est atteint si son gradient

(autrement dit sa dérivée) est nul. On compte parmi ces méthodes, la méthode du gradient simple, la méthode du gradient stochastique…

Méthodes du deuxième ordre : les méthodes du deuxième ordre sont ainsi appelées parce qu'elles prennent en considération la dérivé seconde de la fonction de coût. Parmi ces méthodes, on compte la méthode de newton, la méthode de Levenberg Marquardt (Lucea 2006), (Hagan et al.1995), (Lennox et al. 2001).

Nous présentons ci-dessous celles que nous avons mises en œuvre dans ce travail.

a- Méthode du gradient récursif

L'algorithme récursif du gradient, porte aussi le nom d'algorithme du gradient stochastique est la méthode la plus utilisée dans la communauté neuronale. Il consiste à modifier les paramètres proportionnellement au gradient de la fonction du coût partiel.

Afin d'estimer les paramètres d'un réseau de neurones par l'algorithme du gradient, il faut suivre les étapes suivantes :

1. choisir la taille du réseau et initialiser les poids et les seuils du réseau (selon le système à modéliser).
2. calculer des sorties des différentes couches

   la sortie d'un neurones i est : $$y_i = g(v_i) = g\left(\sum_{j=1}^{n1} W_{ij}.y_j\right) \qquad \text{……..…} \quad \text{(II-8)}$$

   Où g est la fonction d'activation du neurone, *W* représente les poids des liens qui relient ces neurones aux neurones de la couche précédente.
3. calculer la fonction de coût partiel relatif à l'exemple *k*, généralement on choisit une fonction de coût quadratique.

   Pour un seul exemple, l'erreur quadratique sur la sortie est :

   $$J = \frac{1}{2}\sum_{i=1}^{n2}(y_i - y_{di})^2 \qquad \text{…….} \quad \text{(II-9)}$$

   Où y_i est la sortie du réseau de neurones, y_{di} est la sortie désirée (mesurée).
4. calculer le gradient de la fonction de coût par la rétropropagation des erreurs depuis les sorties vers les entrées en utilisant des grandeurs intermédiaires relatives à des neurones qui se trouvent entre ce paramètre et les sorties du réseau (Bose et al. 2007).

5. modifier les paramètres selon la relation suivante :

$$W(k) = W(k-1) - \mu(k) \frac{\partial J}{\partial W} /_{W(k-1)} \quad \text{..... (II-10)}$$

Avec $\mu(k) > 0$ est le pas de progression dans la direction de descente, La direction de descente est l'opposée de celle du gradient

Le choix de la valeur de $\mu(k)$ est très important, on le choisit généralement entre 0 et 1. Il permet d'accélérer la convergence lorsqu'il est proche de 1 où de la freiner lorsqu'il est proche de 0 (Personnaz et al. 2003).

b- Méthode du gradient à pas asservi

La méthode du gradient est efficace loin du minimum de la fonction du coût, mais quand on s'approche du minimum la norme du gradient diminue et donc l'algorithme progresse plus lentement. En effet si le pas est petit l'algorithme ralenti et si le pas est grand, on aboutit à un phénomène d'oscillation autour de la solution (Personnaz 2003). Pour éviter ce problème, on peut régler le pas à l'aide de la norme du gradient de sorte qu'il évolue dans le sens inverse de la norme comme suit (Oussar 1998) :

$$\mu(k) = \frac{\mu}{1 + \|\nabla J\|}$$

Où μ est un paramètre constant.

c- Méthode de Levenberg-Marquardt (LM)

L'algorithme de Levenberg-Marquardt est un algorithme itératif, consiste à modifier les paramètres proportionnellement au gradient de la fonction du coût total. Elle a été appliquée pour l'apprentissage d'un SSNN dans de plusieurs travaux de recherche (Van Lint et al. 2002), (Gil et al. 2005), (Mirikitani et al.2010), cet algorithme adopte une direction de déplacement qui n'est plus la direction du gradient, mais une transformation linéaire appropriée du gradient du coût total, il consiste à modifier les paramètres du réseau selon la formule suivante (Dreyfus et al. 2002) :

$$W(i) = W(i-1) - \left[H\left(W(i-1)\right) + \lambda_i I\right]^{-1} \nabla J\left(W(i-1)\right) \quad \cdots\cdots\cdots\cdots\cdots\cdots \text{(II-11)}$$

où μ_i est un paramètre constant strictement supérieur à zéro

H est la matrice hessienne où la dérivé seconde par rapport aux paramètres.

Cet algorithme s'applique à la fonction de coût quadratique parce que son Hessien peut être approché par : $H = Ja^T Ja$

Ja est la matrice jacobienne ou le jacobien.

Un calcul détaillé de la matrice jacobienne peut être trouvé dans la thèse de doctorat de Mahul (Mahul 2005).

L'algorithme de Levenberg-Marquardt est détaillé ci-dessous.

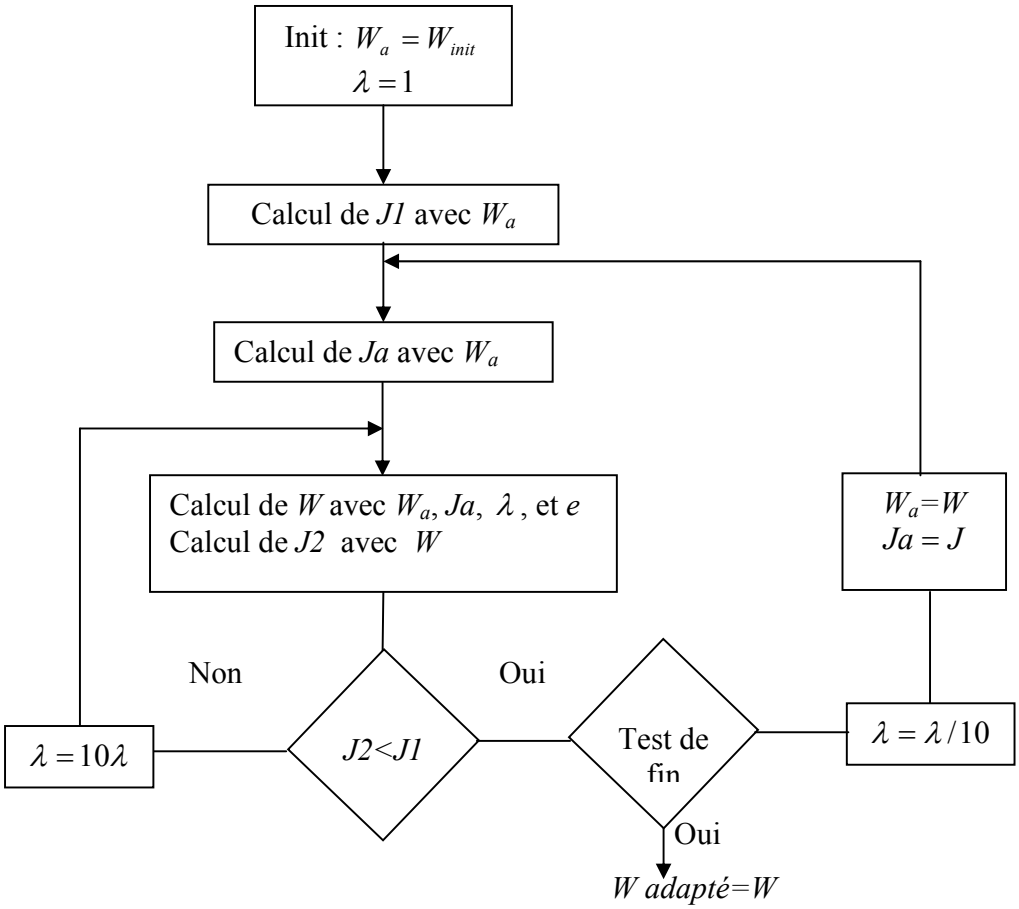

Figure. II.10 : Algorithme de Levenberg-Marquardt

II-6-Etude des cas particuliers

II-6-1- Système linéaire

Le modèle (II-7) présenté précédemment est un modèle général qui inclut les cas linéaire et non linéaire (Pan et al. 2004), (Zamarreno et al. 1998). Lorsque les fonctions de transfert du SSNN sont linéaires et lorsque le biais est égal à zéro, alors l'équation (II-7) devient :

$$\begin{cases} \hat{x}(k+1) = W^h.(W^r.\hat{x}(k) + W^i.\vec{u}(k)) \\ \hat{y}(k) = W^0.(W^{h2}.\hat{x}(k)) \end{cases} \quad \text{..... (II-12)}$$

En posant $W^h.W^r = \Phi$, $W^h.W^i = \Gamma$ et $W^0.W^{h2} = \Psi$ et en effectuant les remplacements nécessaires dans l'équation (II-12), on obtient :

$$\begin{cases} \hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) \\ \hat{y}(k) = \Psi\hat{x}(k) \end{cases} \quad \text{..... (II-13)}$$

Le modèle de (II-13) est de la forme d'un modèle linéaire déterministe. Φ, Γ et Ψ sont des matrices dont les éléments sont les poids des connections du réseau SSNN.

II-6-2-Systèmes non linéaires avec perturbations

Soit le système non linéaire présentant des perturbations mesurées et non mesurées, dont la représentation d'état discrète est la suivante :

$$\begin{cases} x(k+1) = f_1(x(k), u(k), d(k), \xi(k)) \\ y(k) = f_2(x(k)) \end{cases} \quad \text{..... (II-14)}$$

Dans ce système, x est le vecteur d'états, u le vecteur des entrées de commande, d est le vecteur de perturbations, ζ représente toutes les perturbations non mesurées, et y est la sortie du processus. f_1 et f_2 sont deux fonctions non linéaires qui représentent le comportement des états et le comportement de la sortie.

Le réseau de neurones à espace d'état qui peut représenter le système donné par (II-14) est le suivant (Zamarreno et al. 1997), (Zamarreno et al. 1998) :

$$\begin{cases} \hat{x}(k+1) = W^h.f_1(W^r.\hat{X}(k) + W^i\begin{pmatrix} u(k) \\ d(k) \end{pmatrix} + W^e.w(k) + B^h) \\ \hat{y} = W^0.f_2(W^{h2}.\hat{x}(k) + B^{h2}) \end{cases} \quad \text{..... (II-15)}$$

avec

$w(k) = y(k) - \hat{y}(k)$

- W et B : respectivement des matrices et des vecteurs à estimer,
- f_1 et f_2 : deux fonctions (non-linéaires en général, et très souvent sigmoïdales).

De cette manière, le modèle stochastique est parfaitement décrit.

II-7-Réseau de neurones à espace d'état (SSNN) à structure affine.

La structure affine proposée par (Henriques et al. 2001) est un peu particulière. En effet, elle peut être vue comme une modification du réseau de neurones récurrent à temps discret proposée par Hopfield avec une entrée exogène. Cette architecture, du type récurrent, se compose d'un corps principal appelé feedforward où les signaux se propagent en avant des neurones d'entrée aux neurones cachés puis des neurones de la couche cachée à ceux de la couche de sortie pour produire la sortie de réseau. De plus, la couche cachée contient deux types de neurones, des neurones avec une fonction d'activation linéaire et des neurones à fonction d'activation sigmoïde. La propriété de récurrence du SSNN est réalisée à l'aide de boucles de retour avec un retard d'une période d'échantillonnage. Chaque boucle de retour est injectée dans tous les neurones ayant une fonction d'activation non linéaire et dans un seul neurone linéaire de la couche cachée. Par exemple, le premier retour est injecté dans tous les neurones non linéaires et dans le premier neurone linéaire, le deuxième retour est injecté dans tous les neurones non linéaires et dans le deuxième neurone linéaire, etc (Gil et al. 2001), (Gil et al. 2004), (Gil et al. 2005). Ce principe est illustré par la figure. II.11 avec un réseau ayant deux sorties.

L'utilisation de ces deux types de neurones dans la couche cachée a pour but d'améliorer les performances de modélisation du réseau de neurones. Ceci est particulièrement vrai dans le cas d'un système à identifier ayant une dynamique non-linéaire douce.

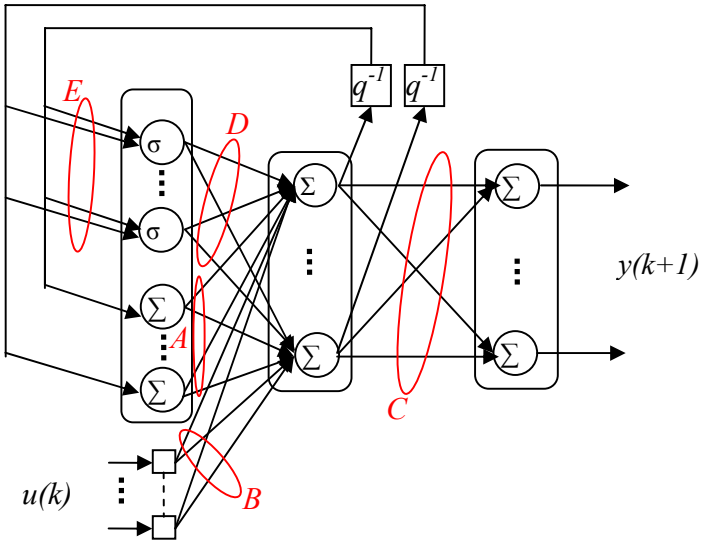

Figure. II.11 : Réseau de neurones à espace d'état à structure affine

Mathématiquement, le modèle peut être décrit dans un espace d'état comme suit :

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) + D\sigma.(Ex(k)) \\ y(k) = Cx(k) \end{cases} \quad \text{..... (II-16)}$$

$\vec{x} \in \Re^s$ est le vecteur d'états, $\vec{u} \in \Re^n$ est le vecteur d'entrées (commandes), $\vec{y} \in \Re^m$ est le vecteur de sorties.

Les grandeurs A, B, C, D et E sont des matrices à valeurs réelles. $\sigma(\bullet)$ est une fonction non linéaire, elle peut être sigmoïde, hyperbolique ou éventuellement d'une autre forme encore.

II-8- Utilisations et applications des SSNN

Le SSNN peut être utilisé comme un modèle, autrement dit un simulateur, ou comme un prédicteur à un pas.

- *Les modèles de simulation* : un modèle de simulation peut prédire la sortie du processus qu'il représente à long terme, il est utilisé de manière indépendante du processus, il doit

donc posséder un comportement aussi proche que possible à celui du processus. De tels modèles sont utilisés pour valider la conception d'un système avant sa fabrication, pour la prévision à long terme…etc.

Dans ce cas, le modèle neuronal ne peut pas utiliser les sorties mesurées sur le processus comme des entrées, ses entrées seront les sorties prédites par lui-même aux instants précédents. L'estimation de paramètres de tels modèles est non adaptatif (Oussar 1998).

- *Les modèles prédicteur à un pas* : un prédicteur à un pas est utilisé en parallèle avec le processus dont il est le modèle, en effet une partie des entrées du modèle sont les sorties mesurées du processus. Il prédit la sortie du processus qui vient juste après un pas d'échantillonnage,; il est ainsi par exemple possible de prévoir la sortie du processus à l'instant k+1 sachant les sorties de ce dernier à l'instant k (Dreyfus et al. 2002).

Les figures. II.12 et II.13 montre respectivement un prédicteur neuronal et un simulateur neuronal.

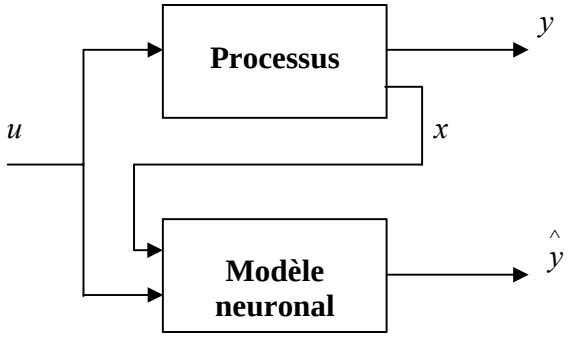

Figure. II.12 : modèle prédicteur du SSNN

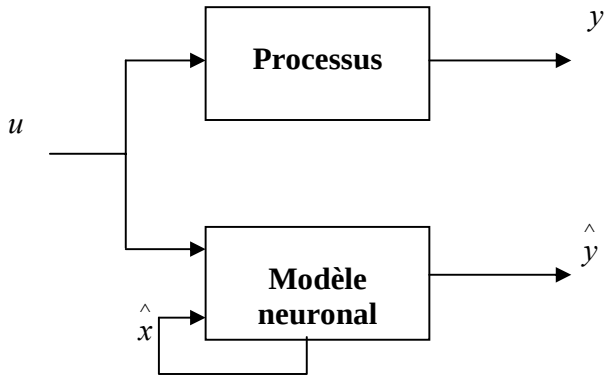

Figure. II.13 : modèle simulateur du SSNN

II-8-1-choix du nombre de neurones

L'élaboration d'un bon modèle revient au choix de la structure et aussi au nombre de neurones dans la couche cachée. Concernant les couches d'entrée et de sortie d'un SSNN, elles doivent incorporer autant de neurones que le nombre d'entrées et de sorties du système à modéliser. A l'inverse, le choix du nombre de neurones dans la couche cachée n'est pas imposé par le système à modéliser. Le nombre de neurones à incorporer dans la couche cachée devrait être choisi afin de fournir de bonnes possibilités de généralisation pour l'estimateur neuronal. En effet, un réseau de neurones composant d'un nombre insuffisant de neurones dans la couche cachée ne peut pas reproduire au mieux le comportement dynamique du système, alors qu'un réseau de neurones où le nombre de neurones cachés est trop important pourrait souffrir d'une faible généralisation ou sur-ajustement (Gil et al. 2005).

II-8-2-Les applications du SSNN

Les réseaux de neurones à espace d'état peuvent être appliqués à tout type de système dynamique linéaire et non linéaire (avec une linéarité douce ou forte). Le SSNN convient également à la commande prédictive des processus non linéaires qui fonctionnent sous différents modes. Dans le cas de la commande prédictive, le SSNN joue le rôle d'un prédicteur, on l'insère dans des lois de commande dans le but de prévoir la sortie du système, comme avec toutes autres méthodes prédictives basées sur un modèle.

Plusieurs règles d'apprentissage peuvent être appliquées pour ajuster les poids du SSNN. La majorité des algorithmes utilisés généralement pour l'apprentissage des réseaux de neurones à

couches peuvent parfaitement convenir. On peut citer les algorithmes basés sur le calcul du gradient et d'autres algorithmes comme ceux évoqués dans la section 1. Les algorithmes basés sur le calcul du gradient sont plus faciles à mettre en œuvre. Cependant, de meilleurs résultats ont été obtenus dans des expériences d'apprentissage avec les SSNN en utilisant les techniques de recherche aléatoire, telle que la méthode d'optimisation de recherche aléatoire modifiée (Zamarreno et al 1999).

Plusieurs applications on été faites en utilisant ce type de réseaux de neurones pour montrer ses possibilités dans l'identification et la prévision non linéaires ainsi que ses avantages, ses inconvénients et ses limites. Parmi ces applications, on peut détailler les cas suivants, celui d'un système assez complexe, l'autre d'un système stochastique et le cas d'un système qui présente de fortes linéarités.

Ces exemples sont les suivants :

1) l'identification d'un champ de capteur solaire distribué. Le capteur solaire distribué est une centrale solaire située à Plataforma Solar de Almería dans le désert de Tabernas, dans le sud de l'Espagne. Un SSNN a été utilisé avec une structure affine et avec l'algorithme d'apprentissage de Levenberg-Marquardt. Cet algorithme fournit un compromis très intéressant entre la vitesse de la méthode de Newton et la garantie de convergence de l'algorithme de descente rapide. L'apprentissage de Levenberg-Marquardt est le mieux adapté si le réseau contient moins de quelques centaines de paramètres. L'apprentissage de Levenberg-Marquardt a donné de bons résultats et cette approche a pu capturer la dynamique du système en boucle fermée (Gil et al 2004).

2) l'identification et la commande prédictive de l'unité de dissolution dans une usine de sucre située à Binavente (Espagne). Le processus est un réservoir de mélange qui est conçu pour dissoudre les cristaux de sucre de mauvaise qualité dans un jus qui a été précédemment obtenu au moyen d'un procédé de diffusion à partir de betteraves et de l'eau. Afin d'améliorer le processus de fonte, un écoulement de vapeur est employé comme un élément de chauffe. La variable employée pour commander le processus est le débit d'entrée du jus, le processus est stochastique, il présente des perturbations non mesurées (le jus est mélanger aux cristaux de sucre). Le but est de maintenir la concentration de sucre à la sortie aux valeurs fixées auparavant, afin de permettre à la prochaine étape du processus de fabrication de sucre de fonctionner d'une façon optimale. Malgré le comportement stochastique du système, l'utilisation d'un modèle SSNN déterministe

suffit pour modéliser fidèlement le système. Pour ajuster les poids et les biais dans le SSNN, une technique basée sur le gradient peut être utilisée, cependant de meilleurs résultats ont été obtenus dans des expériences d'apprentissage en utilisant la méthode d'optimisation de recherche aléatoire modifiée (Zamarreno et al 1997).

3) la modélisation et la commande d'un processus (chimique) de sulfitage. Cet exemple est, comme le précédent, appliqué dans l'usine du sucre. Ce processus chimique est complexe, sa dynamique présente de fortes non linéarités et le SSNN est utilisé pour prédire la sortie du système. Le SSNN est inséré en tant que prédicteur dans le modèle de commande prédictive qui a pour but de maintenir le PH de la sortie à une valeur indiquée, en dépit des perturbations agissant sur le système. La méthode d'apprentissage qui a donné de bons résultats est la méthode d'optimisation de recherche aléatoire modifiée (Zamarreno et al 1999).

II-9- Conclusion

Les réseaux récurrents à espace d'état, autrement dit les SSNN, sont différents des autres réseaux de neurones. Ils possèdent des particularités qui leur donnent la possibilité de mieux représenter les systèmes dynamiques. En effet, leurs structures correspondent, dans la forme de ses équations, à une description d'un système classique sous la forme d'une représentation d'une équation d'état non-linéaire. Ils peuvent être facilement employés pour établir une représentation réelle d'un processus tout en permettant une analyse facile. En utilisant des paramètres du réseau permet une étude directe des propriétés, telles que la stabilité, le temps de réponse… Ils peuvent être utilisés pour l'identification d'une large gamme des systèmes dynamiques linéaires et non-linéaires. L'utilisation de bases de données réelles (à partir de mesures prises sur le système réel) dans l'étape de l'apprentissage, permet de modéliser un système réel et d'avoir accès à certains de ses paramètres. Comme conclusion, on peut dire que les SSNN permettent de s'approcher de l'automatique classique, puisque les SSNN et l'automatique utilisent le même espace (espace d'état) et les mêmes grandeurs. Le SSNN peut lui-même de ce fait faire l'objet d'une étude basée sur le point de vue de l'automatique classique.

Chapitre III

III-1- Introduction

Le deuxième chapitre de ce mémoire nous à permis de présenter les réseaux de neurones à espace d'état et nous avons vu comment les mettre en œuvre pour la modélisation et l'identification des systèmes dynamiques linéaires et non linéaires.

Dans le présent chapitre, nous allons modéliser différents processus dynamiques simulés à partir des équations d'état en utilisant ce type de réseaux (SSNN). Pour cela deux méthodes d'apprentissage ont été programmées, une méthode du deuxième ordre dite "méthode de Levenberg Marquardt" qui est largement utilisée pour l'apprentissage des réseaux bouclés (dynamiques) et non bouclés (feedforward) (Mirikitani et al.2010), et une méthode du premier ordre dite «méthode du gradient stochastique" qui est largement utilisée pour l'apprentissage des réseaux non bouclés. Les séquences de mesures utilisées pour l'apprentissage du SSNN sont générées à partir du modèle mathématique du processus.

Nous allons commencer par la modélisation et l'identification des différents systèmes linéaires du premier et quatrième ordre à une entrée de commande et un système du troisième ordre à deux entrées de commande, en utilisant les deux méthodes d'apprentissage mises en œuvre. Ces modèles neuronaux sont des modèles de simulation (simulateurs) qui reproduisent le comportement du système sur un horizon infini et qui peuvent être utilisés d'une manière indépendante du processus. Nous étudions l'influence du bruit sur le SSNN en prenant comme exemple un système linéaire du deuxième ordre en ajoutant sur la sortie de ce système un bruit gaussien de différentes variances. Nous abordons par la suite les systèmes non linéaires, en commençant par le SSNN prédicteur à un pas. Nous prenons comme exemple un système d'ordre quatre, ensuite nous essayons de trouver le modèle simulateur correspondant à ce système en utilisant les deux méthodes d'apprentissage mises en œuvre et en faisant varier la non linéarité du système. Nous terminons par une comparaison du SSNN avec un autre type de réseaux de neurones, le perceptron multicouche, qui est souvent utilisé pour sa capacité d'identification des processus dynamiques et de modélisation entrée/sortie. Deux exemples de systèmes sont utilisés pour évaluer les performances du SSNN, un système linéaire d'ordre trois et un système non linéaire d'ordre deux.

III-2- Identification de systèmes linéaires

Les objectifs de cette identification sont :

1. de trouver des simulateurs neuronaux à des systèmes linéaires dynamiques d'ordre différents, c'est-à-dire de trouver de modèles neuronaux qui reproduisent au mieux le comportement du système sur un horizon infini.
2. d'évaluer les performances d'apprentissage du SSNN,
3. d'évaluer la qualité de l'estimation déduite du SSNN.

a- Identification d'un système linéaire d'ordre 1

Prenons l'exemple d'un système linéaire du premier ordre dont la représentation d'état est la suivante :

$$\begin{cases} x(k+1) = 0.9429\,x(k) + 0.01942\,u(k) \\ y(k) = 1.17\,x(k) \end{cases} \quad \text{..... (III-1)}$$

L'architecture du modèle neuronal (SSNN) est : 1 entrée de commande, 2 neurones dans la première couche cachée, 1 seul neurone d'état, 2 neurones dans la deuxième couche cachée et un seul neurone de sortie, ce que l'on peut noter (1,2,1,2,1). Cette architecture est montrée sur la figure. III.1 :

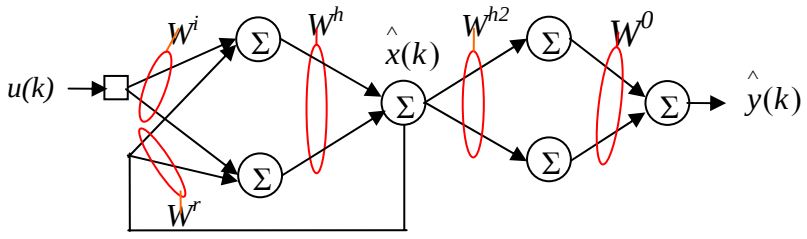

Figure III.1 : Architecture du SSNN (1,2,1,2,1)

La représentation d'état qu'il est possible d'écrire à partir de cette architecture neuronale est la suivante :

$$\begin{cases} \hat{x}(k+1) = \Phi\,\hat{x}(k) + \Gamma u(k) \\ \hat{y}(k) = \Psi\,\hat{x}(k) \end{cases} \quad \text{..... (III-2)}$$

Tel que :

$$\Phi = W^h W^r \,,\ \Gamma = W^h W^i \,,\ \Psi = W^0 W^{h2}$$

Les détails concernant l'apprentissage sont les suivants :

- La séquence de commande utilisée pour l'apprentissage est une suite de créneaux d'amplitude aléatoires et de durées aléatoires, la séquence comporte 400 pas d'échantillonnage et est représentée sur la figure III.2 –a-.
- Les séquences de variables d'état et de sorties utilisées pour l'apprentissage sont générées à partir du modèle mathématique de systèmes.
- La séquence de commande utilisée pour l'estimation de la performance du SSNN (séquence de test) est une suite de créneaux d'amplitudes aléatoires et de durées aléatoires. La séquence comporte 300 pas d'échantillonnage et est représentée sur la figure III.2 –b-.

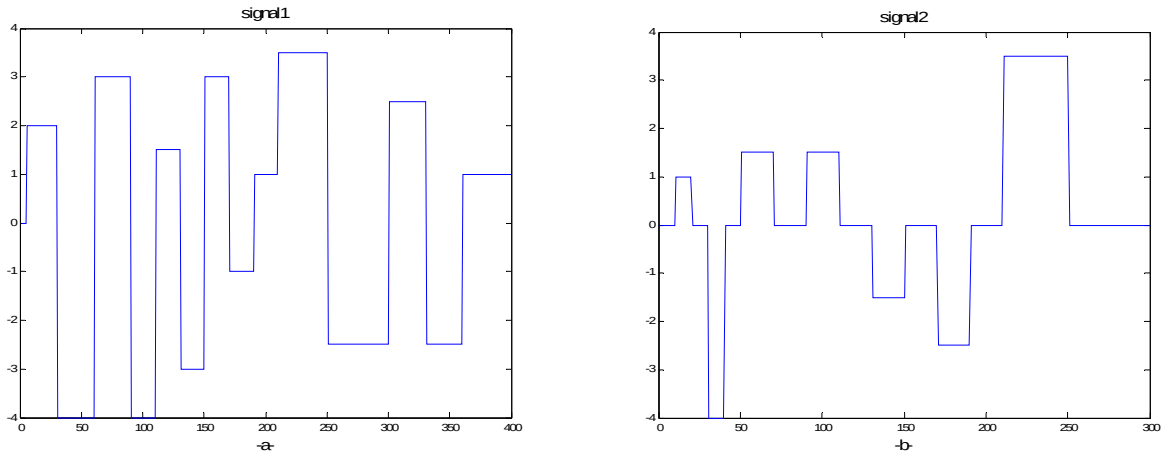

Figure.III.2 : -a- Séquence d'apprentissage (signal1)
-b- Séquence de test (signal2)

La figure III.3 montre les résultats d'apprentissage et de test en utilisant un apprentissage hors ligne avec l'algorithme de Levenberg Marquardt. Le pas d'apprentissage est fixé au début à 0.001.

La figure III.4 montre les résultats d'apprentissage et de test en utilisant un apprentissage hors ligne avec l'algorithme du gradient. Le pas d'apprentissage est fixé à 0.65.

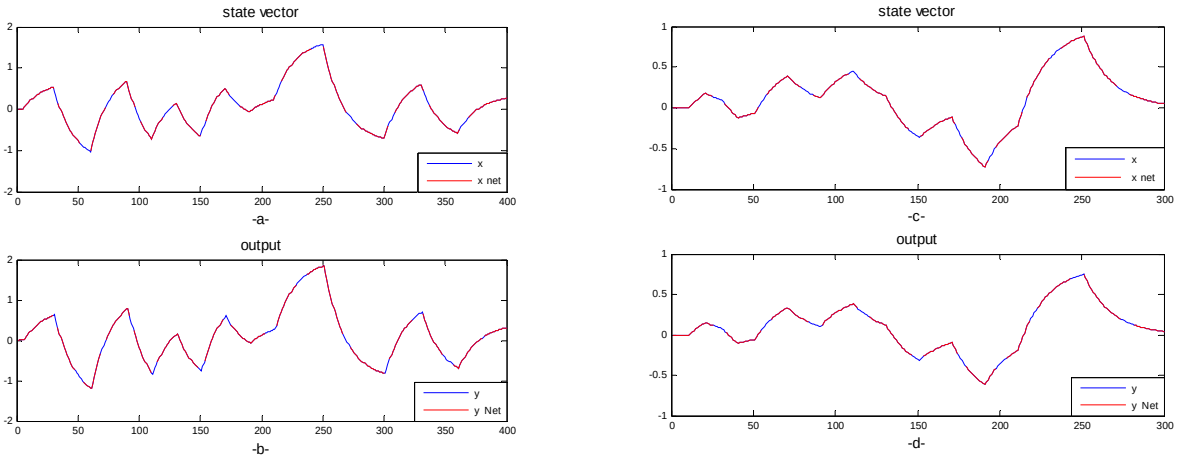

Figure. III.3 : Résultats obtenus avec la méthode de L.M

-a- et -b- séquence d'apprentissage

-c- et -d- séquence de test

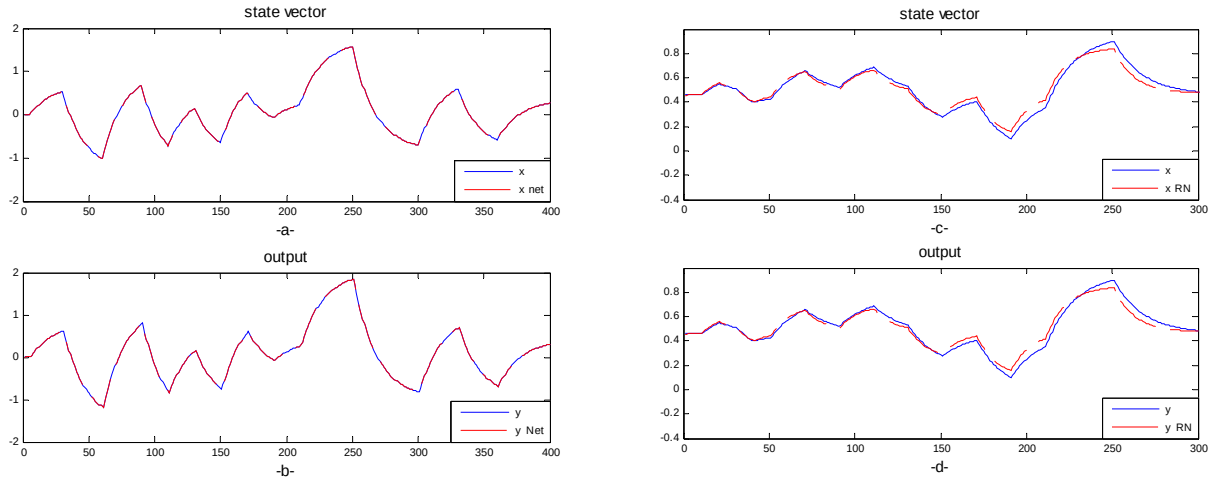

Figure. III.4: Résultats obtenus avec la méthode du gradient

-a- et -b- séquence d'apprentissage

-c- et -d- séquence de test

	EQMAx	**EQMAy**	**EQMTx**	**EQMTy**
Levenberg-Marquardt	2.28×10^{-32}	1.38×10^{-32}	0.19×10^{-32}	1.09×10^{-32}
Gradient	2.29×10^{-6}	4.35×10^{-32}	1.3×10^{-3}	1.3×10^{-3}

Tableau. III.1 : Erreurs quadratiques moyennes entre les états réels et les états estimés par le SSNN

EQMAx : erreur quadratique moyenne de l'état x sur l'ensemble d'apprentissage.

EQMAy : erreur quadratique moyenne de la sortie y sur l'ensemble d'apprentissage.

EQMTx : erreur quadratique moyenne de l'état x sur l'ensemble de test.

EQMTy : erreur quadratique moyenne de la sortie y sur l'ensemble de test.

Le tableau ci-dessus rassemble les erreurs quadratiques moyennes, d'apprentissages notés (EQMA) et de tests notés (EQMT).

D'après les figures.III.3, III.4 et le tableau.III.1, nous constatons que des performances excellentes sont obtenues en utilisant l'algorithme de Levenberg Marquardt avec seulement deux neurones dans les couches cachées et seulement après 5 itérations ; l'EQMTx et l'EQMTy sont de l'ordre de 10^{-32}, tandis que l'erreur quadratique moyenne obtenue en utilisant la méthode du gradient stochastique est de l'ordre de 10^{-3}.

Les matrices des poids obtenues après l'apprentissage sont les suivantes :

$$W^i = \begin{bmatrix} -0.9751 \\ 0.9249 \end{bmatrix} \qquad W^r = \begin{bmatrix} 0.8980 \\ 0.7991 \end{bmatrix}$$

$$W^h = \begin{bmatrix} 0.5321 & 0.5820 \end{bmatrix} \qquad W^{h2} = \begin{bmatrix} 0.8373 \\ -0.6796 \end{bmatrix}$$

$$W^0 = \begin{bmatrix} 0.7189 & -0.8359 \end{bmatrix}$$

Le modèle neuronal est peut être mis sous la représentation d'état suivante avec les valeurs numériques obtenues après apprentissage :

$\Phi = W^h W^r = 0.9429$, $\Gamma = W^h W^i = 0.0194$, $\Psi = W^0 W^{h2} = 1.1700$

$$\begin{cases} \hat{x}(k+1) = 0.9429\,\hat{x}(k) + 0.0194u(k) \\ \hat{y}(k) = 1.1700\,\hat{x}(k) \end{cases} \quad \text{.....} \quad \text{(III-3)}$$

Les matrices poids obtenues (Φ, Γ, Ψ) sont identiques aux matrices d'état du système à modéliser. L'un des avantages des SSNN est la possibilité d'analyser le système à partir des

poids obtenus, on sait par exemple que le système est stable si les valeurs propres de $\Phi = W^h W^r$ sont dans le disque unité ouvert.

b- Identification d'un système linéaire d'ordre 4

Prenons l'exemple d'un système linéaire du quatrième ordre dont la représentation d'état est la suivante :

$$\begin{cases} x(k+1) = \begin{bmatrix} 0.9515 & 0.1004 & -0.0129 & 0.0002 \\ -0.1004 & 0.7033 & 0.1346 & -0.0013 \\ -0.0129 & -0.1346 & 0.2500 & -0.1538 \\ -0.0002 & -0.0013 & 0.1538 & 0.7153 \end{bmatrix} x(k) + \begin{bmatrix} 0.0122 \\ 0.0109 \\ 0.0019 \\ 0 \end{bmatrix} u(k) \\ y(k) = \begin{bmatrix} 0.1181 & -0.1307 & 0.0563 & 0.0069 \end{bmatrix} x(k) \end{cases} \quad \text{..... (III-4)}$$

L'architecture du modèle neuronal (SSNN) est (1,4,4,2,1) en utilisant la méthode de Levenberg-Marquardt, comme elle est représentée sur la figure III.5.

La figure III.6 montre l'architecture neuronale (1,4,4,4,1) en utilisant la méthode du gradient

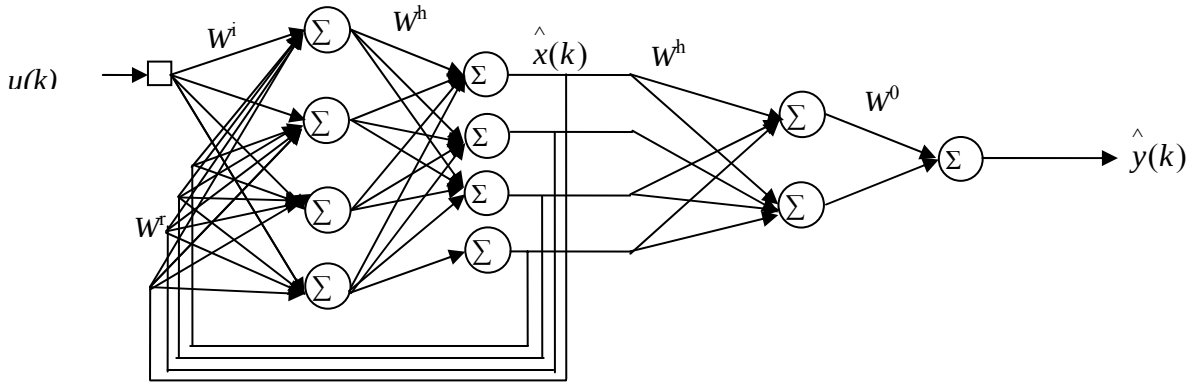

Figure. III.5: Architecture du SSNN (1,4,4,2,1)

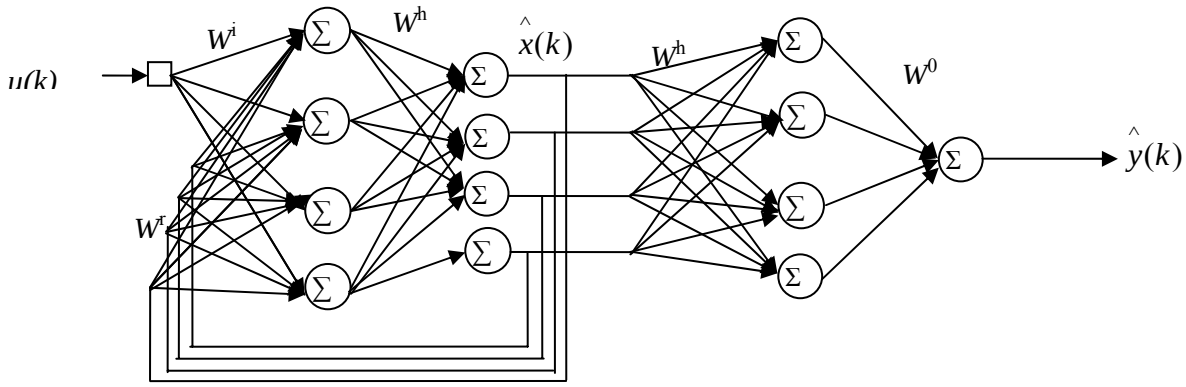

Figure.III.6 : Architecture du SSNN (1,4,4,4,1)

La figure III.7 montre les résultats du test du SSNN (1,4,4,4,1) avec le signal (1), après 2200 itérations d'un apprentissage hors ligne avec la méthode du gradient stochastique à pas variable. Le pas d'apprentissage est fixé au début à 0.0025.

La figure III.8 montre les résultats du test du SSNN (1,4,4,2,1) avec le signal(1), après un apprentissage hors ligne avec la méthode de Levenberg-Marquardt, après 90 itérations seulement.

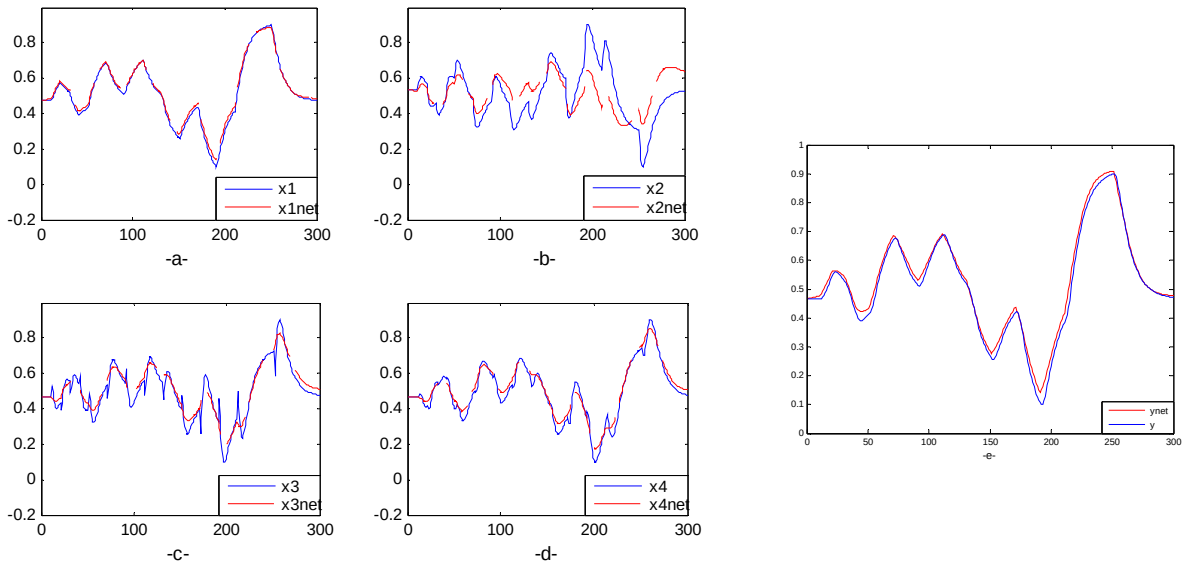

Figure.III.7 : Résultats du test obtenus avec la méthode du gradient

a, b, c et d : Les états x_1, x_2, x_3 et x_4

e : La sortie y

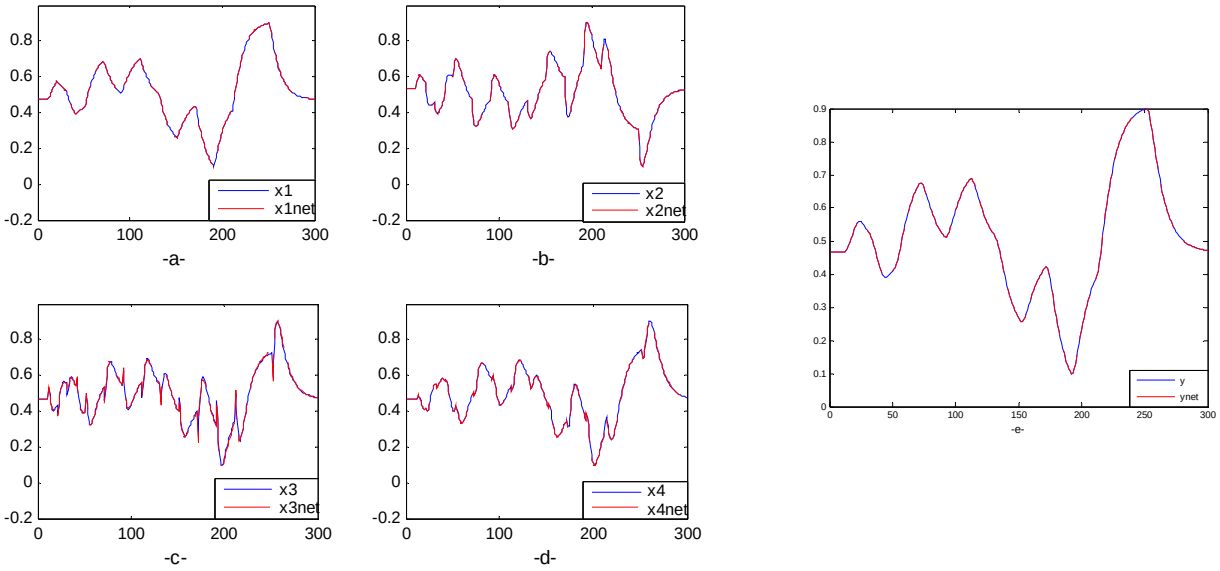

Figure.III.8 : Résultats du test obtenus avec la méthode de LM

a, b, c et d : Les états x_1, x_2, x_3 et x_4

e : La sortie y

	EQMTx_1	EQMTx_2	EQMTx_3	EQMTx_4	EQMTy
Levenberg-Marquardt	0.56×10^{-31}	0.06×10^{-31}	0.01×10^{-31}	0.02×10^{-31}	33.6×10^{-31}
Gradient	0.3×10^{-3}	15×10^{-3}	2.2×10^{-3}	1.1×10^{-3}	0.5×10^{-3}

Tab. III.2 : Erreurs quadratiques moyennes entre les états réels et les états estimés, sortie réel et sortie estimée par SSNN (1,4,4,2,1)

D'après les figures III.7, III.8 et le tableau.III.2, les performances sont meilleures en utilisant l'algorithme de Levenberg Marquardt avec 4 neurones dans la 1ère couche cachée et seulement deux neurones dans la 2ème couche cachée. L'EQMTx et l'EQMTy sont de l'ordre de 10^{-32}, tandis que l'erreur quadratique moyenne obtenue en utilisant la méthode du gradient est de l'ordre de 10^{-3}, et ce avec plus de paramètres à ajuster, 4 neurones dans la 1ère couche cachée et 4 neurones dans la 2ème couche cachée.

Les matrices des poids obtenues après l'apprentissage avec la méthode de Levenberg-Marquardt sont les suivantes :

$$W^i = \begin{bmatrix} -0.0219 \\ -0.0097 \\ 0.0090 \\ -0.0073 \end{bmatrix}$$

$$W^r = \begin{bmatrix} -0.5733 & -1.1831 & -0.0467 & -0.8185 \\ -1.4989 & 0.2179 & 0.2320 & -0.3432 \\ -0.2813 & 0.5097 & 0.3199 & -0.0985 \\ -0.4069 & -0.3399 & 0.2348 & 0.3596 \end{bmatrix}$$

$$W^h = \begin{bmatrix} 0.1368 & -0.7568 & 0.6908 & -0.2209 \\ -0.2617 & 0.1586 & 0.4964 & -0.3121 \\ 0.3551 & -0.3707 & 0.9013 & 0.2736 \\ -0.3916 & -0.0682 & -0.2372 & 0.9677 \end{bmatrix}$$

$$W^{h2} = \begin{bmatrix} 0.4630 & -0.1275 & 0.8774 & -0.8272 \\ 0.2158 & -0.4814 & -0.3113 & 0.5514 \end{bmatrix}$$

$$W^0 = \begin{bmatrix} 0.1468 & 0.2326 \end{bmatrix}$$

Le modèle neuronal est peut être mis sous la représentation d'état suivante avec les valeurs numériques obtenues après apprentissage :

$\Phi = W^h W^r$, $\Gamma = W^h W^i$, $\Psi = W^0 W^{h2}$

$$\begin{cases} \hat{x}(k+1) = \begin{bmatrix} 0.9515 & 0.1004 & -0.0129 & 0.0002 \\ -0.1004 & 0.7033 & 0.1346 & -0.0013 \\ -0.0129 & -0.1346 & 0.2500 & -0.1538 \\ -0.0002 & -0.0013 & 0.1538 & 0.7153 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 0.0122 \\ 0.0109 \\ 0.0019 \\ 0.0000 \end{bmatrix} u(k) \\ \hat{y}(k) = \begin{bmatrix} 0.1181 & -0.1307 & 0.0563 & 0.0069 \end{bmatrix} \hat{x}(k) \end{cases} \quad \text{…….. (III-5)}$$

La représentation d'état obtenue avec les poids optimaux du SSNN (équation (III-5)) est exactement aux chiffres significatifs, la même que la représentation d'état du système

(équations (III-4)), donc le modèle neuronal peut reproduire parfaitement le comportement du système.

c- Identification d'un système linéaire d'ordre trois avec deux entrées de commande

Prenons l'exemple d'un système linéaire de troisième ordre avec deux entrées de commande dont la représentation d'état est la suivante :

$$\begin{cases} x(k+1) = \begin{bmatrix} 0.8348 & -0.0045 & -0.0004 \\ 0.4122 & 0.0029 & 0.0004 \\ -0.3832 & -0.0043 & -0.0006 \end{bmatrix} x(k) + \begin{bmatrix} 0.2785 & 0.0150 \\ -0.4835 & -0.8229 \\ 0.4511 & -1.1558 \end{bmatrix} u(k) \\ y(k) = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} x(k) \end{cases} \quad .. \text{(III-6)}$$

L'objectif est d'apprendre ce système avec le SSNN représenté sur la figure III.9.

Les détails concernant l'apprentissage sont les suivants :

- Dans cet exemple on a un système à deux entrées, donc la base d'apprentissage est constituée de deux signaux de commande $u_1(k)$ et $u_2(k)$: $u_1(k)$ est une suite de créneaux utilisée auparavant (signal 1) et $u_2(k)$ est un signal sinusoïdal. La séquence comporte 400 pas d'échantillonnage.
- Les séquences de variables d'état et de sorties utilisées pour l'apprentissage sont générées à partir du modèle mathématique du système.
- La séquence de commande utilisée pour l'estimation de la performance du SSNN (séquence de test) est constituée de deux signaux également : un échelon unitaire et un signal sinusoïdal. La séquence comporte 100 pas d'échantillonnage et est représentée sur la figure III.10.

L'architecture du modèle neuronal (SSNN) est (2,3,3,2,1) et est représentée par la figure suivante :

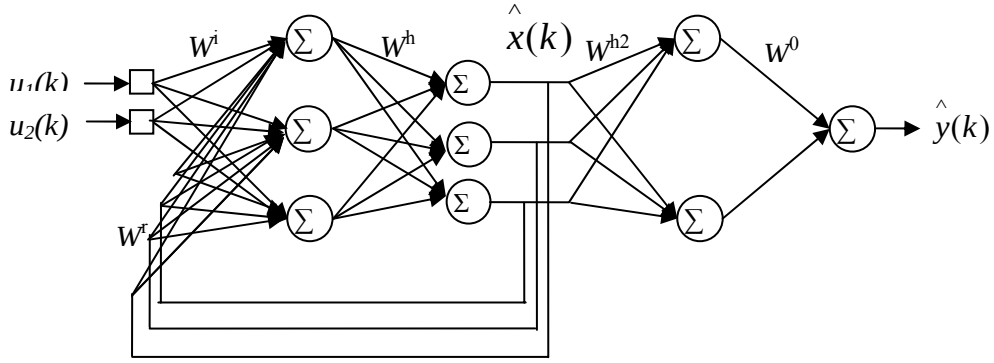

Figure III.9 : Architecture du SSNN (2,3,3,2,1)

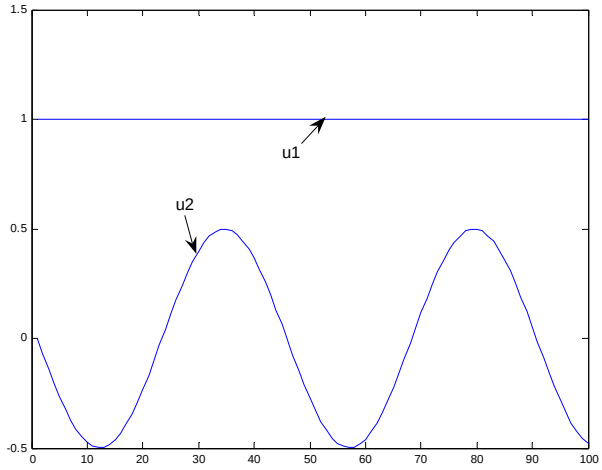

Figure III.10 : Séquences de test (commande u_1, u_2)

La figure.III.11 montre les résultats de test en utilisant un apprentissage hors ligne avec l'algorithme de Levenberg Marquardt.

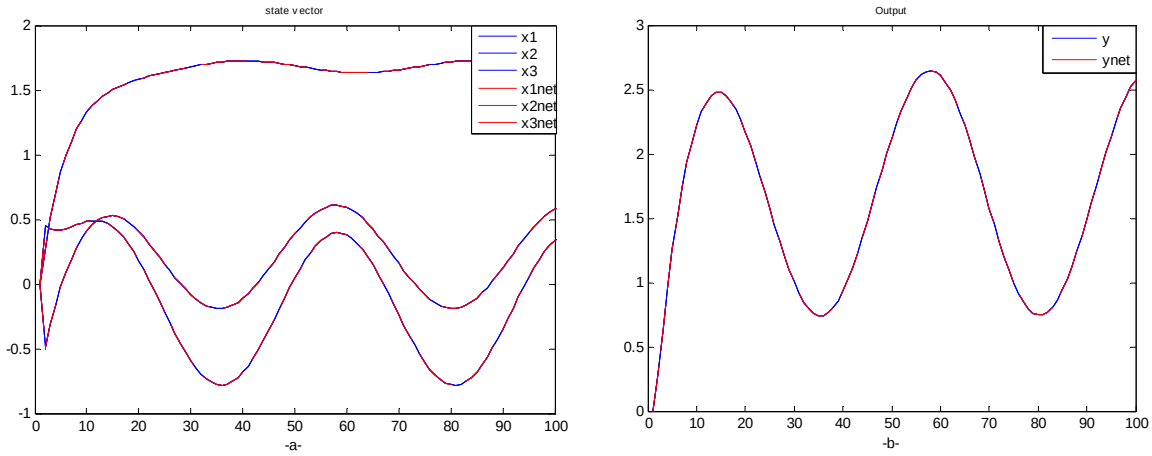

Figure. III.11 : résultats du test obtenus avec la méthode de LM

a: les états x_1, x_2 *et* x_3

b: la sortie y

	EQMT x_1	**EQMT x_2**	**EQMT x_3**	**EQMT y**
L-M	0.56×10^{-31}	0.06×10^{-31}	0.01×10^{-31}	33.6×10^{-31}

Tableau. III.3 : erreurs quadratiques moyennes entre les états réels et les états estimés par un SSNN (2,3,3,2,1)

L'algorithme de Levenberg-Marquardt donne aussi de très bons résultats pour l'identification des systèmes linéaires multivariables, les erreurs quadratiques moyennes de test sont de l'ordre de 10^{-31}.

Les matrices des poids obtenues après l'apprentissage sont les suivantes :

$$W^i = \begin{bmatrix} -1.1597 & 1.2252 \\ 0.7696 & -0.1670 \\ 0.1439 & -0.9618 \end{bmatrix} \qquad W^r = \begin{bmatrix} -2.2527 & -0.0048 & -0.0010 \\ -0.3182 & -0.0066 & -0.0009 \\ -1.6660 & 0.0017 & -0.0001 \end{bmatrix}$$

$$W^h = \begin{bmatrix} -0.1896 & 0.7190 & -0.3820 \\ -0.4196 & -0.0579 & 0.3311 \\ -0.3770 & 1.0536 & 0.5385 \end{bmatrix}$$

$$W^{h2} = \begin{bmatrix} 0.6603 & 0.6289 & 0.5782 \\ 0.8898 & 0.2210 & 0.2773 \\ 0.4873 & -0.3680 & -0.4039 \end{bmatrix}$$

$$W^0 = \begin{bmatrix} 1.0641 & 0.6220 & -0.5255 \end{bmatrix}$$

La représentation d'état du modèle neuronal obtenue avec les poids optimaux est :

$$\begin{cases} \hat{x}(k+1) = \begin{bmatrix} 0.8348 & -0.0045 & -0.0004 \\ 0.4122 & 0.0029 & 0.0004 \\ -0.3832 & -0.0043 & -0.0006 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 0.2785 & 0.0150 \\ -0.4835 & -0.8229 \\ 0.4511 & -1.1558 \end{bmatrix} u(k) \\ \hat{y}(k) = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \hat{x}(k) \end{cases} \quad \dots \text{(III-7)}$$

Nous remarquons que la représentation d'état (III-7) obtenue par les poids optimaux du SSNN est identique à la représentation d'état du système (III-6), donc le SSNN peut reproduire parfaitement le comportement d'un système linéaire multivariables.

Nous avons vus trois exemples de systèmes linéaires, un système de premier ordre, un système de quatrième ordre et un système de troisième ordre avec deux entrées de commande. Pour le système du premier ordre la méthode de L-M a donné de très bons résultats. On peut dire que les résultats obtenus avec la méthode du gradient sont également acceptables mais dés que l'on augmente l'ordre du système au delà du quatrième ordre, cette dernière devient sensible et l'erreur d'estimation augmente. Par contre, les performances obtenues avec la méthode LM restent les mêmes et sont insensibles par rapport à l'ordre de système et au nombre d'entrées de commande.

Pour les systèmes linéaires les SSNN sont des simulateurs idéaux, et puisque tout simulateur est un prédicteur à un pas, les SSNN sont également des prédicteurs à un pas idéaux.

III-3- Influence du bruit

III-3-1- Influence du bruit sur un SSNN

On a vu précédemment que le SSNN est capable d'estimer parfaitement l'état (l'erreur de prédiction égal à 10^{-30} en utilisant l'algorithme de Levenberg-Marquardt) pour les systèmes linéaires déterministes sans bruit. L'étude suivante cherche à évaluer les performances du SSNN en présence d'un bruit de mesure.

Pour cela, nous allons considérer le système linéaire d'ordre trois en lui ajoutant un bruit (w) additif sur la sortie :

$$\begin{cases} x(k+1) = A_D x(k) + B_D u(k) \\ y(k) = C_D x(k) + w(k) \end{cases} \quad \text{(III-8)}$$

Nous allons choisir dans un premier temps un bruit gaussien de variance : $v=0.01$ et de moyenne $m=0$.

Considérons le système continu dont la représentation d'état est définie par :

$$\begin{cases} \dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -3 & -5 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \\ y = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} x \end{cases} \quad \text{(III-9)}$$

La représentation d'état échantillonnée de ce système en utilisant une période d'échantillonnage Te=0.1 s. est la suivante :

$$\begin{cases} x_{k+1} = \begin{bmatrix} 0.9997 & 0.0995 & 0.0043 \\ -0.0085 & 0.9870 & 0.0783 \\ -0.1566 & 0.2434 & 0.5955 \end{bmatrix} x_k + \begin{bmatrix} 0.0001 \\ 0.0043 \\ 0.0783 \end{bmatrix} u_k \\ y_k = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} x_k \end{cases} \quad \text{(III-10)}$$

Le système discrétisé définit ci-dessus est estimé par un SSNN de dimension (1,3,3,2,1) dont la séquence d'entrée de commande utilisée pour l'apprentissage est une suite de créneaux de module et de durée aléatoires (signal1). Les signaux utilisés pour l'apprentissage (les

variables d'état et de sortie) ont été générés par le modèle mathématique du système, l'architecture du réseau de neurones à espace d'état est montrée sur la figure suivante :

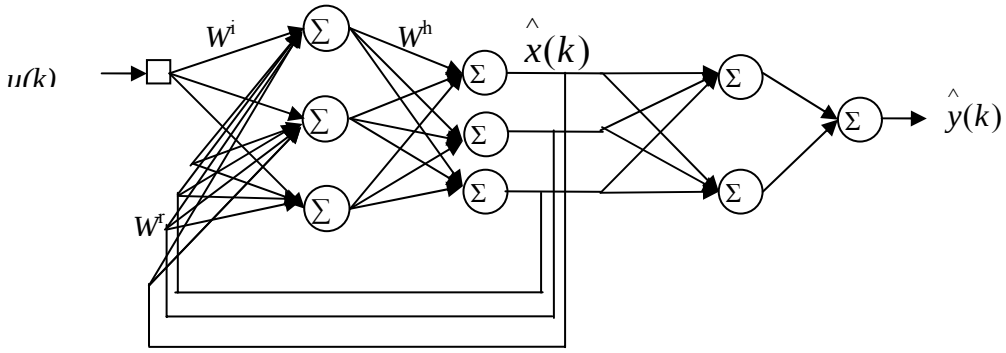

Figure. III.12 : architecture du SSNN (1,3,3,2,1)

Les résultats obtenus sont représenté sur la figure III.13 et le tableau III.4.

	EQMTx1	EQMTx2	EQMTx3	EQMTy	Emax(x1)	Emax(x2)	Emax(x3)
SSNN	0.53×10^{-30}	0.33×10^{-30}	0.09×10^{-30}	1.21×10^{-6}	4.77×10^{-15}	1.47×10^{-15}	1.45×10^{-15}

Tableau.III.4 : erreurs quadratiques moyennes de test et erreurs max entre les états et la sortie réelle et les états et la sortie estimée par un SSNN(1,3,3,2,1)

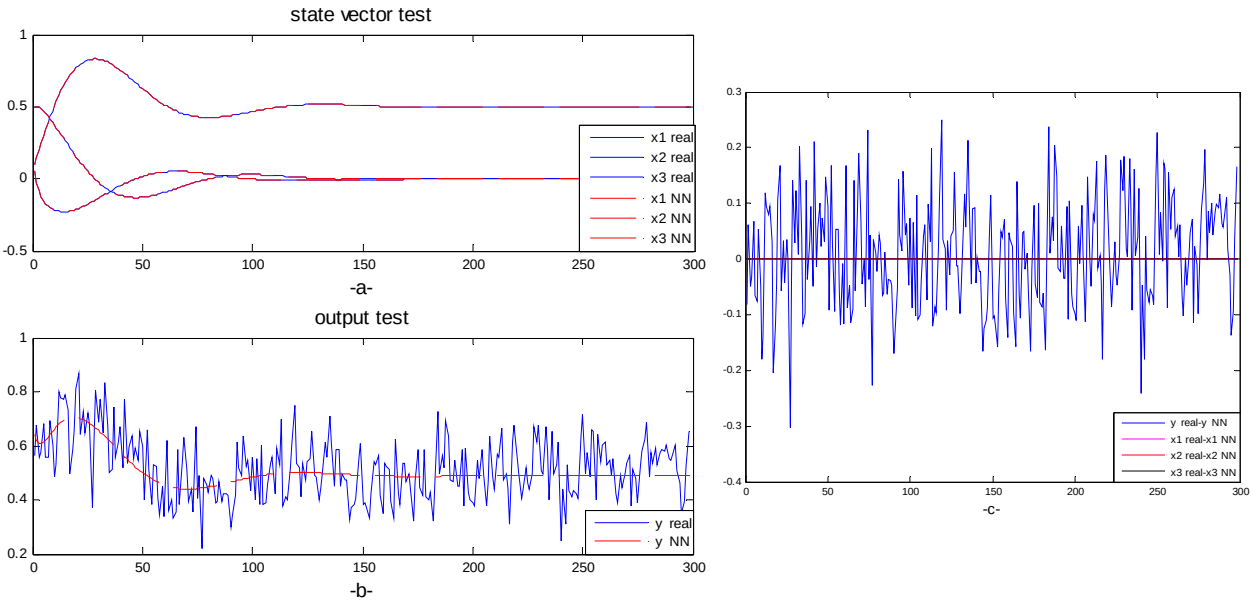

Figure. III.13: -a-b- Les états et la sortie estimés par un SSNN avec un bruit de variance 0.01
-c- Les erreurs entre les états et la sortie estimées et les états et sortie réelle.

Sur la figure. III.13-c- on voit en bleu l'erreur entre la sortie du SSNN et la sortie du système perturbée avec un bruit gaussien additif de variance (0.01). En calculant la variance de cette erreur, on la trouve égale à la variance du bruit additif (0.01), donc le SSNN a bien estimé la sortie ainsi que les états, et le bruit n'as pas d'influence sur le SSNN.

Var(bruit)	EQMTx1	EQMTx2	EQMTx3	EQMTy	Var(yNN-yreal)
0.001	0.76×10^{-29}	0.04×10^{-29}	0.05×10^{-29}	0.001	0.001
0.1	0.76×10^{-29}	0.04×10^{-29}	0.05×10^{-29}	0.1	0.1
0.5	0.76×10^{-29}	0.04×10^{-29}	0.05×10^{-29}	0.5	0.5

Tableau.III.5 : Erreurs quadratiques moyennes de test avec des bruits de différentes variances obtenues avec un SSNN

Le tableau ci-dessus regroupe les erreurs quadratiques moyennes de test sur les états et la sortie avec des bruits gaussiens additifs de variance différentes (0.001, 0.1, 0.5). On remarque que les erreurs restent les mêmes en changeant la variance du bruit, et la variance de la différence entre la sortie estimée par le SSNN et la sortie réelle (bruitée) est égale à la variance du bruit ce qui veux dire que le SSNN a réussit de bien estimer la sortie du système. On peut dire que le SSNN est un outil mathématique robuste par rapport aux bruit de mesure ce qui n'est pas le cas dans la plupart des outils mathématiques tels que les observateurs de Luenberger.

III-3-2- Influence du bruit sur un observateur de Luenberger

Les observateurs d'état et les SSNN sont des techniques très différentes. L'observateur de Luenberger a été mis en œuvre dans un contexte purement déterministe puisqu'il se sert du modèle mathématique du système. Plus précisément, il se sert de la sortie mesurée du système à l'instant *k* pour prédire l'état du système à l'instant (*k+1*).

Le but de cette étude est de montrer la robustesse du SSNN comparée à un autre outil mathématique tel qu'un observateur de Luenberger.

La structure de l'observateur de Luenberger est la suivante :

$$\begin{cases} \hat{x}(k+1) = A_D\,\hat{x}(k) + B_D u(k) + L\left(y(k) - \hat{y}(k)\right) \\ \hat{y}(k) = C_D\,\hat{x}(k) \end{cases} \quad \text{(III-11)}$$

Où $(y_k - \hat{y}_k)$ est l'erreur de prédiction de la sortie et L est la matrice de gain de l'observateur.

D'autre part, l'équation aux différences d'état de l'observateur s'exprime sous la forme :

$$\hat{x}(k+1) = \left[A_D - LC_D\right]\hat{x}(k) + B_D u(k) + LC_D x(k) \quad \text{(III-12)}$$

L'erreur d'estimation de l'état est : $(x_{k+1} - \hat{x}_{k+1}) = [A_D - LC_D](x_k - \hat{x}_k)$

Le principe de l'observateur de Leenberger est représenté par la figure III.14.

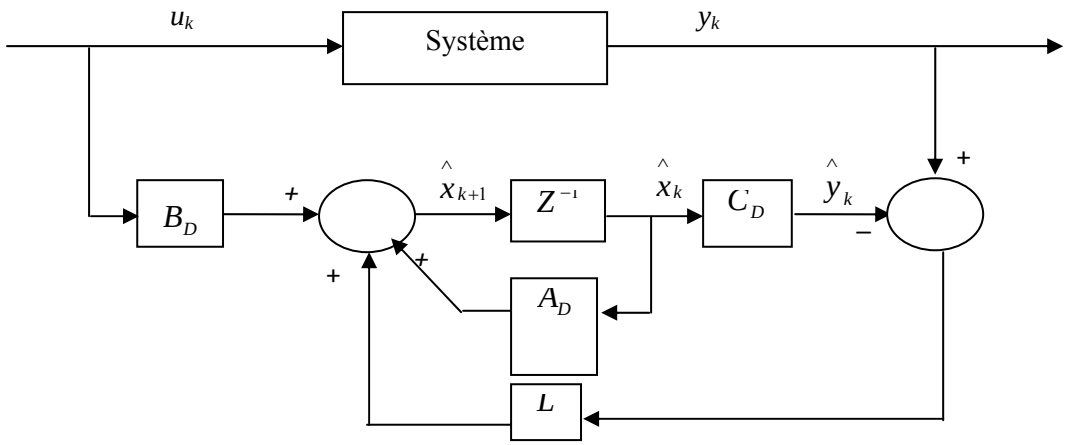

Figure. III.14 : observateur de luenberger

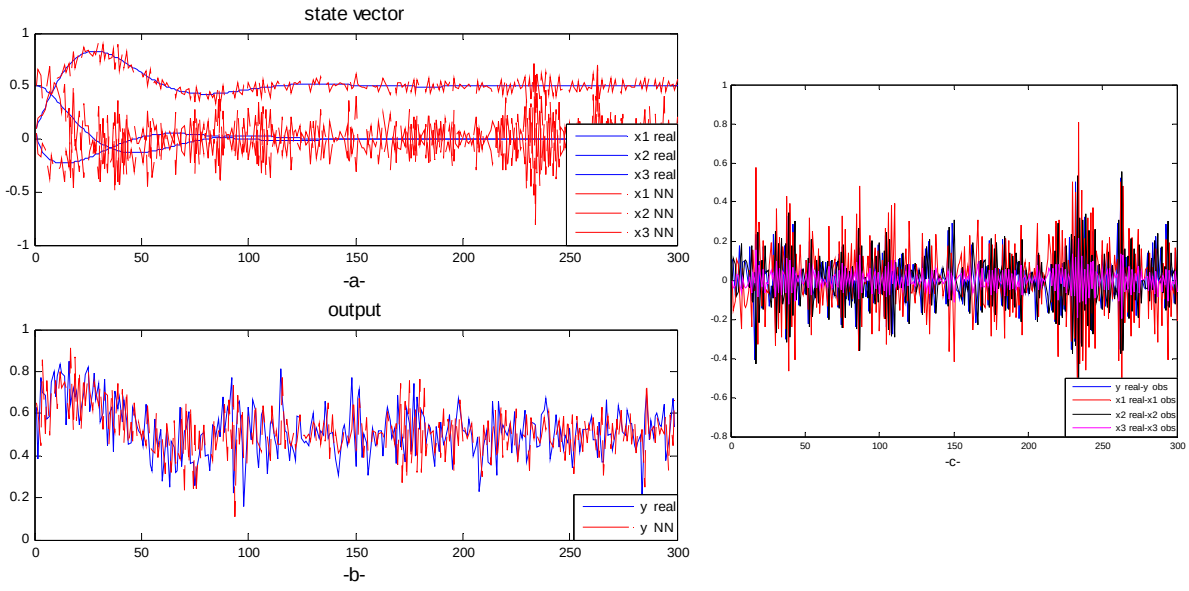

Figure.III.15 :-a-b- Etats et sortie estimés par un observateur avec un bruit de var 0.01

-c- Erreurs entre les états et la sortie estimées et les états et sortie réelle

La figure. III.15-a-b- montre les états estimés par l'observateur et on voit clairement que ces états sont bruités. La figure.III.15-c- montre en bleu l'erreur entre la sortie de l'observateur et la sortie du système perturbée avec un bruit gaussien additif de variance (0.01). En calculant

la variance de cette erreur, on la trouve égal à 0.02. Le bruit est donc amplifié et l'observateur le Luenberger est sensible aux bruits de mesure.

III-4- Identification de systèmes non linéaires

III-4-1- SSNN simulateur

III-4-1-1- Système non linéaire de 4ème ordre

Les objectifs de cette étude sont :

1. d'évaluer les performances d'apprentissage du SSNN,
2. d'évaluer la qualité de l'estimation déduite du SSNN.
3. Pour un système non linéaire.

Le système non linéaire à étudier est le suivant :

$$\mathbf{x}(k+1)=\begin{bmatrix} x_1(k+1)\\ x_2(k+1)\\ x_3(k+1)\\ x_4(k+1)\end{bmatrix}=\begin{bmatrix} 0 & 1 & 0 & 0\\ \dfrac{p\sin x_1(k)+p}{x_1(k)} & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & p & 0\end{bmatrix}\begin{bmatrix} x_1(k)\\ x_2(k)\\ x_3(k)\\ x_4(k)\end{bmatrix}+\begin{bmatrix}0\\0\\0\\1\end{bmatrix}u(k) \quad \text{..... (III-13)}$$

$$y(k)=G(x)$$

L'équation d'état s'écrit également

$$\begin{bmatrix} x_1(k+1)\\ x_2(k+1)\\ x_3(k+1)\\ x_4(k+1)\end{bmatrix}=\begin{bmatrix} x_2(k)\\ p\sin x_1(k)+p+x_3(k)\\ x_4(k)\\ px_3(k)\end{bmatrix}+\begin{bmatrix}0\\0\\0\\1\end{bmatrix}u(k) \quad \text{.....(III-14)}$$

Ou $\quad \mathbf{x}(k+1)=\mathbf{F}\left(\mathbf{x}(k)\right)+\begin{bmatrix}0\\0\\0\\1\end{bmatrix}u(k)$

G(x) sera définie ultérieurement.

Avec la fonction non linéaire $\mathbf{F}(\mathbf{x}(k)) = \begin{bmatrix} x_2(k) \\ p\sin x_1(k) + p + x_3(k) \\ x_4(k) \\ px_3(k) \end{bmatrix}$ ..... (III-15)

Quelques remarques préalables peuvent être faites :

- Ce système est d'ordre 4, mais seuls 2 états sont réellement significatifs puisque $x_1(k+1) = x_2(k)$ et $x_3(k+1) = x_4(k)$.
- S'il fallait le simplifier, ce système pourrait être réduit à un système d'ordre 2 avec les états les plus significatifs (x_1 et x_3).
- Le paramètre p fixe et impose sa non linéarité. Si $p = 0$ le système est linéaire. Si $p \geq 1$ le système est instable.

Les détails concernant le système sont les suivants :

- le système est simulé avec des conditions initiales nulles, c'est-à-dire que $x(0) = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$;
- le système est excité par un signal de commande $u(k)$ sinusoïdal (signalsin.m), montré sur la figure III.16.
- Les performances d'estimation des états et des sorties du système par un SSNN sont évaluées en comparant les réponses à un échelon unitaire et à un signal sinusoïdal.

Cet exemple est très intéressant parce qu'il est possible de varier sa non linéarité en changeant le paramètre *p*. Pour cela nous allons étudier les trois cas suivants qui concernent la sortie du système :

- 1^{er} cas $\mathbf{y}(k) = \begin{bmatrix} x_1(k) & x_2(k) & 0 & 0 \end{bmatrix}^T$ avec $p = 0.85$
- 2^{eme} cas $\mathbf{y}(k) = \begin{bmatrix} x_1(k) & x_2(k) & \tanh(x_3(k)) & \tanh(x_4(k)) \end{bmatrix}^T$ avec $p = 0.85$
- $3^{ème}$ cas $\mathbf{y}(k) = \begin{bmatrix} \tanh(x_1(k)) & \tanh(x_2(k)) & \tanh(x_3(k)) & \tanh(x_4(k)) \end{bmatrix}^T$ avec $p = 0.85$
- $4^{ème}$ cas est le 2^{eme} et $3^{ème}$ cas en faisant varier le paramètre *p=0.5, p=0.85 et p=0.9*

Ces cas sont étudiés selon les deux techniques d'apprentissage.

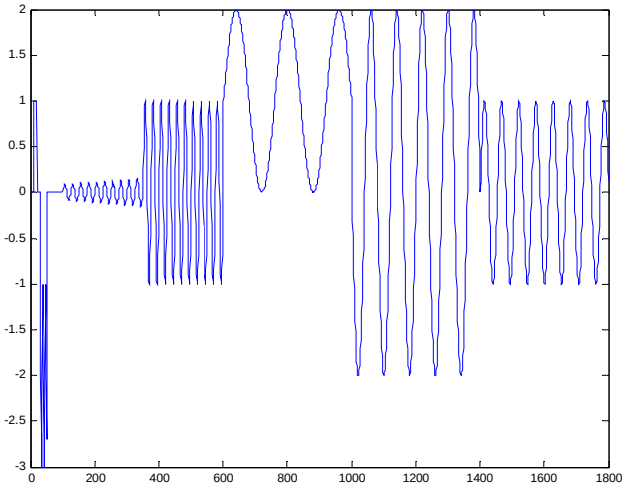

Figure.III.16 : signal d'apprentissage (signalsin)

a- Méthode du gradient

a-1- 1er cas

$$\mathbf{y}(k) = \begin{bmatrix} x_1(k) & x_2(k) & 0 & 0 & 0 \end{bmatrix}^T \text{ avec } p = 0.85$$

Dans ce cas, le système est linéaire entre le vecteur de sortie et le vecteur d'état.

Les détails concernant l'apprentissage sont les suivants :

- le SSNN n'utilise qu'un seul réseau de neurones au vu de la simplicité de la fonction liant l'espace d'état à l'espace de sortie (figure.III.17).
- (conditions initiales) les grandeurs B^h, W^h, W^r et W^i de ANN1 sont initialisées aléatoirement avec des valeurs comprises entre -1 et 1 (W^0, W^{h2} et B^{h2} de ANN2 ne sont pas utilisés) ;
- les 4 neurones de la couche cachée utilisent une fonction d'activation $f_1(.)$ = logsig(.) ;
- les grandeurs nécessaires à l'apprentissage et mesurées à chaque itération sont : $u(k)$, $y(k)$ et $x(k)$;
- la base d'apprentissage est composée de 1800 mesures ;
- la condition d'arrêt de l'apprentissage est fixée à 1000 itérations (l'erreur résultante en sortie est de 0.2×10^{-3} environ) ;

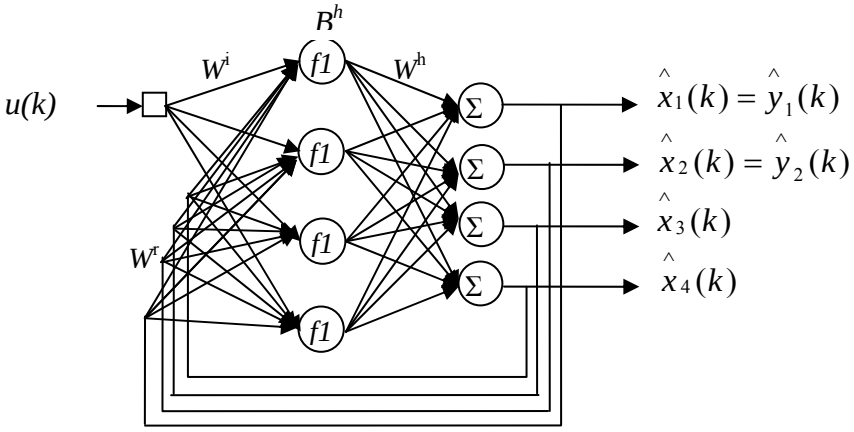

Figure.III.17: l'architecture du SSNN (1,4,4)

La figure III.18 représente les 4 états du système lorsqu'il est excité par un échelon unitaire et un signal sinusoïdal. Sur cette même figure apparaît également les 4 états estimés par le SSNN.

Le Tableau III.6 montre les erreurs quadratiques moyennes entre les états réels et les états estimés par SSNN dans le cas d'une entrée échelon et dans le cas d'une entrée sinusoïdale.

La figure III.19 montre l'évolution des erreurs quadratiques moyennes d'apprentissage sur les quatre états du cas 1 avec une entrée échelon et une entrée sinusoïdale.

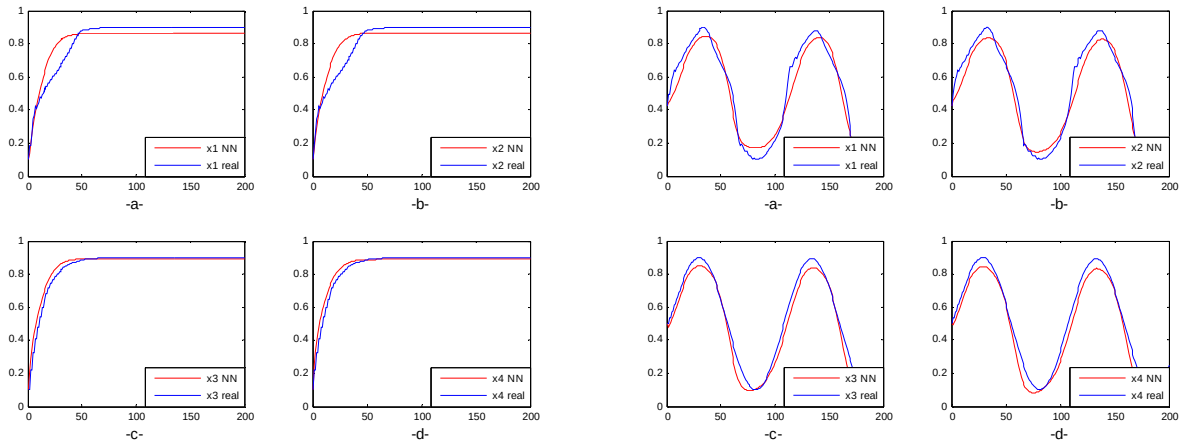

Figure.III.18: Résultats de test obtenus avec la méthode du gradient d'un SSNN(1,4,4)

a, b, c, d (à droite) Réponse à un échelon unitaire

a, b, c, d (à gauche) Réponse à un signal sinusoïdal

	EQMTx1	EQMTx2	EQMTx3	EQMTx4
Echelon	3.7×10^{-3}	3.5×10^{-3}	0.8×10^{-3}	0.7×10^{-3}
Sin	3.8×10^{-3}	3.3×10^{-3}	2.8×10^{-3}	3.1×10^{-3}

Tableau.III.6: Erreurs quadratiques moyennes entre les états réels et les états estimés par SSNN

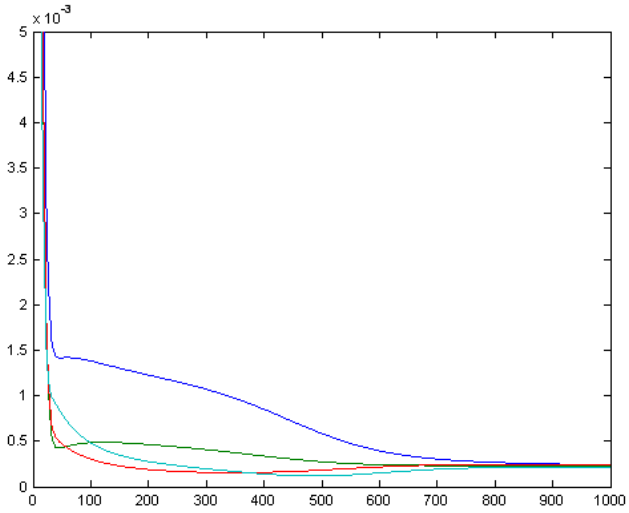

Figure.III.19: L'évolution des erreurs quadratiques moyennes d'apprentissage sur les états du cas 1

L'erreur statique résiduelle entre x_1 et son estimée est de 3.3%. Elle est du même ordre de grandeur pour x_2. Elle est cependant inférieure à 1.2% pour x_3 et x_4. Cette erreur est acceptable en considérant le nombre de paramètres utilisés (4 neurones, soit 36 poids neuronaux).

Le SSNN montre quelques limites dans l'estimation des 4 états dans les périodes transitoires. En effet, l'erreur dynamique maximale vaut 0.17 sur x_1 et x_2 et 0.11 sur x_3 et x_4 (sans jamais dépasser une valeur maximale de 0.3×10^{-3}).

La figure III.20 montre la différence entre les états x_1 et x_2 et celle entre les états x_3 et x_4, que ce soit du système ou estimés par le SSNN. On remarque qu'elles sont corrélées et très proches. On voit que le SSNN n'arrive qu'à reproduire partiellement les relations proches entre ces grandeurs et leurs caractéristiques de dynamique rapide.

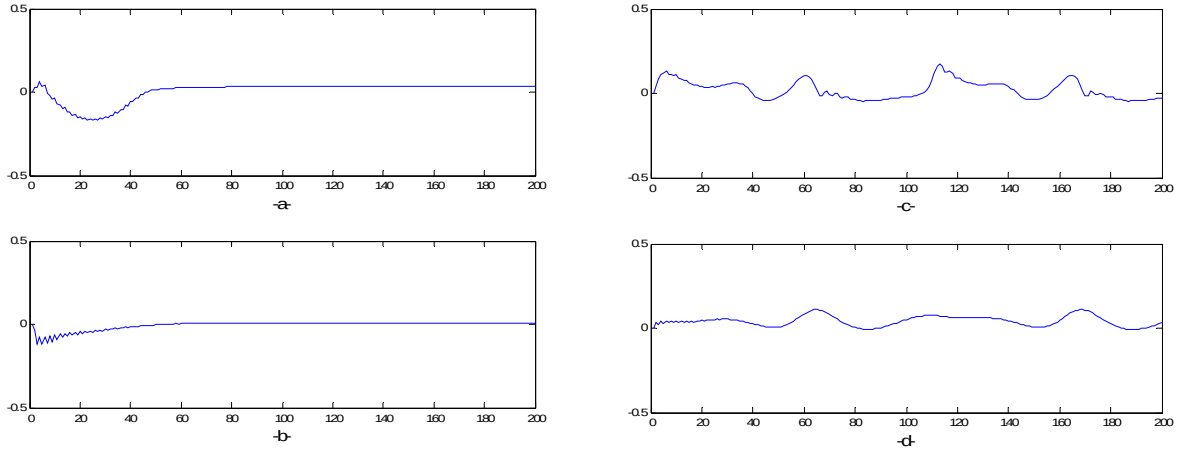

Figure.III.20: Les différences entre les états réels et les états estimés par le SSNN

-a-b- différence $(x_2 - \hat{x}_2)$, $(x_4 - \hat{x}_4)$ entrée échelon

-c-d- différence $(x_2 - \hat{x}_2)$, $(x_4 - \hat{x}_4)$ entrée sinus

La représentation qui peut être formée à partir des poids optimaux du SSNN prend la forme générale suivante :

$$\hat{x}(k+1) = W^h . f_1(W^r . \hat{x}(k) + W^i . \vec{u}(k) + B^h) \quad \text{(III-16)}$$

et avec le choix des fonctions d'activation

$$\hat{x}(k+1) = W^h \ \text{logsig}(W^r . \hat{x}(k) + W^i . \vec{u}(k) + B^h) \quad \text{(III-17)}$$

Soit avec les valeurs numériques suivantes :

$$\hat{x}(k+1) = \begin{bmatrix} -0.0013 & 0.4514 & -0.2952 & 1.0545 \\ 0.4052 & 0.4409 & -0.5000 & 0.7758 \\ 0.4598 & 0.7172 & -0.5632 & 0.4307 \\ 0.0127 & 1.0476 & -0.3150 & 0.2907 \end{bmatrix} \text{logsig}\left\{ \begin{bmatrix} 0.4696 & -0.1461 & 0.4747 \\ 0.2384 & 0.6013 & 0.5734 \\ -0.3562 & -0.0965 & -0.3817 \\ 0.8608 & 1.1491 & 0.3405 \end{bmatrix} . \hat{x}(k) + \begin{bmatrix} 0.2498 \\ 1.2167 \\ -0.2380 \\ -0.9086 \end{bmatrix} . \vec{u}(k) + \begin{bmatrix} -0.3505 \\ -1.8294 \\ 0.7579 \\ -1.3259 \end{bmatrix} \right\} \quad \text{(III-18)}$$

Remarque

La représentation d'état formée à partir des poids neuronaux optimaux du SSNN n'est pas exactement la même que celle du système non linéaire à modélisé, en effet, la représentation d'état d'un système n'est pas unique, bien au contraire pour un système donné il existe une infinité de représentation d'état. Et c'est à partir des tests tel que la réponse indicielle, qu'on peut savoir si les deux représentations d'état représentent le même système où non.

a-2- 2ème cas

$$\mathbf{y}(k) = \left[x_1(k) \quad x_2(k) \quad \tanh(x_3(k)) \quad \tanh(x_4(k)) \right]^T \text{ avec } p = 0.85$$

Les détails concernant l'apprentissage sont les suivants :

- le SSNN utilise deux réseaux de neurones, le premier pour estimer les états *x(k)* et le deuxième pour estimer les sorties *y(k)* (figure III.21) ;
- (conditions initiales) les grandeurs B^h, W^h, W^r, W^i, W^0, W^{h2} et B^{h2} de SSNN sont initialisées aléatoirement avec des valeurs comprises entre -1 et 1.
- les 4 neurones de la couche cachée du premier réseau utilisent une fonction d'activation $f_1(.)$ = logsig(.) , les 4 neurones de la couche cachée du deuxième réseau utilisent une autre fonction d'activation $f_2(.)$ = tansig(.)
- les grandeurs nécessaires à l'apprentissage et mesurées à chaque itération sont : $u(k)$, $y(k)$ et $x(k)$;
- la base d'apprentissage est composée de 1800 mesures ;
- la condition d'arrêt de l'apprentissage est fixée à 1000 itérations.

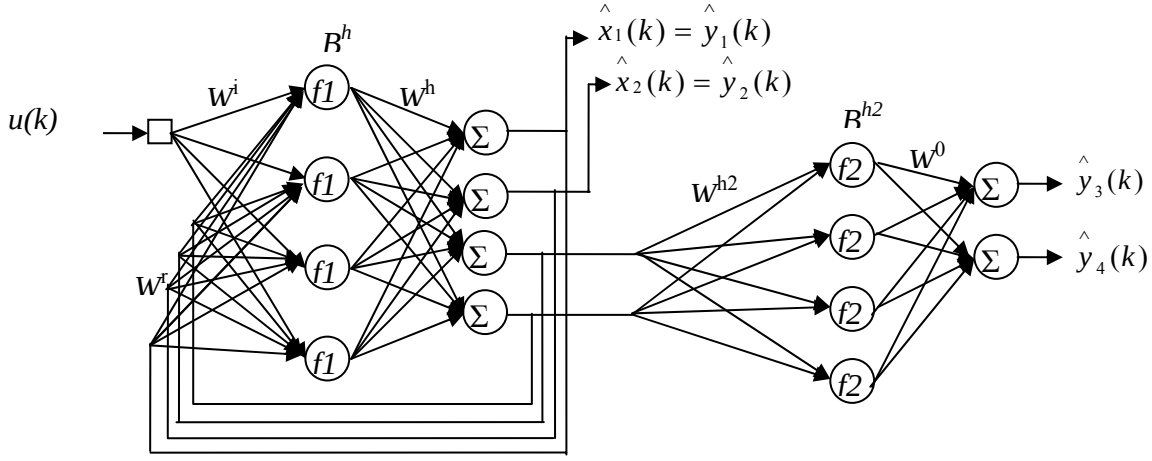

Figure.III.21: l'architecture du SSNN (1,4,4,4,2)

Les performances d'estimation des états du système par un SSNN sont évaluées en comparant les réponses du système et du SSNN à un échelon unitaire et un signal sinusoïdal.

Les courbes a, *b, c* et *d* de la figure III.22 représentent les 4 états du système lorsqu'il est excité par un échelon unitaire. Sur cette même figure apparaît également les 4 états estimés par le SSNN.

Les courbes *e* et *f* de la figure III.22 représentent les sorties y_3, y_4 du système lorsqu'il est excité par un échelon unitaire. Sur cette même figure apparaît également les 4 sorties estimés par le SSNN.

Les sorties y_1 *et* y_2 sont égales aux états x_1 *et x2*

Les erreurs quadratiques moyennes entre les états et les sorties réelles et celles estimées par le SSNN sont regroupées dans le tableau III.7.

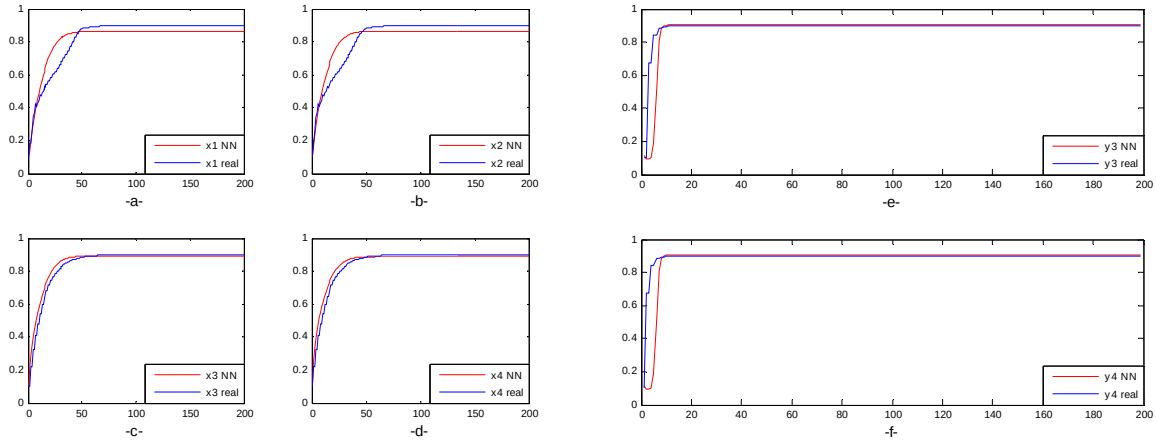

Figure.III.22: Réponses du SSNN (1,4,4,4,2) à un signal échelon

a, b, c, d : les états x_1, x_2, x_3, x_4

e, f : les sorties y_3, y_4

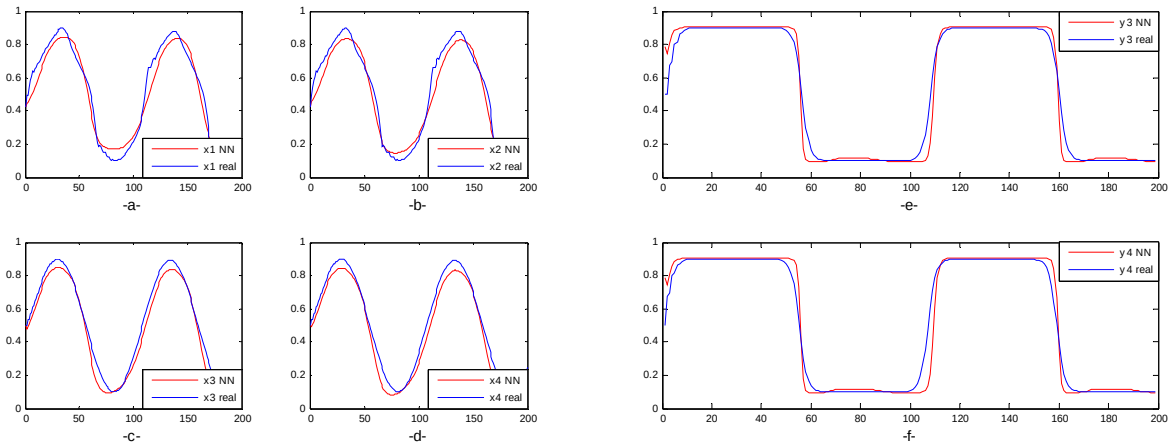

Figure.III.23: Réponses à un signal Sinusoïdal

a, b, c, d : les états x_1, x_2, x_3, x_4

e, f : les sorties y_3, y_4

	EQMT(x_1=y_1)	EQMT(x_2=y_2)	EQMTx3	EQMTx4	EQMTy_3	EQMTy_4
Echelon	3.7×10^{-3}	3.5×10^{-3}	0.8×10^{-3}	0.7×10^{-3}	6.1×10^{-3}	9×10^{-3}
Sin	3.8×10^{-3}	3.3×10^{-3}	2.8×10^{-3}	3.1×10^{-3}	3.8×10^{-3}	4.6×10^{-3}

Tableau.III.7: Erreurs quadratiques moyennes entre les états et les sorties réelles et les états et sorties estimées par le SSNN.

Dans ce cas présent y1=x_1 et x_2=y_2. L'erreur statique résiduelle entre y_1 et son estimé et celle y_2 et son estimé est égal à l'erreur statique résiduelle entre x_1 et son estimé est de l'ordre de 3.5%.

L'erreur résiduelle entre y_3 et son estimé et y_4 et son estimé est nulle.

Le SSNN montre des limites dans l'estimation des sorties durant les périodes transitoires. En effet, l'erreur dynamique maximale vaut 0.17 sur y_1 et y_2, l'estimation est grossière sur y_3 et y_4 où l'erreur maximale dans la période transitoire vaut 0.7 ce qui est une erreur relativement importante.

La figure.III.24 montre la différence entre les sorties y_3 et $\hat{y}_3$ estimé et celle entre la sortie y_4 et $\hat{y}_4$ estimé par le SSNN. On voit que le SSNN n'arrive pas à reproduire le comportement du système en régime transitoire notamment pour y_3 et y_4.

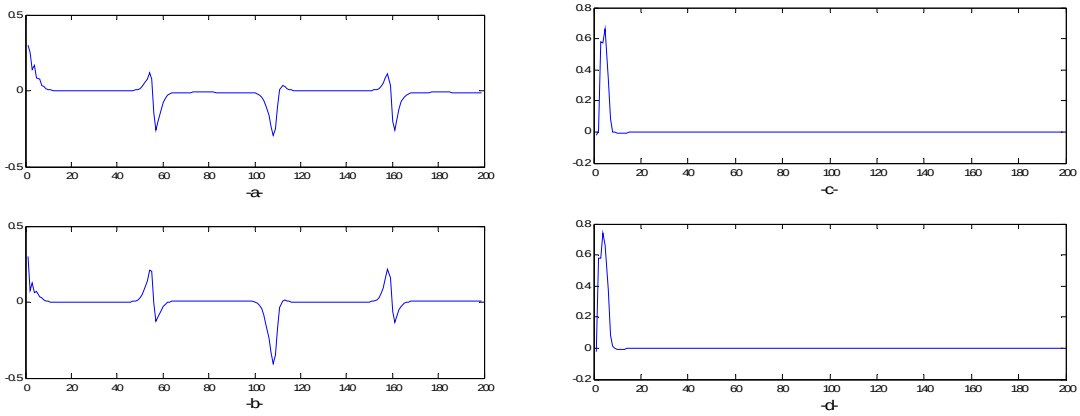

Figure.III.24: La différence entre les sorties réelles et les sorties estimées par le SSNN

a-b- différence $(y_3 - \hat{y}_3)$, $(y_4 - \hat{y}_4)$ entrée sinus

c-d- différence $(y_3 - \hat{y}_3)$, $(y_4 - \hat{y}_4)$, entrée échelon

La représentation qui peut être déduite du SSNN prend la forme générale suivante :

$$\begin{cases} \hat{x}(k+1) = W^h . f_1(W^r . \hat{x}(k) + W^i . \vec{u}(k) + B^h) \\ \hat{y}(k) = W^0 . f_2(W^{h2} . \hat{x}(k) + B^{h2}) \end{cases} \quad \text{.....(III-19)}$$

et avec le choix des fonctions d'activation, on obtient :

$$\begin{cases} \hat{x}(k+1) = W^h.\log sig(W^r.\hat{x}(k) + W^i.\vec{u}(k) + B^h) \\ \hat{y}(k) = W^0.\tan sig(W^{h2}.\hat{x}(k) + B^{h2}) \end{cases} \quad \text{.....(III-20)}$$

Puis finalement :

$$\begin{cases} \hat{x}(k+1) = \begin{bmatrix} -0.0013 & 0.4514 & -0.2952 & 1.0545 \\ 0.4052 & 0.4409 & -0.5000 & 0.7758 \\ 0.4598 & 0.7172 & -0.5632 & 0.4307 \\ 0.0127 & 1.0476 & -0.3150 & 0.2907 \end{bmatrix}.\log sig\left(\begin{bmatrix} 0.4696 & -0.1461 & 0.4747 & -0.2644 \\ 0.2384 & 0.6013 & 0.5734 & 0.8446 \\ -0.3562 & -0.0965 & -0.3817 & -0.6196 \\ 0.8608 & 1.1491 & 0.3405 & 0.4626 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 0.2498 \\ 1.2167 \\ -0.2380 \\ -0.9086 \end{bmatrix} \vec{u}(k) + \begin{bmatrix} -0.3505 \\ -1.8294 \\ 0.7579 \\ -1.3259 \end{bmatrix} \right) \\ \hat{y}(k) = \begin{bmatrix} 0.0238 & 0.1523 & 0.4058 & -0.6734 \\ 0.0232 & 0.0736 & 0.4040 & -0.5961 \end{bmatrix}.\log sig\left(\begin{bmatrix} -11.2049 & -11.2542 \\ -10.1628 & -10.2994 \\ 12.3684 & 12.8620 \\ -10.0137 & -10.2852 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 2.4633 \\ -0.1005 \\ -11.6312 \\ -0.3408 \end{bmatrix} \right) \end{cases}$$

.....(III-21)

L'équation ci-dessus est la représentation d'état formée à partir des poids neuronaux optimaux du SSNN (1,4,4,4,2). On remarque que l'équation d'état de (III-21) est exactement la même que l'équation d'état de (III-18), c'est logique parce que :

- l'architecture du réseau utilisé dans le premier cas et l'architecture du premier sous réseau utilisé dans le deuxième cas sont les mêmes.
- les conditions initiales d'apprentissage sont les mêmes.
- la méthode d'apprentissage utilisée est aussi la même.

a-3- 3ème cas

$$\mathbf{y}(k) = \left[\tanh(x_1(k)) \quad \tanh(x_2(k)) \quad \tanh(x_3(k)) \quad \tanh(x_4(k))\right]^T \text{ avec } p=0.85.$$

Les détails concernant l'apprentissage sont les suivants :

- le SSNN utilise soit trois réseaux de neurones, le premier pour estimer les états *x(k)* et le deuxième pour estimer les sorties *y1(k), y2(k)* et le troisième pour estimer les sorties *y3(k), y4(k)* (figure.III.25).
- (conditions initiales) les grandeurs B^h, W^h, W^r, W^i, W^0, W^{h2}, W^0, W^{h2} et B^{h2} de SSNN sont initialisées aléatoirement avec des valeurs comprises entre -1 et 1.
- les neurones de la couche cachée du premier réseau utilisent une fonction d'activation $f_1(.)$ = logsig(.), les neurones de la couche cachée du deuxième et troisième réseau utilisent une autre fonction d'activation $f_2(.)$ = tansig(.)
- la base d'apprentissage est composée de 1800 mesures ; la condition d'arrêt de l'apprentissage est fixée à 1000 itérations.

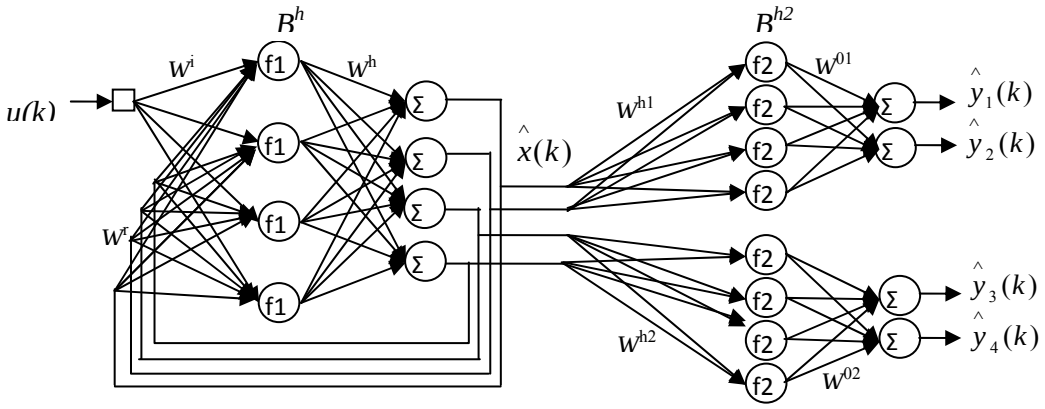

Figure.III.25: l'architecture du SSNN

La figure III.26.*a, b, c* et *d* droite représente les 4 sorties du système lorsqu'il est excité par un échelon unitaire. Sur cette même figure apparaît également les 4 sorties estimées par le SSNN.

La figure III.26.*a, b, c* et *d* a gauche représente les sorties y_3, y_4 du système lorsqu'il est excité par un échelon unitaire. Sur cette même figure apparaît également les 4 sorties estimées par le SSNN.

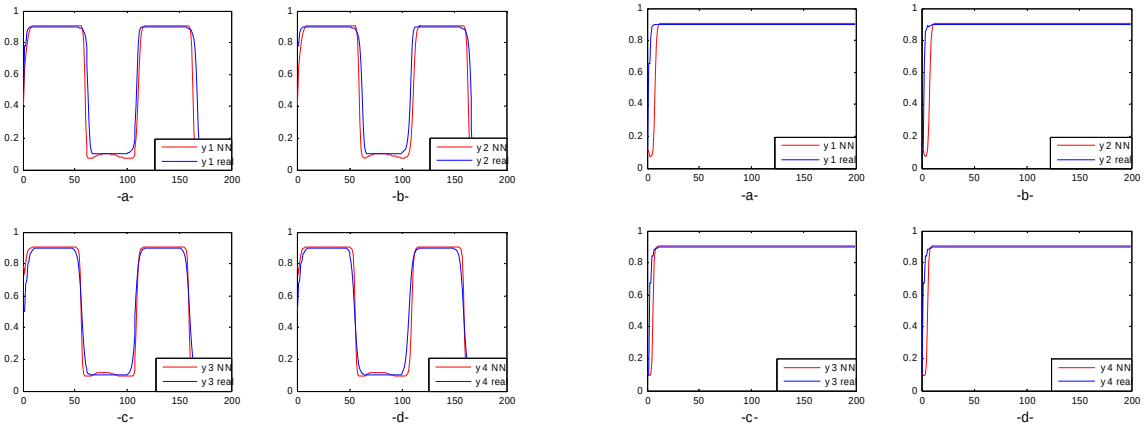

Figure.III.26: Les sorties y_1, y_2, y_3, y_4 du SSNN

a, b, c, d (gauche) : entrée sinus

a, b, c, d (droite): entrée échelon

Le tableau suivant montre les erreurs quadratiques moyennes sur les états ainsi que sur les sorties.

	$EQMTx_1$	$EQMTx_2$	$EQMTx_3$	$EQMTx_4$	$EQMTy_1$	$EQMTy_2$	$EQMTy_3$	$EQMTy_4$
sin	3.8×10^{-3}	3.4×10^{-3}	2.4×10^{-3}	2.7×10^{-3}	16×10^{-3}	13×10^{-3}	3.9×10^{-3}	4.5×10^{-3}
échelon	3.7×10^{-3}	3.5×10^{-3}	0.8×10^{-3}	0.7×10^{-3}	15×10^{-3}	13×10^{-3}	6.1×10^{-3}	9×10^{-3}

Tableau.III.8: Les erreurs quadratiques moyennes entre les états et les sorties réels et les états et sorties estimés par le SSNN.

Les erreurs résiduelles entre y_1 et son estimé, y_2 et son estimé, y_3 et son estimé et y_4 et son estimé sont nulles.

Le SSNN montre des limites dans l'estimation des 4 sorties dans les périodes transitoires. En effet, l'erreur dynamique maximale vaut 0.8.

La figure.III.27 montre la différence entre la sortie y_3 et $\hat{y}_3$ estimée et celle entre la sortie y_4 et $\hat{y}_4$ estimée par le SSNN. On voit clairement que le SSNN est incapable de reproduire le comportement du système en régime transitoire.

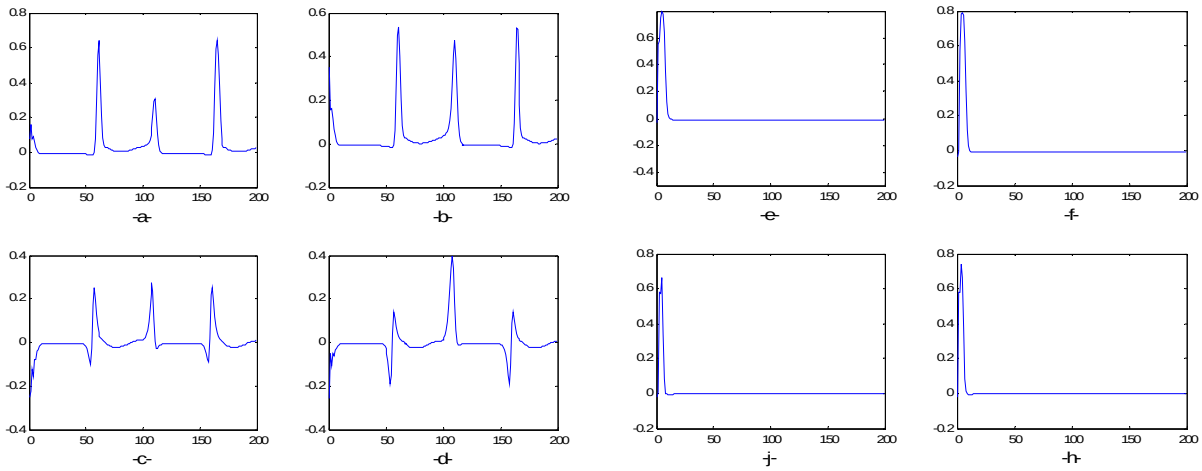

Figure.III.27: les différences les sorties réelles et estimées par le SSNN

a-b-c-d : entrée sinus

e-f-g-h : entrée échelon

La représentation qui peut être déduite du SSNN prend la forme générale suivante :

$$\begin{cases} \hat{x}(k+1) = W^h . f_1(W^r . \hat{x}(k) + W^i . \vec{u}(k) + B^h) \\ \hat{y}(k) = W^0 . f_2(W^{h2} . \hat{x}(k) + B^{h2}) \end{cases} \quad \text{..... (III-22)}$$

et avec le choix des fonctions d'activation elle s'écrit

$$\begin{cases} \hat{x}(k+1) = W^h . \log sig(W^r \hat{x}(k) + W^i u(k) + B^h) \\ \hat{y}(k) = W^0 . \tan sig(W^{h2} . \hat{x}(k) + B^{h2}) \end{cases} \quad \text{.....(III-23)}$$

$$\begin{cases} \hat{x}(k+1) = \begin{bmatrix} -0.0013 & 0.4514 & -0.2952 & 1.0545 \\ 0.4052 & 0.4409 & -0.5000 & 0.7758 \\ 0.4598 & 0.7172 & -0.5632 & 0.4307 \\ 0.0127 & 1.0476 & -0.3150 & 0.2907 \end{bmatrix} . \log sig \left(\begin{bmatrix} 0.4696 & -0.1461 & 0.4747 & -0.2644 \\ 0.2384 & 0.6013 & 0.5734 & 0.8446 \\ -0.3562 & -0.0965 & -0.3817 & -0.6196 \\ 0.8608 & 1.1491 & 0.3405 & 0.4626 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 0.2498 \\ 1.2167 \\ -0.2380 \\ -0.9086 \end{bmatrix} \vec{u}(k) + \begin{bmatrix} -0.3505 \\ -1.8294 \\ 0.7579 \\ -1.3259 \end{bmatrix} \right) \\ \hat{y}(k) = \begin{bmatrix} 0.0238 & 0.1523 & 0.4058 & -0.6734 \\ 0.0232 & 0.0736 & 0.4040 & -0.5961 \end{bmatrix} . \tan sig \left(\begin{bmatrix} -11.2049 & -11.2542 \\ -10.1628 & -10.2994 \\ 12.3684 & 12.8620 \\ -10.0137 & -10.2852 \end{bmatrix} \hat{x}(k) + \begin{bmatrix} 2.4633 \\ -0.1005 \\ -11.6312 \\ -0.3408 \end{bmatrix} \right) \end{cases}$$

.....(III-24)

Rappelons que les poids initiaux sont les mêmes que dans le cas de la simulation précédente. Le tableau suivant montre les erreurs quadratiques moyennes sur les états ainsi que sur les sorties dans les deux cas : 2eme cas et le 3ème cas, avec des valeurs différente de *p* (0.55, 0.85, 0.95) c'est-à-dire en faisant varier la non linéarité du système.

	Erreurs moyennes sur les états (x1, x2, x3, x4) $\times 10^{-3}$												Erreurs moyennes sur les sorties (y1, y2, y3, y4) $\times 10^{-3}$											
p	**0.55**				**0.85**				**0.95**				**0.55**				**0.85**				**0.95**			
	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4	y_1	y_2	y_3	y_4	y_1	y_2	y_3	y_4
2er	0.7	0.9	0.4	0.5	3.7	3.5	0.8	0.7	4.9	4.9	8.3	8.5	0.7	0.9	0.4	0.5	3.7	3.5	0.8	0.7	4.9	4.9	8.3	8.5
3ème	0.7	0.9	0.4	0.5	3.7	3.5	0.8	0.7	7.2	6.6	9.3	9.1	5.2	4.8	0.7	2.2	15	13	6.1	9	21	20	11	14

Tableau.III.9: les erreurs quadratiques moyennes entre les états et les sorties réels et les états et sorties estimés par le SSNN selon la valeur de *p.*

- **Interprétation des résultats**

Après l'étude de ce système et les résultats de l'identification obtenus, on peut dire que la méthode du gradient ne convient pas à l'identification des systèmes dynamiques fortement non linéaires avec un SSNN, notamment en régime transitoire où nous remarquons que le SSNN est incapable de suivre la dynamique du système (l'erreur est très grande).

Le tableau III.9 montre les erreurs quadratiques moyennes sur les états et les sorties, ces erreurs augmentent en augmentant le paramètre *p* de 0.85 à 0.95 (en augmentant la non linéarité du système) et diminuent quand *p* égal à 0.55. Finalement, on peut dire que le SSNN devient sensible en augmentant la non linéarité du système, et il n'est pas capable d'estimer les dynamiques rapides.

b- La méthode de Levenberg-Marquardt

b-1- 1er cas

$\mathbf{y}(k) = \begin{bmatrix} x_1(k) & x_2(k) & 0 & 0 0 \end{bmatrix}^T$ avec $p = 0.85$

Les détails concernant l'apprentissage sont les suivants :

- le SSNN n'utilise qu'un seul réseau de neurones au regard de la simplicité de la fonction liant l'espace d'état à l'espace de sortie (figure.III.28) ;
- on ajoute un biais B^1 sur la couche de sortie (dans ce cas c'est la couche d'état)
- (conditions initiales) les grandeurs B^h , B^1, W^h , W^r et W^i de ANN1 sont initialisées avec des valeurs comprises entre -1 et 1
- les 4 neurones de la couche cachée utilisent une fonction d'activation $f_1(.)$ = logsig(.) ;
- les grandeurs nécessaires à l'apprentissage sont générées à partir de la représentation d'état du système : $u(k)$, $y(k)$ et $x(k)$;
- la base d'apprentissage est composée de 1800 mesures (signalsin.m) ;
- la condition d'arrêt de l'apprentissage est fixée à 1000 itérations (l'erreur résultante en sortie est de 10^{-2} environs) ; en utilisant la méthode de Levenberg-Marquardt de la boite à outils de Matlab appelée " Neural Network Toolbox".

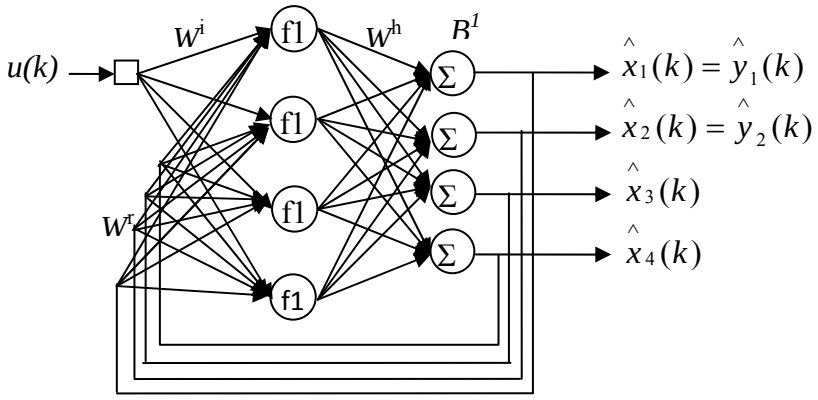

Figure.III.28: l'architecture du SSNN pour estimer le système simple du cas 1.

La figure III.29 représente les 4 états du système lorsqu'il est excité par un échelon unitaire et un signal sinusoïdal. Sur cette même figure apparaît également les 4 états estimés par le SSNN.

Le Tableau III.10 montre les erreurs quadratiques moyennes entre les états réels et les états estimés par le SSNN dans le cas d'une entrée échelon et dans le cas d'une entrée sinusoïdale.

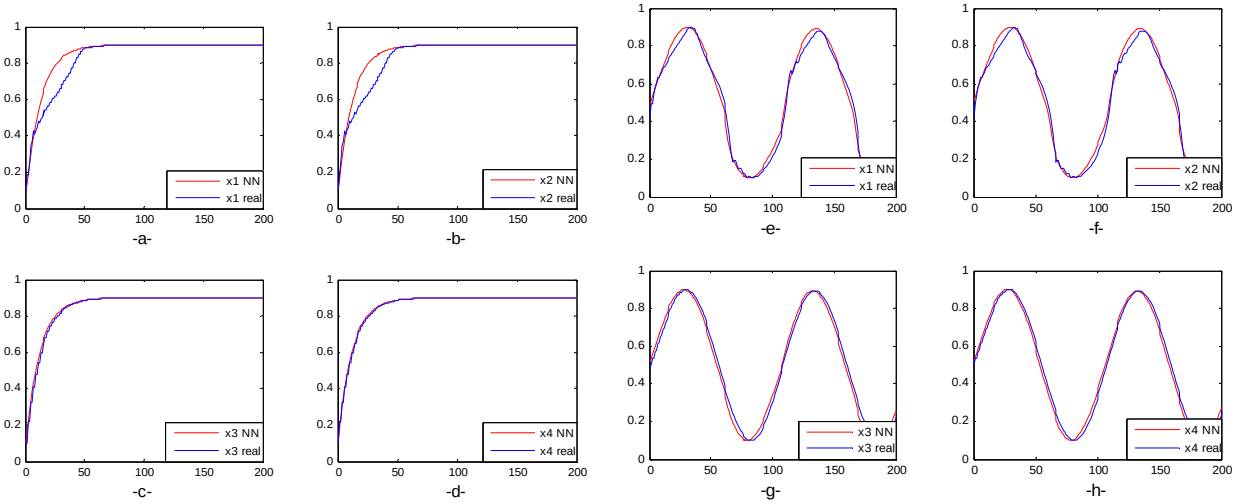

Figure.III.29: résultats de test obtenus avec la méthode de LM pour estimer le système du 1er cas

a, b, c, d : réponse à un échelon unitaire

e, f, g, h : réponse à un signal Sinusoïdal

	EQMTx1	EQMTx2	EQMTx3	EQMTx4
Echelon	2.9×10^{-3}	2.5×10^{-3}	0.1×10^{-3}	0.1×10^{-3}
Sin	1.1×10^{-3}	1.0×10^{-3}	1.1×10^{-3}	2.1×10^{-3}

Tableau.III.10: les erreurs quadratiques moyennes entre les états et les sorties réels et les états et sorties estimés

La figure.III.30 montre la différence entre les états x_1 et x_2 et celle entre les états x_3 et x_4, que ce soit du système ou estimés par le SSNN. On remarque qu'elles sont très proches.

Les erreurs statiques résiduelles entre x_1 et son estimée, x_3 et son estimé sont nulles. Elles sont du même ordre de grandeur pour x_2 et x_4. L'erreur dynamique maximale vaut 0.17 sur x_1 et x_2 et 0.04 sur x_3 et x_4.

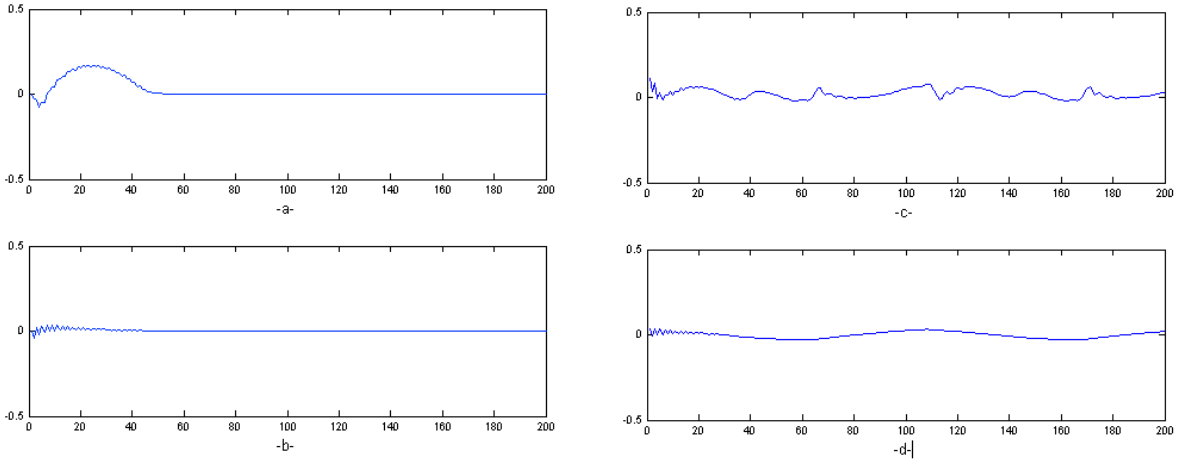

Figure.III.30: les différences entre les états réels et les états estimés par le SSNN

-a-b- différence $(x_2 - \hat{x}_2)$, $(x_4 - \hat{x}_4)$ entrée échelon

-c- d-différence $(x_2 - \hat{x}_2)$, $(x_4 - \hat{x}_4)$ entrée sinus

La représentation qui peut être déduite du SSNN prend la forme suivante :

$$x(k+1) = W^h . f1(W^r . \hat{x}(k) + W^i . \vec{u}(k) + B^h) + B^1 \quad \text{..... (III-25)}$$

On appelle cette représentation d'état "représentation d'état affine" dont la forme générale est la suivante :

$$\begin{cases} x(k+1) = F(x(k), u(k)) + C \\ y(k) = G(x(k)) + D \end{cases} \quad \text{..... (III-26)}$$

C et D sont des matrices constantes. Les estimations après apprentissage des poids du SSNN sont :

$$W^i = \begin{bmatrix} -1.0190 \\ 0.0100 \\ 0.0145 \\ -0.6923 \end{bmatrix} \qquad W^r = \begin{bmatrix} -6.0077 & 4.6747 & -8.2025 & 3.9456 \\ 0.0001 & -0.0043 & 0.0085 & 0.0070 \\ 0.0148 & -0.0946 & -0.0019 & 0.0262 \\ 2.6050 & -1.4413 & 0.9095 & 0.5882 \end{bmatrix}$$

$$W^h = \begin{bmatrix} -0.2785 & 83.0071 & -57.1716 & 0.9309 \\ -0.6100 & 133.5113 & -43.1038 & 0.2914 \\ -0.0617 & 230.7391 & -20.7870 & 0.2914 \\ -0.0680 & 272.9594 & -9.5428 & 0.1870 \end{bmatrix} \quad B^h = \begin{bmatrix} -4.0224 \\ -0.0015 \\ 0.7382 \\ 17.5569 \end{bmatrix}$$

$$B^1 = \begin{bmatrix} -3.6017 \\ -37.8664 \\ -101.6331 \\ -129.9741 \end{bmatrix}$$

Le SSNN montre quelques limites dans l'estimation des 4 états dans les périodes transitoires, mais en comparant avec les résultats trouvés avec la méthode du gradient, ceux trouvés avec la méthode de Levenberg-Marquardt sont meilleurs dans les deux régimes, transitoire et permanent.

b-2- 2ème cas

$$\mathbf{y}(k) = \left[x_1(k) \quad x_2(k) \quad \tanh(x_3(k)) \quad \tanh(x_4(k)) \right]^T \text{ avec } p = 0.85$$

Les détails concernant l'apprentissage sont les suivants :

- le SSNN utilise deux réseaux de neurones, le premier pour estimer les états *x(k)* et le deuxième pour estimer les sorties *y(k)* (figure III.31) ;
- les poids sont initialisés aléatoirement avec des valeurs comprises entre -1 et 1
- les 4 neurones de la couche cachée du premier réseau utilisent une fonction d'activation $f_1(.)$ = logsig(.) , les 4 neurones de la couche cachée du deuxième réseau utilisent une autre fonction d'activation $f_2(.)$ = tansig(.)
- les grandeurs nécessaires à l'apprentissage et mesurées à chaque itération sont : $u(k)$, $y(k)$ et $x(k)$;
- la base d'apprentissage est toujours la même, composée de 1800 mesures (signalsin.m) ;
- la condition d'arrêt de l'apprentissage est fixée à 1000 itérations (l'erreur résultante en sortie est de 10^{-2} environs) ; en utilisant la méthode de Levenberg-Marquardt de la boite à outils de Matlab appelée " Neural Network Toolbox".

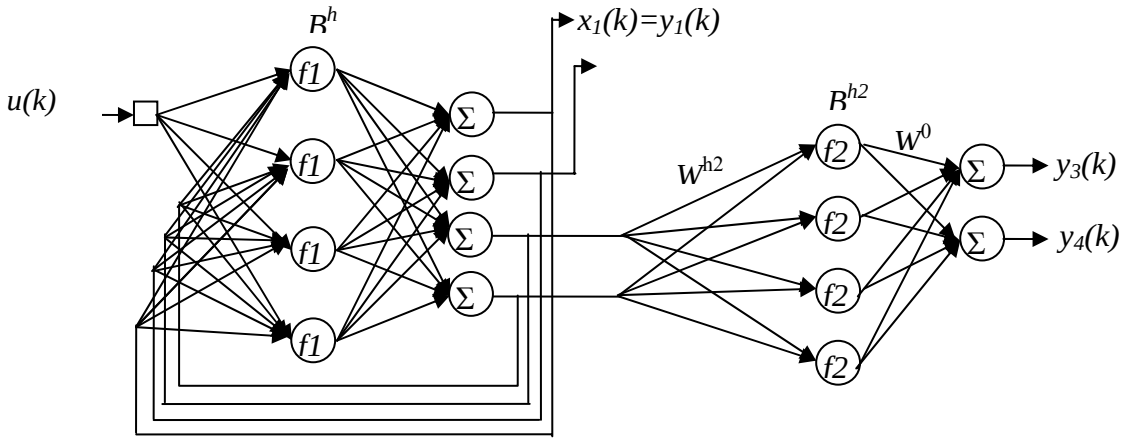

Figure.III.31: l'architecture du SSNN pour estimer le système simple du cas 2.

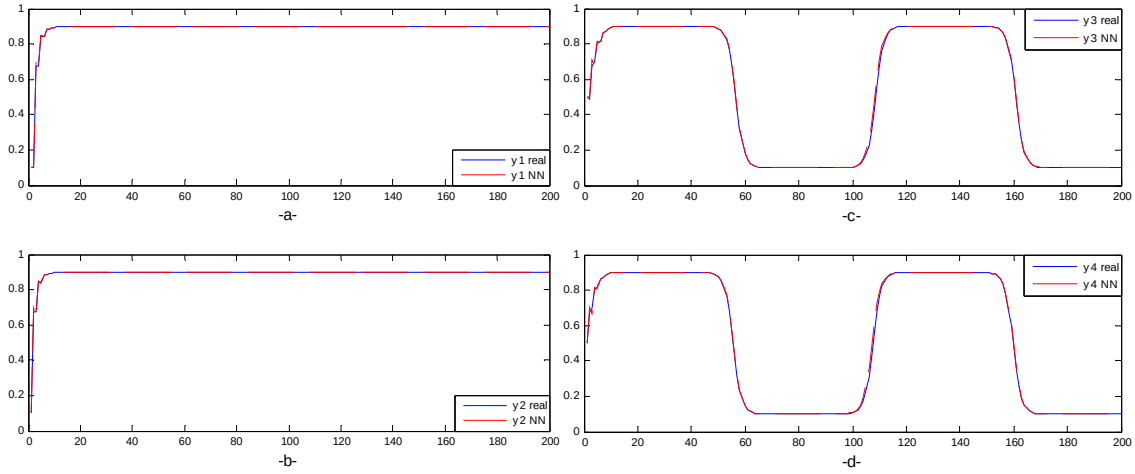

Figure.III.32: résultats de test obtenus avec la méthode de LM pour estimer le système du 2ème cas

a, b: réponse à un échelon unitaire

c, d : réponse à un signal Sinusoïdal

Le Tableau III.11 montre les erreurs quadratiques moyennes entre les sorties réels et les sorties estimés par le SSNN dans le cas d'une entrée échelon et dans le cas d'une entrée sinusoïdale.

	EQMTy3	EQMTy4
Echelon	0.62×10^{-5}	0.92×10^{-5}
Sin	5.6×10^{-4}	1.2×10^{-4}

Tableau.III.11: les erreurs quadratiques moyennes entre les sorties réelles et les sorties estimées

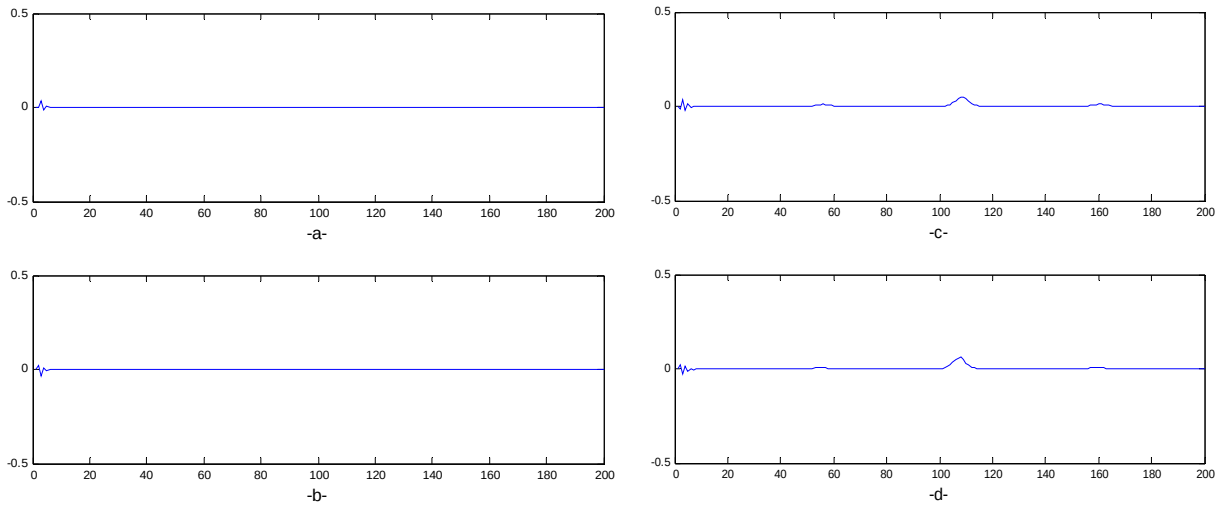

Figure.III.33: différence entre les sorties réelles et estimées obtenues avec le SSNN(1,4,4,4,2) pour le système du 2ème cas.

La figure ci-dessus montre la différence entre les sorties y_3 et y_4 *et* leurs estimés $\hat{y}_3$ et $\hat{y}_4$ par le SSNN. On voit que le SSNN a réussi à reproduire parfaitement le comportement du système en régime permanent et transitoire. En effet, l'erreur en régime permanent est nulle et l'erreur en régime transitoire ne dépasse pas la valeur maximale 0.037, soit inférieur à 0.37%.

Les matrices des poids obtenues après l'apprentissage sont les suivantes :

$$W^i = \begin{bmatrix} 0.0156 \\ 0.0377 \\ -0.0149 \\ 10.5630 \end{bmatrix} \qquad W^r = \begin{bmatrix} -0.0023 & -0.0080 & 0.0155 & -0.0148 \\ -0.0222 & 0.0673 & 0.0594 & -0.0988 \\ 0.0001 & 0.0103 & -0.0128 & -0.0031 \\ -13.6996 & 14.7950 & 1.0547 & -13.8548 \end{bmatrix}$$

$$W^h = \begin{bmatrix} -334.4226 & 200.9833 & -161.0453 & 0.0031 \\ -230.0348 & 269.0608 & -212.7009 & -0.3815 \\ -287.9525 & 25.7113 & -272.5775 & 0.0117 \\ -172.4608 & 134.5577 & -322.4193 & 0.0017 \end{bmatrix} \qquad B^h = \begin{bmatrix} -0.6584 \\ -2.5907 \\ -0.0046 \\ -10.0014 \end{bmatrix}$$

$$B^1 = \begin{bmatrix} 180.4371 \\ 166.1248 \\ 232.3823 \\ 210.3156 \end{bmatrix} \qquad W^{h2} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 3.1694 & -1.8463 \\ 0 & 0 \end{bmatrix}$$

$$W^0 = \begin{bmatrix} -1 & 0 & 0 & 0.8473 \\ 0 & -1 & 0 & -0.5303 \end{bmatrix} \qquad B^{h2} = \begin{bmatrix} 0 \\ 0 \\ 03.3880 \\ -1.9788 \end{bmatrix} \qquad B^2 = \begin{bmatrix} 0.8155 \\ -0.5104 \end{bmatrix}$$

b-3- 3ème cas

$\mathbf{y}(k) = \left[\tanh(x_1(k)) \quad \tanh(x_2(k)) \quad \tanh(x_3(k)) \quad \tanh(x_4(k))\right]^T$ avec p=0.85

Les détails concernant l'apprentissage sont les mêmes que dans le cas précédent (2ème cas)

La figure.III.34 montre l'architecture du SSNN (1,4,4,4,4) utilisée pour la modélisation du système du 3ème cas.

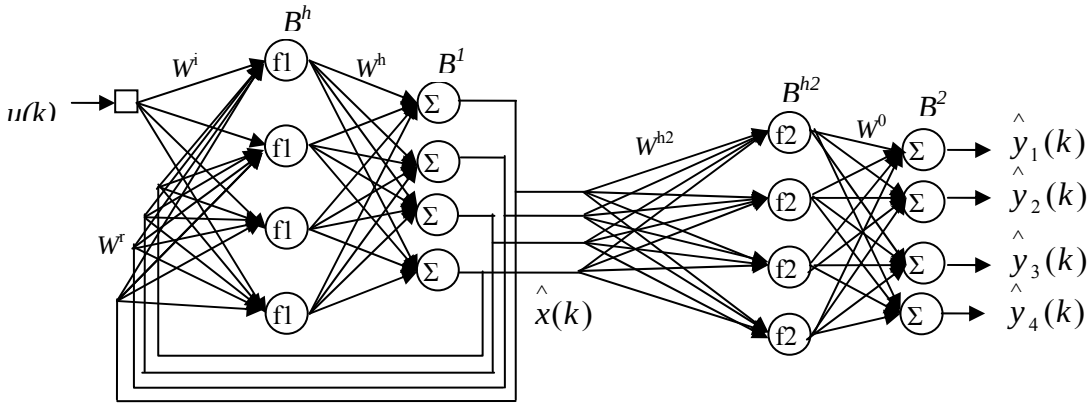

Figure.III.34: Architecture du SSNN (1,4,4,4,4) pour estimer le système du 3ème cas

La figure III.35 représente les 4 sorties du système lorsqu'il est excité par un échelon unitaire et un signal sinusoïdal. Sur cette même figure apparaît également les 4 sorties estimées par le SSNN (1,4,4,4,4).

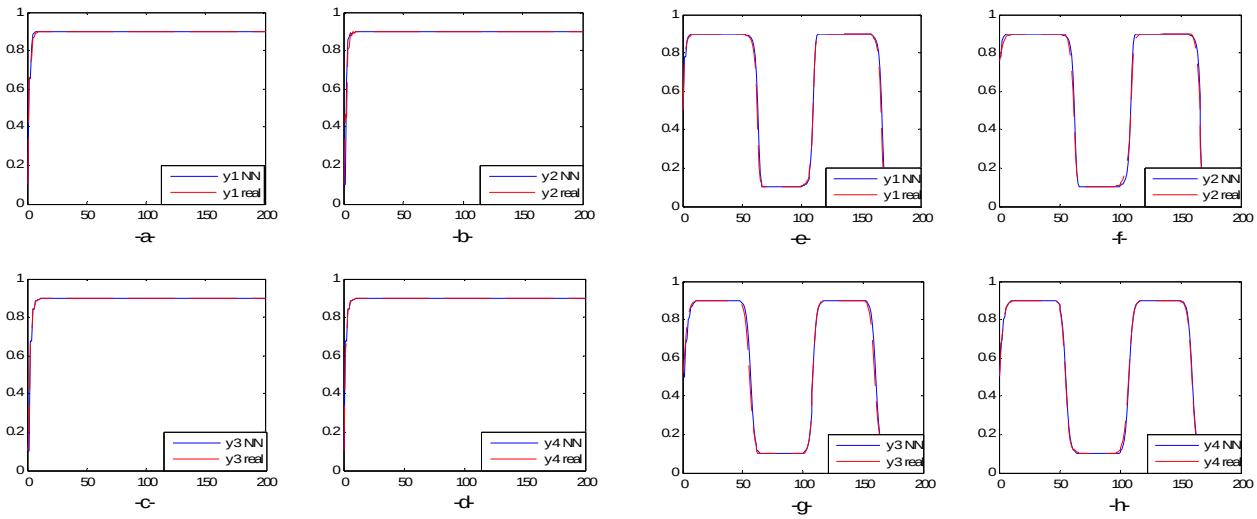

Figure.III.35: les sorties réelles et estimées par le SSNN (1,4,4,4,4) pour estimer le système du 3ème cas

a, b, c, d : entrée échelon. e, f, g, h : entrée sinus

Le Tableau III.12 montre les erreurs quadratiques moyennes entre les sorties réelles et les sorties estimées par le SSNN dans le cas d'une entrée échelon et dans le cas d'une entrée sinusoïdale.

	EQMTy1	EQMTy2	EQMTy3	EQMTy4
Echelon	0.6×10^{-4}	8.9×10^{-4}	0.8×10^{-4}	0.1×10^{-4}
Sin	4.4×10^{-4}	3.5×10^{-4}	5.6×10^{-4}	1.2×10^{-4}

Tableau.III.12: les erreurs quadratiques moyennes entre les sorties réelles et les sorties estimées

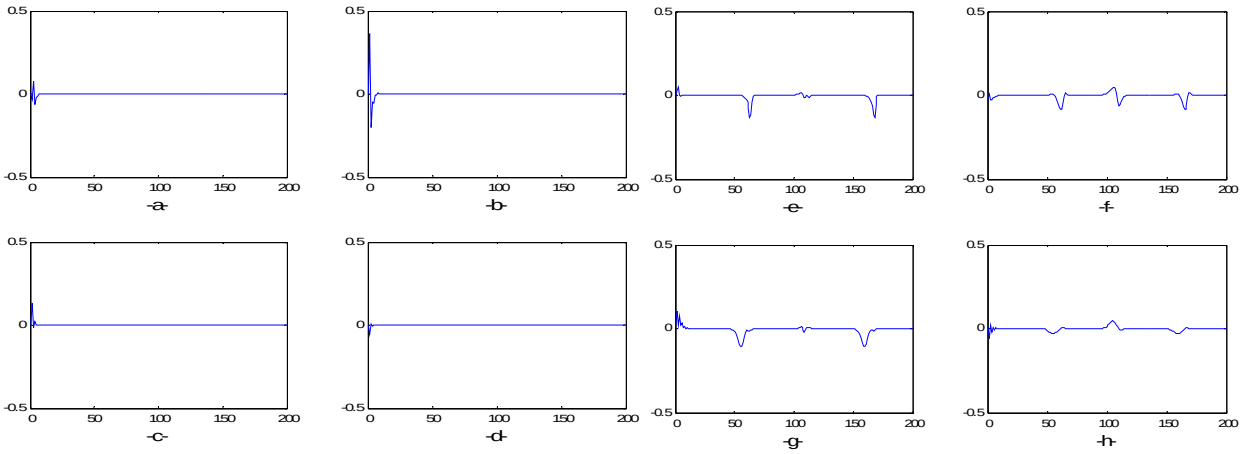

Figure.III.36: différence entre les sorties réelles et estimées obtenues avec le SSNN(1,4,4,4,4) pour le système du 2ème cas.

La figure.III.36 montrent la différence entre les sorties y_1, y_2, y_3 et y_4 *et* leurs estimés $\hat{y}_1, \hat{y}_2, \hat{y}_3$ et $\hat{y}_4$ par le SSNN. On voit que le SSNN a réussi à reproduire parfaitement le comportement du système en régime permanent (l'erreur est nulle), et l'erreur d'estimation a beaucoup diminuée en régime transitoire. En effet, les erreurs ne dépassent pas les valeurs maximales suivantes : 0.07, 0.3, 0.1, 0.05.

Les matrices des poids obtenues après l'apprentissage sont les suivantes :

$$W^i = \begin{bmatrix} -1.0190 \\ 0.0100 \\ 0.0145 \\ -0.6923 \end{bmatrix} \qquad W^r = \begin{bmatrix} -6.0077 & 4.6747 & -8.2025 & 3.9456 \\ 0.0001 & -0.0043 & 0.0085 & 0.0070 \\ 0.0148 & -0.0946 & -0.0019 & 0.0262 \\ 2.6050 & -1.4413 & 0.9095 & 0.5882 \end{bmatrix}$$

$$W^h = \begin{bmatrix} -0.2785 & 83.0071 & -57.1716 & 0.9309 \\ -0.6100 & 133.5113 & -43.1038 & 0.2914 \\ -0.0617 & 230.7391 & -20.7870 & 0.2914 \\ -0.0680 & 272.9594 & -9.5428 & 0.1870 \end{bmatrix} \quad B^h = \begin{bmatrix} -4.0224 \\ -0.0015 \\ 0.7382 \\ 17.5569 \end{bmatrix} \quad B^1 = \begin{bmatrix} -3.6017 \\ -37.8664 \\ -101.6331 \\ -129.9741 \end{bmatrix}$$

$$W^{h2} = \begin{bmatrix} -0.2013 & 1.0271 & -0.3423 & 0.3128 \\ 0.1728 & -0.9885 & 0.2599 & 0.5055 \\ -0.0440 & 0.0432 & 0.4564 & 0.5055 \\ 0.9772 & 0.0306 & 0.0228 & -0.0159 \end{bmatrix}$$

$$W^0 = \begin{bmatrix} 0.3624 & 0.4126 & 0.0067 & 1.0424 \\ -4.1450 & -5.1916 & -0.0816 & 0.0421 \\ -7.0252 & -7.2224 & 0.9310 & -0.1311 \\ 4.9684 & 5.1432 & 1.0653 & 0.1123 \end{bmatrix} \quad B^{h2} = \begin{bmatrix} -0.0115 \\ -0.0079 \\ 0.0072 \\ 0.0034 \end{bmatrix} \quad B^2 = \begin{bmatrix} 0.0006 \\ -0.0041 \\ 0.0052 \\ -0.0031 \end{bmatrix}$$

La représentation d'état qui peut être formée à partir des poids optimaux est la suivante :

$$\begin{cases} \hat{x}(k+1) = W^h . f1(W^r . \hat{x}(k) + W^i . \vec{u}(k) + B^h) + B^1 \\ \hat{y}(k) = W^0 . f_2(W^{h2} . \hat{x}(k) + B^{h2}) + B^2 \end{cases} \quad \text{..... (III-27)}$$

III-4-2- SSNN prédicteur à un pas

III-4-2-a- Système non linéaire de 4ème ordre

Soit le système d'ordre 4 dont la représentation d'état est la suivante :

$$\mathbf{x}(k+1)=\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix}=\begin{bmatrix} 0 & 1 & 0 & 0 \\ \dfrac{p\sin x_1(k)+p}{x_1(k)} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & p & 0 \end{bmatrix}\begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix}+\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}u(k) \quad \ldots\ldots \text{(III-18)}$$

$$\mathbf{y}(k)=\begin{bmatrix} x_1(k) & x_2(k) & \tanh(x_3(k)) & \tanh(x_4(k)) \end{bmatrix}^T$$

Où p est une constante égal à 0.85

L'objectif est :

1. de trouver des prédicteurs à un pas neuronaux à un système non linéaire dynamique d'ordre 4, c'est-à-dire trouver un modèle neuronal qui prédit la sortie du système qui vient juste après un pas d'échantillonnage.
2. d'évaluer la qualité de l'estimation déduite du SSNN.

L'architecture de prédicteur neuronal à un pas correspondant à ce système est montrée sur la figure.III.37. Il s'agit d'un SSNN (5,4,4,4,2) dont les neurones utilisent les fonctions d'activation :

f_1 : fonction d'activation pour la première couche cachée qui est une sigmoïde unipolaire

f_2 : fonction d'activation pour la deuxième couche cachée qui est une tangente hyperbolique

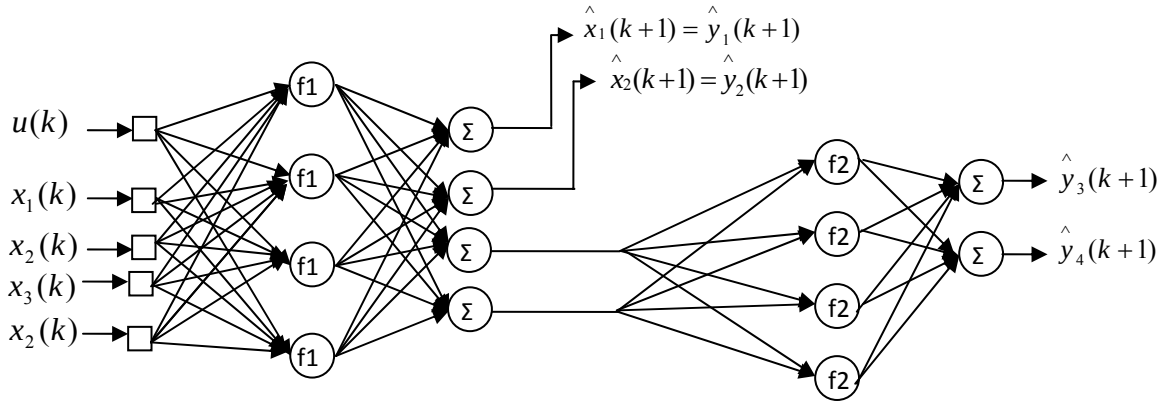

Figure. III.37: L'architecture de prédicteur neuronal (5,4,4,4,2)

Les détails concernant l'apprentissage sont les suivants :

- La séquence de commande utilisée pour l'apprentissage est un signal de commande $u(k)$ sinusoïdal (signalsin.m), la séquence comporte 1800 mesures (figure III.17).
- Les séquences de variables d'état et de sorties utilisées pour l'apprentissage sont générées à partir du modèle mathématique du système.
- La séquence de commande utilisée pour l'estimation de la performance du SSNN est le même signal utilisé précédemment (signal2).

La figure.III.38 montre les résultats du test en utilisant un apprentissage hors ligne avec l'algorithme du gradient et la figure. III.39 montre les résultats du test avec l'algorithme de Levenberg Marquardt.

Le tableau III.13 regroupe les erreurs quadratiques moyennes de test entre les états et sorties réels et les états et sorties estimés.

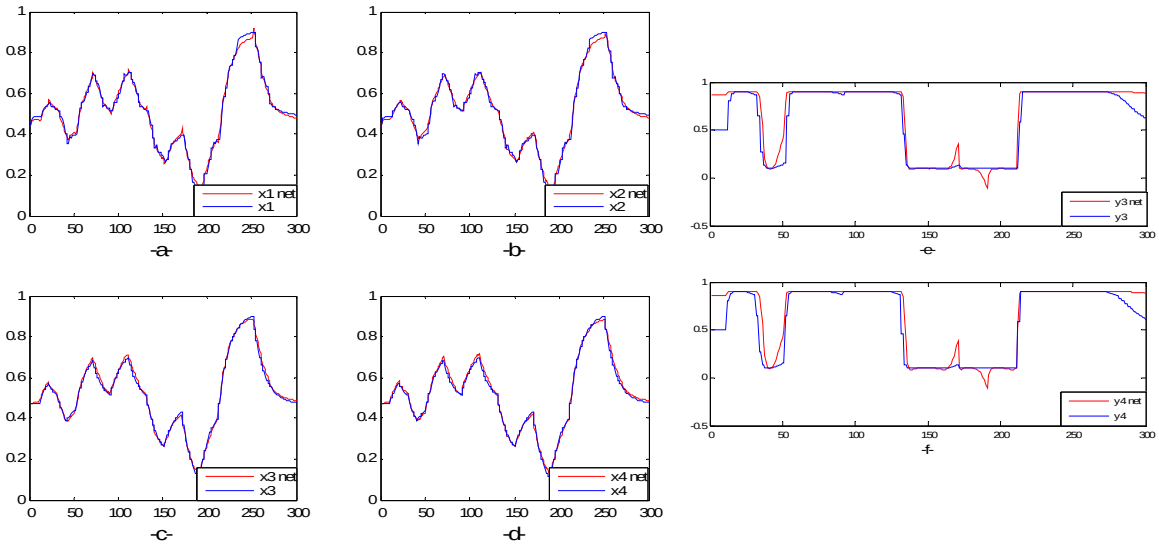

Figure. III.38: les états et les sorties prédites par le SSNN en utilisant la méthode du gradient

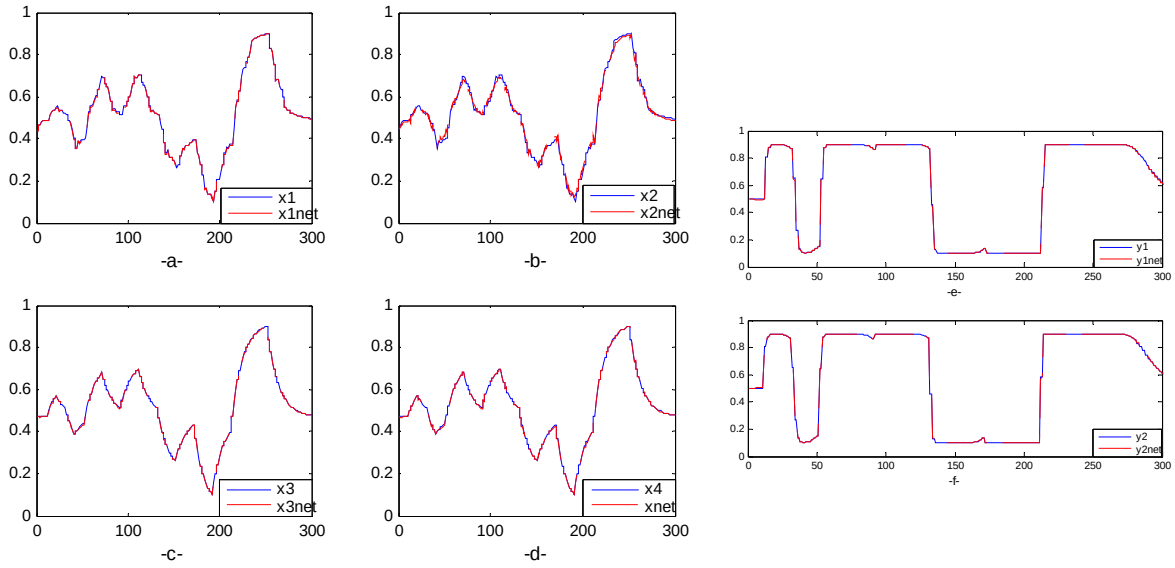

Figure. III.39: les états et les sorties prédites par le SSNN en utilisant la méthode de L-M

	EQMTx1 EQMTy1	**EQMTx2 EQMTy2**	**EQMTx3**	**EQMTx4**	**EQMTy3**	**EQMTy4**
Gradient	0.07×10^{-3}	0.32×10^{-3}	0.23×10^{-3}	0.10	0.30	0.17
L-M	6.10×10^{-6}	4.58×10^{-6}	2.83×10^{-6}	1.47×10^{-4}	3.68×10^{-7}	$1.30\times10^{--7}$

Tableau.III.13 : erreurs quadratiques moyennes entre les états réels et les états prédits par le SSNN (5,4,4,4,2)

D'après les résultats obtenus, on peut voir que l'algorithme de Levenberg-Marquardt est plus performant que l'algorithme du gradient pour l'identification des systèmes non linéaires dynamiques. Comme prédicteur à un pas, le SSNN est très précis. En effet, l'erreur quadratique moyenne de test est de l'ordre de 10^{-6}. Comme simulateur, les résultats obtenus sont moins bons mais ils sont satisfaisants car l'erreur quadratique moyenne de test est de l'ordre de 10^{-4}.

III-5- Comparaisons de l'identification d'un système linéaire avec un SSNN et un multicouche

III-5-1- Prédicteurs à un pas

III-5-1-a- Système linéaire de deuxième ordre

Prenons comme exemple un système linéaire du deuxième ordre. Ce processus dynamique simulé est décrit par les équations d'état suivantes :

$$\begin{cases} x1(k+1) = 0.72x1(k) + 0.5u(k) \\ x2(k+1) = 0.72x2(k) + 0.002u(k) \\ y(k+1) = x1(k+1) + x2(k+1) \end{cases} \quad \text{……..……} \quad \text{(III-19)}$$

L'objectif est de comparer les performances du SSNN et celle d'un réseau multicouche pour la prédiction à un pas d'échantillonnage de la sortie d'un système linéaire du deuxième ordre.

L'architecture du SSNN prédicteur à un pas correspondant au système (III-19) est la suivante :

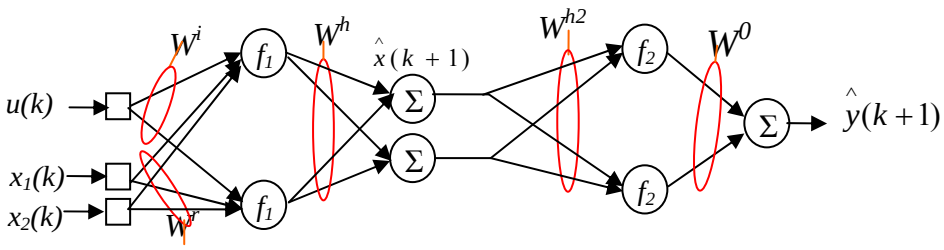

Figure III.40 : Architecture du SSNN (3,2,2,2,1)

f_1 et f_2 sont des fonctions linéaires

- Le modèle entrée/sortie correspondant à un processus d'ordre n=2 est :

$$y(k+1) = h(y(k), y(k-1), u(k)) \quad \text{……..……} \quad \text{(III-20)}$$

Le modèle neuronal (multicouche) correspondant est :

$$\hat{y}(k+1) = \Psi_{RN}(y(k), y(k-1), u(k), W) \quad \text{……..} \quad \text{(III-21)}$$

La Figure.III.41 montre un réseau de neurones entrée-sortie (multicouche) correspondant au système (III-19) et qui est d'ordre *n*=2.

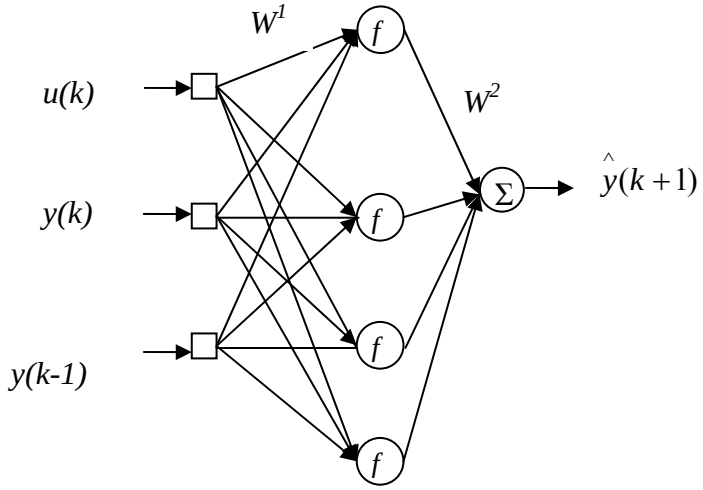

Figure III.41 : Réseau Multicouche non bouclé avec 3 entrées, 8 neurones dans la couche cachées et 1 sortie

Les séquences d'apprentissages et de test sont les mêmes séquences utilisées précédemment pour l'apprentissage des réseaux SSNN, la séquence d'apprentissage est montrée sur la figure III.2.a et la séquence de test est montrée sur la figure III.2.b.

La séquence de commande utilisée pour le test est une suite de créneaux d'amplitudes aléatoires et de durées aléatoires, la séquence comporte 300 pas d'échantillonnage.

La figure.III.42 montre les résultats du test obtenus avec un SSNN en utilisant un apprentissage hors ligne avec l'algorithme de Levenberg Marquardt. La figure. III.43 montre les résultats du test obtenus sur un réseau multicouche en utilisant un apprentissage hors ligne avec l'algorithme de Levenberg Marquardt.

Les erreurs quadratiques moyennes d'apprentissage et de test sont regroupées dans le tableau III.14.

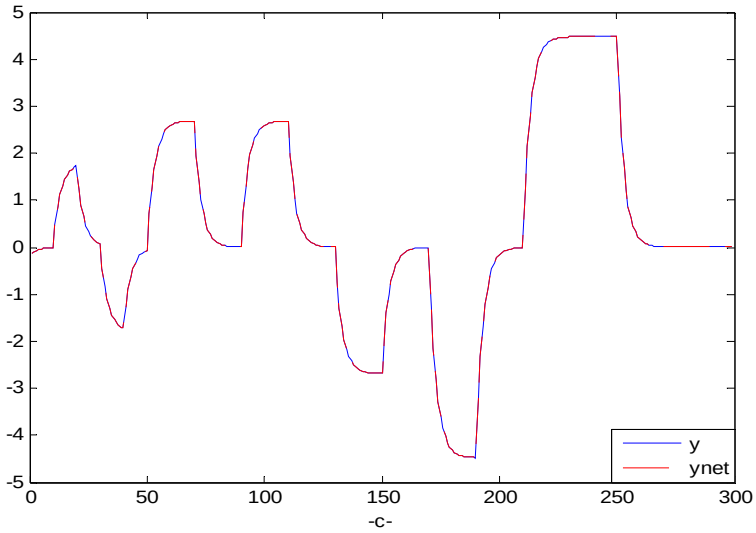

Figure. III.42 : la sortie réelle et prédite par un SSNN

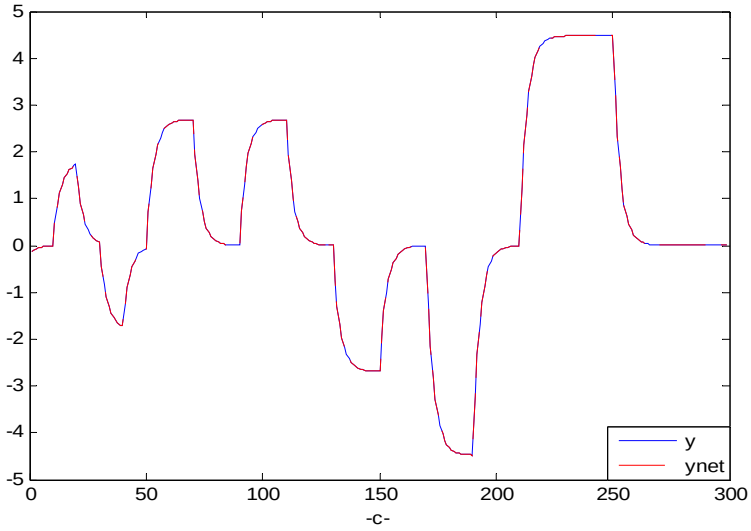

Figure. III.43 : la sortie réelle et prédite par un réseau multicouche

	Nb neurones	EQMAy	EQMTy
SSNN	2	4.07×10^{-31}	8.45×10^{-31}
multicouche	4	5.07×10^{-5}	5.21×10^{-5}

Tableau III.14 : les erreurs quadratiques moyennes d'apprentissage et de test entre les sorties réels et les sorties estimés par un SSNN et un multicouche

Les résultats montrent que les deux types de réseaux peuvent prédire la sortie du système. Le SSNN (3,2,2,2,1) prédit la sortie avec une précision parfaite (10^{-31}) avec seulement 2 neurones dans les deux couches cachées soit 16 poids neuronaux . Le multicouche peut prédire la sortie du système avec une erreur d'estimation de 10^{-5} qui est une erreur très petite avec 4 neurones dans la couche cachée soit 16 poids neuronaux.

III-5-2-b- Système non linéaire du deuxième ordre

Prenons comme exemple un système non linéaire de 2ème ordre dont les équations d'état sont les suivantes :

$$\begin{cases} x(k+1) = 1.145x1(k) + 0.549x2(k) + (0.222 + 2*0.181)u(k) \\ x2(k+1) = \dfrac{x1(k)}{1+0.01x2(k)^2} + \dfrac{0.181}{0.549}u(k) \\ y(k) = 4\tanh\left(\dfrac{x1(k)}{4}\right) \end{cases} \quad \text{..... (III-22)}$$

L'objectif est de comparer les performances de l'SSNN et celle d'un réseau multicouche pour la prédiction de la sortie d'un système non linéaire du deuxième ordre.

L'architecture neuronale d'un prédicteur à un pas est montrée sur la figure III.44. Ce réseau peut prédire la sortie et l'état à l'instant *k+1*, connaissant l'état à l'instant *k*.

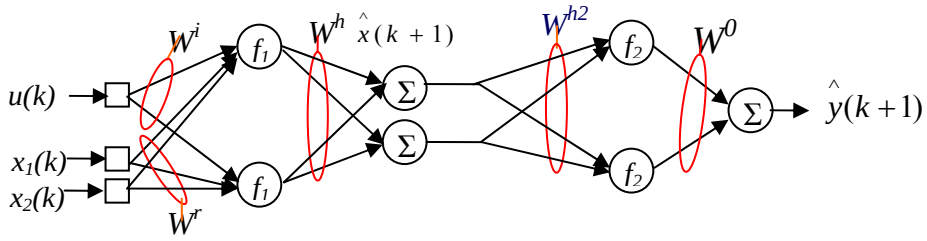

Figure. III.44 : SSNN prédicteur à un pas (3,2,2,2,1)

Après apprentissage en utilisant l'algorithme de Levenberg Marquardt, les erreurs d'apprentissage et de test pour un SSNN et un multicouche sont montrées dans le tableau.III.15 et les réponses de ce système à des échelons d'amplitudes aléatoires sont montrées sur la figure. III.45 en utilisant un SSNN avec seulement 2 neurones dans chaque couche cachée et sur la figure III.46 en utilisant un réseau multicouche avec 8 neurones dans la couche cachée.

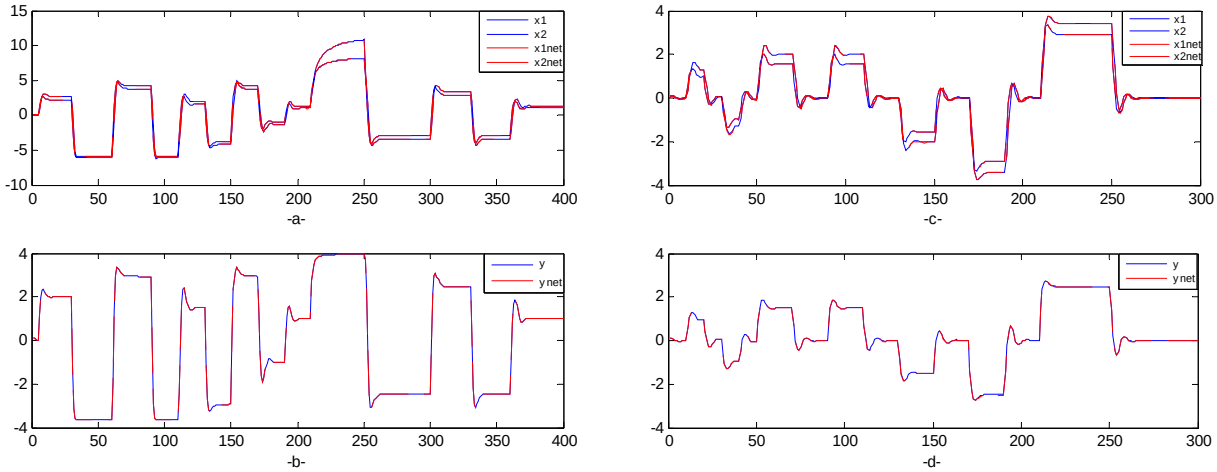

Figure. III.45 : résultats obtenus avec un SSNN

-a- et -b- : séquence d'apprentissage

-c- et -d- : séquence de test

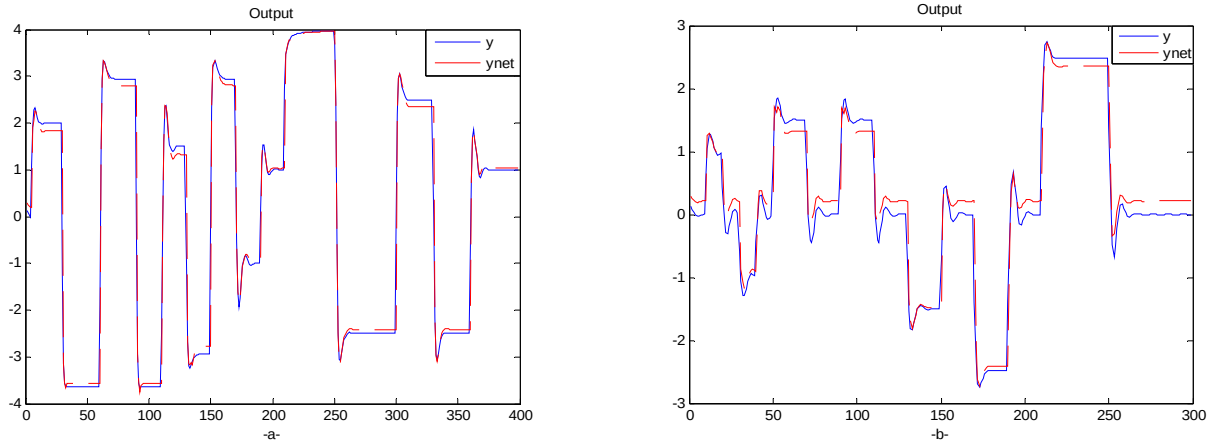

Figure. III.46 : résultats obtenus avec un réseau multicouche

-a- séquence d'apprentissage

-b- séquence de test

	Nb de neurones	EQMAy	EQMTy
SSNN	2	8.68×10^{-28}	1.39×10^{-5}
multicouche	8	0.1489	0.0694

Tableau III.15 : les erreurs quadratiques moyennes d'apprentissage et de test entre les sorties réelles et les sorties estimées par le SSNN et un multicouche

D'après les résultats montrés auparavant, on voit clairement que les meilleurs performances ont été obtenues avec un SSNN en utilisant l'algorithme de Levenberg-Marquardt après 200 itérations et uniquement 2 neurones dans les couches cachées, soit 16 poids neuronaux. En effet pour le système non linéaire EQMTy est de l'ordre de 10^{-5}. En revanche, pour le réseau multicouche des tests ont été fait en augmentant le nombre de neurones dans la couche cachée jusqu'à 8 neurones. Il se révèle peu économe en nombre de neurones (8 neurones dans la couche cachée soit 32 poids neuronaux à ajuster). Les résultats sont moins bons même si l'on augmente l'ordre du réseau à plus de 8 neurones dans la couche cachée et le nombre d'itérations à plus de 3000 itérations, EQMTy est de l'ordre de 0.069.

III-5-2- Simulateur

III-5-2-a- Système linéaire

L'architecture du simulateur neuronal correspondant au système linéaire (III-19) en utilisant un SSNN (1,2,2,2,1) est montrée sur la figure. III.47. Un simulateur qui utilise un réseau multicouche bouclé avec31 entrées, 8 neurones dans la couche cachée et 1 sortie, deux des entrées du réseau sont crées avec des lignes à retards pour utiliser la sortie aux instants précédant et est montrée sur la figure. III.48 :

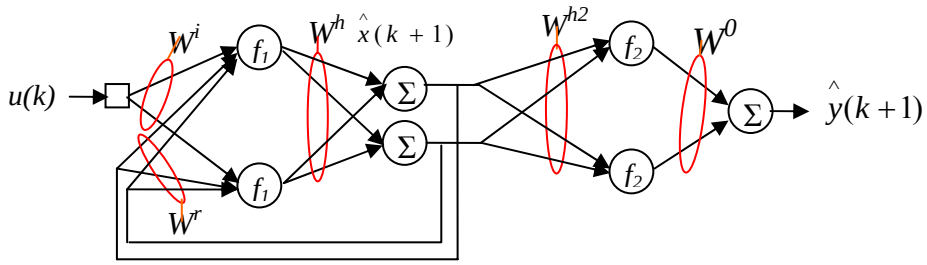

Figure. III.47 : Simulateur neuronal, SSNN(1,2,2,2,1) pour l'identification du système III-19

f_1 et f_2 sont des fonctions non linéaires (sigmoïde)

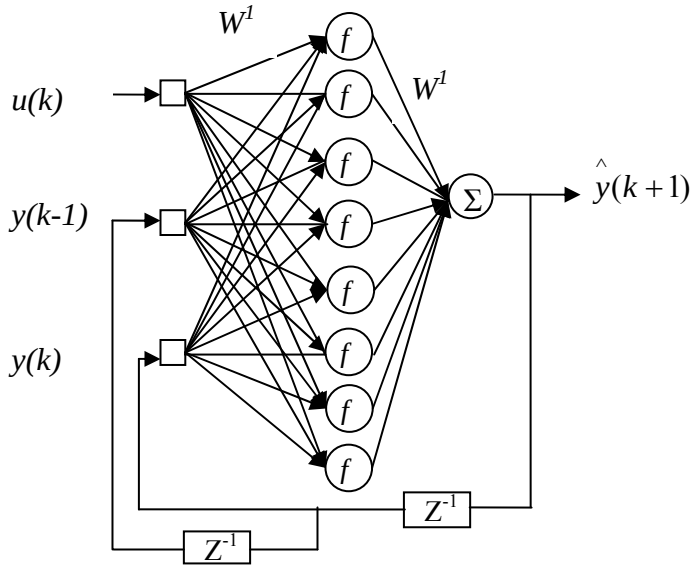

Figure. III.48 : Réseau multicouche bouclé pour l'identification du système III-19

Nous utilisons les mêmes séquences d'apprentissages et de test que nous avons utilisé pour l'apprentissage du prédicteur à un pas.

Après apprentissage en utilisant l'algorithme de Levenberg Marquardt, les réponses de ce système à des échelons d'amplitudes aléatoires sont montrées sur les figure III.49 en utilisant un SSNN (1,2,2,2,1) et sur la figure. III.50 en utilisant un réseau multicouche. Les erreurs d'apprentissage et de test pour un SSNN et un multicouche sont regroupées dans le tableau. III.16.

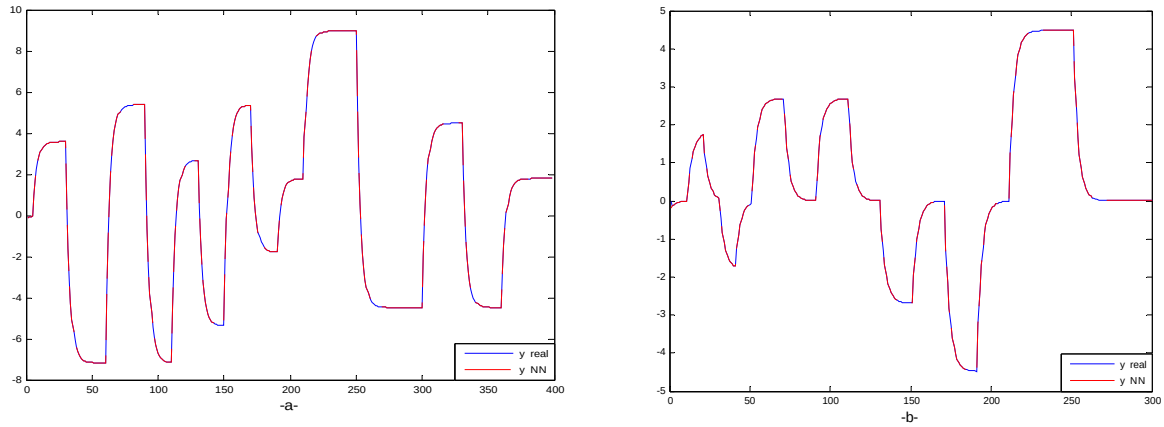

Figure. III.49 : Identification du système III-19 avec un SSNN

-a- séquence d'apprentissage

-b- séquence de test

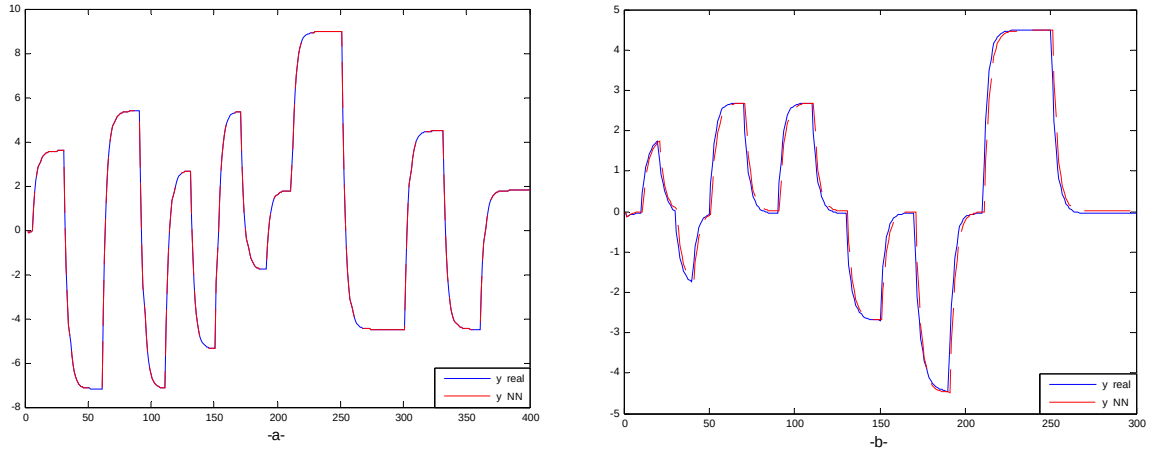

Figure. III.50 : Identification du système III-19 avec un réseau multicouche

-a- séquence d'apprentissage

-b- séquence de test

	Nb de neurones	EQMAy	EQMTy
SSNN	2	6.32×10^{-31}	1.33×10^{-4}
multicouche	4	5.31×10^{-5}	5×10^{-2}

Tableau III.16 : les erreurs quadratiques moyennes d'apprentissage et de test entre les sorties réelles et les sorties estimées

Le SSNN prédit la sortie sur un horizon infini avec une précision de (10^{-4}) avec seulement 2 neurones dans les deux couches cachées soit 16 poids neuronaux à ajuster tandis que le multicouche peut prédire la sortie du système avec une erreur d'estimation de 10^{-2} avec le même nombre de poids à ajuster (16 poids). Le SSNN donne donc des résultats meilleurs que le multicouche.

III-5-2-b- Système non linéaire

L'architecture du SSNN pour reproduire le comportement d'un système non linéaire du deuxième ordre (III-22) est exactement la même que pour le système linéaire (III-19), et l'architecture du réseau multicouche est montrée sur la figure III.51. Les séquences d'apprentissage et de test sont également les mêmes.

Le système non linéaire considéré est celui donné par l'équation III.22.

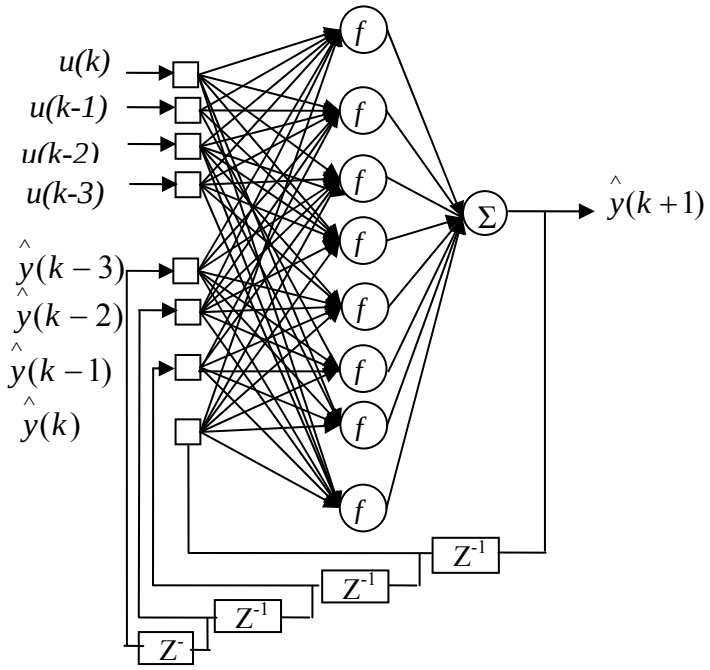

Figure. III.51 : l'architecture du réseau multicouche

Après apprentissage en utilisant l'algorithme de Levenberg Marquardt, les erreurs d'apprentissage et de test pour un SSNN (1,2,2,2,1) et un multicouche avec 3 entrées, 8 neurones dans la couche cachée et 1 sortie sont montrées dans le tableau III.17 et les réponses de ce système à des échelons d'amplitudes aléatoires sont montrées sur la figure III.52 en utilisant un SSNN et sur la figure III.53 en utilisant un réseau multicouche.

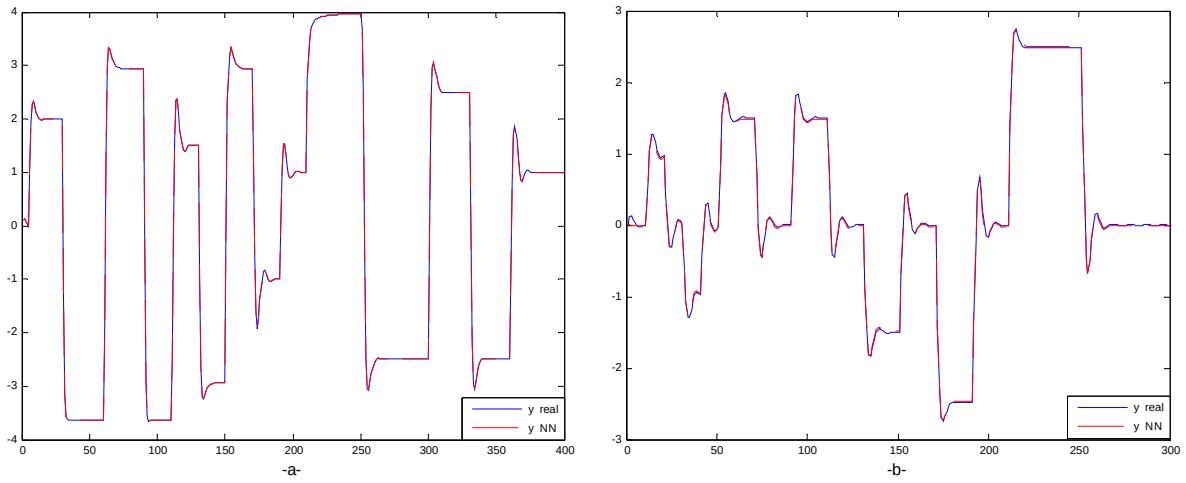

Figure. III.52 : Identification du système non linéaire III-22 avec un SSNN(1,2,2,2,1)

-a- séquence d'apprentissage

-b- séquence de test

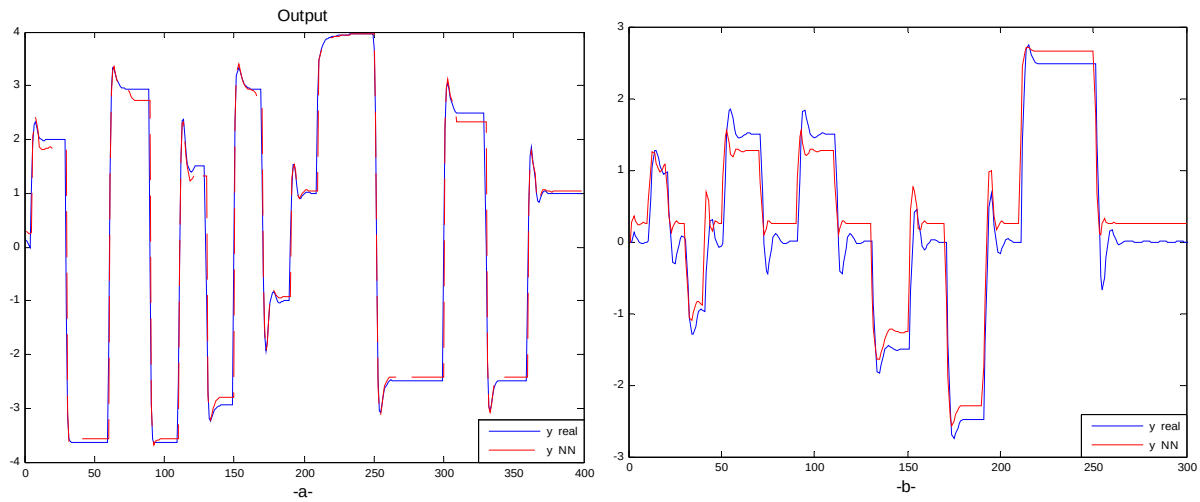

Figure. III.53 : Identification du système non linéaire III-22 avec un réseau multicouche bouclé (3 entrées, 8 neurones cachés et 1 sortie)

-a- séquence d'apprentissage

-b- séquence de test

	Nb de neurones	EQMAy	EQMTy
SSNN	2	4.43×10^{-31}	5.49×10^{-4}
multicouche	8	0.147	0.125

Tableau III.17 : les erreurs quadratiques moyennes d'apprentissage et de test entre les sorties réelles et les sorties estimées

Les résultats montrent que le simulateur neuronal SSNN (1,2,2,2,1) a réussi à reproduire le comportement du système non linéaire avec une précision de 10^{-4} en utilisant l'algorithme de Levenberg-Marquardt après 200 itérations et uniquement 2 neurones dans les couches cachées, soit16 poids neuronaux. Le réseau multicouche se révèle peu économe en nombre de neurones (8 neurones dans la couche cachée, soit 64 poids neuronaux), avec un nombre d'itérations à plus de 3000 itérations pour une erreur d'estimation de 0.125.

On peut dire que la modélisation neuronale basée sur un espace d'état (SSNN) est meilleure que la modélisation entrée/sortie (multicouche). Le SSNN estime la sortie avec précision contrairement au réseau multicouche. De plus, le SSNN est plus économe en nombre de neurones dans les couches cachées (nombre de paramètres à ajuster) et en nombre d'itérations.

III-6- Conclusion

Ce chapitre illustre la mise en œuvre de réseaux de neurones à espace d'état pour la modélisation des systèmes dynamiques linéaires et non linéaires en utilisant deux méthodes d'apprentissage différentes. La méthode du gradient stochastique, bien que relativement simple à implémenter, nécessite un certain savoir-faire pour une utilisation efficace. En effet, la convergence de l'algorithme n'est pas prouvée et de multiples variables sont à ajuster précisément en fonction du problème traité. Parmi ces variables à fixer, citons par exemple : le gain d'apprentissage (μ), la pente de la fonction sigmoïde, la sélection des exemples pour l'apprentissage et le test, le nombre de neurones dans la couche cachée, la configuration initiale des poids, le nombre d'itérations d'apprentissage.

Les résultats obtenus sur les différents exemples montrent clairement que la méthode de Levenberg Marquardt est plus performante que la méthode du gradient récursif pour l'identification des systèmes linéaires et non linéaires en utilisant un SSNN.

La comparaison du réseau de neurones à espace d'état avec un réseau multicouche (modélisation entrée/sortie) a mis en valeur l'efficacité des SSNN notamment pour les systèmes non linéaires. On peut dire que les SSNN sont des prédicteurs à un pas idéaux pour les systèmes linéaires et non linéaires.

Les résultats obtenus avec un SSNN pour simuler des systèmes linéaires sont excellents. Le SSNN est capable de reproduire fidèlement le comportement du système sur un horizon infini et avec n'importe quelle entrée de commande. Cependant pour simuler des systèmes non linéaires, le SSNN présente quelques limites dans le régime transitoire et plus précisément pour les systèmes fortement non linéaires.

Conclusion générale

Conclusion générale

Dans ce travail, une étude complète sur les réseaux de neurones à espace d'état (autrement dit les SSNN) a été présentée. Cette architecture des réseaux de neurones artificiels est basée sur la notion d'apprentissage, elle même issue de l'intelligence artificielle, mais également sur la représentation d'état utilisée par les automaticiens. L'intérêt majeur des SSNN réside dans le fait de pouvoir estimer et modéliser un processus dynamique. Au regard de l'intérêt suscité, il est intéressant et opportun d'étudier ce type de réseau afin d'évaluer ses capacités, ses avantages et ses limites, pour la modélisation des systèmes dynamiques. Cette étude est articulée en plusieurs parties.

Dans une première partie, le principe et l'architecture du SSNN sont introduits. Nous montrons également comment mettre en œuvre un SSNN pour des tâches de modélisation et d'identification des processus dynamiques simulés. Nous avons utilisé ce réseau pour modéliser différents systèmes dynamiques linéaires en utilisant un apprentissage dirigé avec deux méthodes. La première méthode est une méthode du premier ordre appelée "méthode du gradient stochastique". La seconde méthode est une méthode du deuxième ordre dite de "Levenberg-Marquardt". Les performances de ces deux méthodes d'apprentissage ont été évaluées. D'après les résultats obtenus, la deuxième méthode est meilleure que la première, notamment lorsqu'elle est appliquée à des systèmes linéaires d'ordre supérieur à 2. Par la suite, nous avons étudié les performances d'une identification par un SSNN d'un système linéaire du deuxième ordre en ajoutant à ce système un bruit (gaussien de variance 0.01) additif sur la sortie, puis en ajoutant des bruits de variances différentes. Le SSNN est insensible aux bruits de mesures.

Dans une seconde partie, nous nous sommes intéressés à la modélisation des systèmes non linéaires. Un SSNN a été utilisé en tant que prédicteur à un pas d'un système non linéaire d'ordre quatre où une partie des entrées du SSNN sont les sorties mesurées du système à modéliser.

Ainsi, le SSNN prédit la sortie du système à l'instant d'échantillonnage suivant. De très bons résultats ont été obtenus en utilisant l'algorithme de Levenberg-Marquardt. Par la suite, nous avons étudié le simulateur neuronal du même système non linéaire avec les deux méthodes d'apprentissage mise en œuvre. Le système considéré a été choisi pour sa particularité très intéressante qui consiste à changer son degré de non linéarité en ajustant un paramètre. Les résultats obtenus montrent les limites du SSNN dans l'estimation des sorties et des états dans les périodes transitoires avec la méthode du gradient notamment lorsque la non linéarité du système est grande. En régime permanent le SSNN reproduit le comportement du système avec précision. En revanche, les résultats sont meilleurs avec la méthode de Levenberg-Marquardt dans les deux périodes, la phase transitoire et le régime permanent.

Dans une dernière partie, nous avons effectué des comparaisons entre un SSNN et un réseau de neurones du type multicouche. Ces travaux ont été conduits en prenant deux systèmes, un système linéaire du deuxième ordre et un système non linéaire du second ordre. L'analyse des résultats montre que :

- les SSNN sont des prédicteurs à un pas et des simulateurs idéaux dès lors qu'ils sont appliqués à des systèmes dynamiques linéaires et non linéaires lorsque l'état est mesuré ;
- la méthode de Levenberg-Marquard est plus performante et plus robuste que celle du gradient récursif, elle convient mieux pour l'identification des systèmes dynamiques avec les SSNN.

Ce travail a permis d'envisager de nombreuses perspectives et des orientations futures dans le domaine de réseaux de neurones. On peut citer :

- l'utilisation d'un autre algorithme pour l'apprentissage des SSNN tels que les techniques de recherche aléatoire et méta heuristiques.
- l'étude de la mise en œuvre des fonctions d'ondelette (Borowa et al 2007), (Brdys et al 2007) au sein même de l'architecture d'un SSNN pour la modélisation statique et dynamique de processus.

Bibliographie

Références bibliographiques

Åkesson, B.M., Toivonen, H.T., 2006. "A neural network model predictive controller," Journal of Process Control, vol. 16, no. 9, pp. 937-946.

Borne, P., Benrejeb, M., Haggège, J., 2007. "Les réseaux de neurones – Présentation et Applications," Editions Technip, Paris.

Borowa, A., Brdys, M.A,. Mazur, K., 2007. "Modelling of Wastewater Treatment Plant for Monitoring and Control Purposes by State – Space Wavelet Networks," International Journal of Computers, Communications and Control, vol. 2, no. 2, pp. 121-131.

Bose, B.K., 2007. "Neural Network Applications in Power Electronics and Motor Drives- An Interoduction and Perspective," IEEE Transactions on Industrial Electronic, vol. 54,no 1,pp. 14-33.

Brdys, M.A., Borowa, A., Idzkowiak, P., Brdys, M., 2007. "Adaptive prediction of stock exchange indices by State Space Wavelet Networks," International Journal of Applied Mathematics and Computer Sciences, vol. 19, no. 2, pp. 337-348.

Cao, Y., Ding, R.T., Zhang, Q.J., 2006. "State-space dynamic neural network technique for high-speed IC applications: modeling and stability analysis,"IEEE Transactions ob Mcrowave Theory and Techniques, vol.54, no. 6, pp. 2398-2409.

Chen, L., Narendra, K.S., 2004. "Identification and control of a nonlinear discrete-time system based on its linearization: a unified framework," IEEE Transactions on Neural Networks, vol. 15, no. 3, pp. 663-673.

Crosnier, A., Abba, G., Jouvencel, B., Zapata, R., 2001. " Ingénierie de la commande des systèmes ", Ellipses Edition.

Deng, H., Xu, Z., Li, H.-X., 2009. "A novel neural internal model control for multi-input multi-output nonlinear discrete-time processes," Journal of Process Control, vol. 19, no. 8, pp. 1392-1400.

Dreyfus, G., Martinez, J.M., Samuelides, M., Gordon, M.B., Badran, F., Thiria, S., Hérault, L., 2002. "Réseaux de neurones : méthodologie et applications," Eyrolles, Paris.

Elman , J. L., 1988. "Finding Structure in Time," Center for Research in Language Technical Report 8801, University of California, San Diego.

Elman, J.L., 1990. "Finding Structure in Time," Cognitive Science, vol. 14, no. 2, pp. 179-211.

Gil, P., Henriques, J., Dourado, A., Duarte-Ramos,H., Dourado, A., 2001. "State-Space Neural Networks and the Unscentend Kalman Filter in On-Line Nonlinear System Identification," LASTED International Conference on Intelligent Systems and Control, November 19-22, USA.

Gil1, P., Henriques, J., Dourado., A., Duarte-Ramos, H., 2004. "Stability Analysis for a Class of Affine State-Space Neural Networks," 12th Mediteranean conference on Control and Automation, Kusadasi, Turkey, June 6-9.

Gil, P., Henriques, J., Dourado, A., Duarte-Ramos, H., 2005. "Order estimation in affine state space neural networks," IEEE Mid-Summer Workshop on Soft Computing in Industrial applications, Espoo, Finland, June 28-30.

Hagan, M.T., Demuth, H.B., Beale, M., 1995. "Neural Network Design," PWS Publishing Company.

Haykin, S., 1997. "Neural Networks: A Comprehensive Foundation," Macmillan, 1994.

Henriques, J.O., Dourado, A., Gil, P., 2001. "State Space Neural Networks in Nonlinear Adaptive System Identification and Control," in IFAC Workshop on Advanced Fuzzy/Neural Control, Valencia, Spain, Octobre 15-16.

Herault, J., Jutten, C., 1994. "Les réseaux de neurones et le traitement du signal," Editions Hermès, Paris.

Hertz, J., Krogh, A., Palmer, R.G., 1991. "Introduction to the Theory of Neural Computation," Addison-Wesley.

Hornik, K., Stinchcombe M., White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*, 359–366.

Hornik, K., Stinchcombe, M., White, H., Auer, P., 1994. "Degree of approximation results for Feedforward networks approximating unknown mappings and their derivatives, " Neural Computation, vol. 6, pp.1262-1275.

Hunt, K.J., Sbarbaro, D., Zbikowski, R., Gawthrop, P.J., 1992. "Neural networks for control—a survey, " Automatica, vol. 28, no. 6, pp. 1083-1112.

Jordan M. I., 1986. "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, " In Proceedings of the Eighth Annual Conference of the Cognitive Science Society, pp. 531-546,Amherst,MA,1986.

Kotta, U., Chowdhury, F.N., Nomm, S., 2006. "On realizability of neural networks-based input-output models in the classical state-space form," Automatica, vol. 42, no.7, pp. 1211-1216.

Lennox, B., Montague, G.A., Frith, A.M., Gent, C., Bevan, V., 2001. "Industrial application of neural networks — an investigation," Journal of Process Control, vol. 11, no.3, pp. 497-507.

Lucea, M., 2006. " Modélisation dynamique par réseaux de neurones et machines à vecteurs supports : contribution à la maîtrise des émissions polluantes de véhicules automobiles," Thèse de Doctorat , Université de Paris 6, Paris.

Mahul, A., 2005. "Apprentissage de la qualité de service dans les réseaux multiservices : application au routage optimal sous contraintes", Thèse de Doctorat à l'Université de Sciences Pour l'Ingénieur de Clermont Ferrand.

Mirikitani, D. T., Nikolaev, N., 2010. "Recursive Bayesian Recurrent Neural Networks for Time-Series Modeling," IEEE Transactions on Neural Networks, vol. 21, no. 2, pp. 262 -274.

Narendra, K.S., Parthasarathy, K., 1990. "Identification and Control of Dynamical Systems Using Neural Networks," IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 4-27.

Nørgaard, M., Ravn, O., Poulsen, N.K., Hansen, L.K., 2000. "Neural Networks for Modelling and Control of Dynamic Systems," Springer-Verlag, London.

Nørgaard, M., Poulsen, N.K., Ravn, O., 2000. "New Developments in State Estimation for Nonlinear Systems," Automatica, vol. 36, no 11, pp. 1627-1638.

Ossar, Y., 1998. "Réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus," Thèse de Doctorat , Université de Paris 6, Paris.

Ould Abdeslam, D., 1992. "Techniques neuromimétiques pour la commande dans les systèmes électriques : application au filtrage actif parallèle dans les réseaux électriques basse tension," Thèse de Doctorat, Université de Haute-Alsace, Mulhouse.

Pan, T.Y., Wang, R.Y., 2004. "State space neural networks for short term rainfall-runoff forecasting," Journal of Hydrology , 297 , pp. 34–50.

Patan, K., 1998."Approximation of state-space trajectories by locally recurrent globally feed-forward neural networks," Neural Networks, vol.21, no. 1, pp. 59-64.

Pearlmutter, B., 1995." Gradient Calculations for Dynamic Recurrent Neural Network," Asurvey, IEEE, trans, on neural networks, vol. 6, no. 5.

Personnaz, L., Rivals, I., 2003. "Réseaux de neurones formels pour la modélisation, la commande et la classification," Editions CNRS, Paris.

Rivals, I., 1995. "Modélisation et commande de processus par réseaux de neurones : application au pilotage d'un véhicule autonome," Thèse de Doctorat à l'Université Paris 6.

Saul, L.K., Jordan, M.I., 2000 "Attractor dynamics for feedforward neural networks," Neural Computation, vol. 12, pp. 1313-1335.

Slotine, J.J.E., Li, W., 1991. "Applied Nonlinear Control," Prentice-Hall, Englewood Cliffs, NJ.

Thiria, S., Lechevalier, L., Gascuel, O., Canu, S., 1997. "Statistiques et méthodes neuronales," Dunod, Paris.

Touzet, C., 1992. "Les réseaux de neurones artificiels : introduction au connexionnisme, 150 pages, Préface de Jeanny Hérault," EC2 éd., Paris.

Touzet, C., 1990. "Contribution à l'étude et au développement de modèles connectionnistes séquentiels de l'apprentissage,"Thèse de Doctorat, Université de Montpellier II.

Van Lint, H,. Hoogendoorn, S.P., van Zuylen, H.J., 2002. "State Space Neural Networks for Freeway Travel Time Prediction," in J.R. Dorronsoro (Ed.): ICANN 2002, LNCS 2415, Springer-Verlag Berlin Heidelberg, pp. 1043–1048.

Widrow, B., Lehr, M.A., 1990. "30 years of adaptive neural networks: Perceptron, madaline and Backpropagation," Proceeding of the IEEE, vol. 15, pp. 365–428.

Wilamowsky, B.M., 2009. "Neural Network Architectures and Learning Algorithmes"

Williams, R. J., Peng, J., 1990. "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories," Appears in neural computation, vol. 2, pp. 490–501.

Wira, P., 2002. "Réseaux neuromimétiques, modularité et statistique : estimation du mouvement pour l'asservissement visuel de robots," Thèse de Doctorat, Université de Haute-Alsace, Mulhouse.

Werbos, P. J., 1974. "Beyond Regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. Thesis, Harvard University, Cambridge, MA.

Werbos, P.J, 1990. "Backpropagation trough time: what it does and how do it," *Proc IEEE*, 78, 1550–1560.

Xu, A., 2002. "Observateurs adaptatifs non-linéaires et diagnostic de pannes," Thèse de Doctorat, Université de Rennes 1, Rennes.

Zamarreno, J.M., Vega, P., 1998. "State space neural network. Properties and application," Neural Networks, vol. 11, no. 6, pp. 1099-1112.

Zamarreno, J.M., Vega, P., 1998. "Identification and predictive control of a melter unit used in the sugar industry," Artificial Intelligence in Engineering, vol. 11, no. 4, pp. 365-373.

Zamarreno, J.M., Vega, P., 1999. "Neural predictive control. Application to a highly non-linear system," Engineering Applications of Artificial Intelligence, vol. 12, no. 2, pp. 149-158.

Zamarreno, J.M., Vega, P., Garcia, L.D., Francisco, M., 2000. "State-space neural network for modelling, prediction and control," Control Engineering Practice, vol. 8, no. 9, pp. 1063-1075.