République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITER MOULOUD MAMMERI DE TIZI-OUZOU

FACULTÉ DU GÉNIE ÉLECTRIQUE ET D'INFORMATIQUE
DEPARTEMENT D'AUTOMATIQUE

Mémoire De Fin d'Etudes de MASTER ACADÉMIQUE

Domaine : Sciences et Technologies
Filière : Automatique
Spécialité : **Automatique et Informatique Industrielle**

Présenté par :

Hichem AMOURA
Malik ACHAB

Thème :

Réseaux de neurones à espace d'état pour l'identification des systèmes dynamiques

Mémoire soutenu publiquement le 26/09/2024 devant le jury composé de :

Mme Ouerdia CHILALI
MCB , UMMTO , Président

Mme Karima AMOURA
MCB , UMMTO , Encadrent

Mme Ghania IDIRI
MCB , UMMTO , Examinateur

Mme Lamia SERSOUR
MCB , UMMTO , Examinateur

Remerciements

Nous souhaitons exprimer notre profonde gratitude à nos familles respectives pour leur soutien indéfectible, leur patience et leurs encouragements constants. Leur compréhension face aux difficultés rencontrées et leur présence à nos côtés ont été des piliers essentiels pour mener à bien ce mémoire.

Nous adressons également nos remerciements les plus sincères à Mme AMOURA Karima, notre encadrante, pour son accompagnement attentif. Ses conseils éclairés, sa disponibilité infaillible et son engagement ont non seulement enrichi notre travail, mais aussi affiné nos compétences. Son expertise et sa rigueur ont été des éléments clés dans l'achèvement de ce projet.

Nous tenons également à remercier chaleureusement les membres du jury pour leur temps, leur expertise et leurs conseils précieux qui ont contribué à la finalisation de ce travail. À vous tous, nous adressons nos remerciements les plus sincères..

Résumé

Ce mémoire explore les réseaux de neurones, en introduisant d'abord leurs concepts fondamentaux et leur importance croissante dans le domaine de l'intelligence artificielle. Il présente les neurones biologiques comme une source d'inspiration pour la conception des neurones artificiels et définit les réseaux de neurones, en distinguant les architectures à propagation avant et récurrentes. Le principe de fonctionnement des réseaux de neurones et leurs mécanismes d'apprentissage, qu'ils soient supervisés, non supervisés ou semi-dirigés, sont ensuite examinés en profondeur. La deuxième partie du document s'intéresse aux réseaux de neurones à espace d'état (SSNN), en abordant leur représentation d'état, leur commandabilité et leur observabilité, ainsi que les avantages et inconvénients spécifiques à cette approche. Les méthodes d'apprentissage appliquées aux SSNN sont discutées, en mettant l'accent sur leur capacité à modéliser des systèmes dynamiques complexes. Enfin, la partie applicative du mémoire se concentre sur l'identification de systèmes linéaires et non linéaires, démontrant la pertinence et l'efficacité des SSNN dans la modélisation et le contrôle des systèmes réels.

Mots clés : Réseaux de neurones, neurone biologique, neurone artificiel, réseaux à propagation avant, réseaux récurrents, apprentissage supervisé, apprentissage non supervisé, réseaux de neurones à espace d'état (SSNN), commandabilité, observabilité, systèmes dynamiques, identification des systèmes, modélisation, systèmes linéaires, systèmes non linéaires.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

L'histoire des réseaux de neurones artificiels commence avec les travaux novateurs de Warren McCulloch et Walter Pitts, qui ont introduit le concept du "neurone formel" dans les années 1940 (McCulloch et Pitts, 1943). Ce modèle simplifié de neurone biologique a jeté les bases des premières recherches dans ce domaine. Dans les décennies suivantes, des chercheurs ont exploré les capacités théoriques de ces neurones formels pour réaliser des fonctions logiques, arithmétiques et symboliques. Ensuite, Frank Rosenblatt a développé le Perceptron, un modèle de neurone artificiel avec des poids ajustables, dans le but d'apprendre à classer des entrées en deux catégories distinctes (Rosenblatt, 1958). Bien que cette avancée ait suscité un grand intérêt initial, les limites du Perceptron en termes de résolution de problèmes complexes ont conduit à un déclin temporaire de l'intérêt pour les réseaux de neurones. Cependant, dans les décennies suivantes, en raison de l'émergence de nouveaux algorithmes et de l'amélioration des capacités de calcul, les réseaux de neurones ont connu un regain d'intérêt (Bengio, Goodfellow et Courville, 2016). Des techniques d'apprentissage plus sophistiquées, telles que la rétropropagation du Gradient, ont permis aux réseaux de neurones de résoudre des problèmes plus complexes (Rumelhart, Hinton et Williams, 1986).

Aujourd'hui, les réseaux de neurones artificiels sont largement utilisés dans divers domaines de l'intelligence artificielle, jouant un rôle essentiel dans des applications telles que les moteurs de recherche, les recommandations de produits, les assistants virtuels, et bien plus encore (LeCun, Bengio et Hinton, 2015). Ils sont considérés comme des outils fondamentaux pour résoudre une variété de problèmes complexes dans le domaine de l'intelligence artificielle.

Notre étude se focalise sur une catégorie spécifique de réseaux de neurones dynamiques appelés "réseaux de neurones à espace d'état" ou SSNN (State Space Neural Networks en an-

glais). Les SSNN font partie des outils de l'intelligence artificielle et trouvent des applications directes dans le domaine de l'Automatique. Ils se distinguent par des caractéristiques uniques absentes dans d'autres types de réseaux de neurones, notamment leur capacité à se rapprocher de l'Automatique classique grâce à leur architecture qui s'aligne sur la représentation d'état. Ainsi, les poids dans ces réseaux correspondent aux matrices de cette représentation. D'autres particularités intrigantes nous incitent à explorer et à évaluer les performances ainsi que les limites de cette approche (Bengio, Goodfellow et Courville, 2016).

Notre objectif est d'évaluer le potentiel et les performances optimales des SSNN dans diverses tâches telles que la modélisation, l'identification et la prédiction appliquées à des systèmes dynamiques. Les méthodes d'apprentissage appropriées à cette étude sont la méthode du gradient et la méthode de Levenberg-Marquardt.

Le chapitre I de notre étude propose une présentation des réseaux de neurones artificiels, commençant par une évocation succincte de leur évolution historique. Nous abordons ensuite des définitions des réseaux de neurones biologiques et mathématiques, ainsi que leurs principes de fonctionnement fondamentaux. Les principales architectures et techniques d'apprentissage sont également exposées.

Le chapitre II introduit le concept des SSNN, également connus sous le nom de réseaux de neurones à espace d'état. Ce type de réseau, de nature récurrente, présente une architecture spécifique adaptée à la modélisation et à la commande numérique des processus. Nous discutons également des notions de base de la représentation d'état des systèmes dans ce chapitre.

Le chapitre III présente les résultats obtenus en utilisant les SSNN pour modéliser divers systèmes dynamiques. Afin d'évaluer l'efficacité des SSNN, nous l'avons testé sur plusieurs types de systèmes dynamiques, notamment un système linéaire monovariable de premier ordre, un système linéaire multivariable de premier ordre, ainsi que des systèmes non linéaires de second et quatrième ordre. Pour ce faire, nous avons utilisé deux méthodes d'apprentissage : la méthode de Levenberg-Marquardt et la méthode du Gradient stochastique. Les performances de ces approches ont été analysées dans le cadre de l'identification de systèmes linéaires et non linéaires, ainsi que pour la prédiction de leurs comportements dynamiques.

Chapitre I

Généralités sur les réseaux de neurones

I.1 Introduction

L'intelligence artificielle (IA) est un domaine en plein essor qui vise à créer des machines capables d'imiter les capacités cognitives humaines (Russell et Norvig, 2020). Parmi les différentes approches de l'IA, les réseaux de neurones artificiels (RNA) constituent un domaine particulièrement prometteur. Ces réseaux s'inspirent du fonctionnement du cerveau humain, qui est composé de milliards de neurones inter-connectés (Haykin, 2009). Chaque neurone biologique est une unité de traitement élémentaire capable de recevoir des signaux, de les traiter et de les transmettre à d'autres neurones. Les RNA sont des outils puissants pour la résolution de nombreux problèmes d'apprentissage automatique, de traitement du signal et de reconnaissance de formes (Géron, 2017). Ils ont permis des avancées majeures dans des domaines tels que la vision par ordinateur (LeCun, Bengio et Hinton, 2015), la traduction automatique, le traitement du langage naturel et la robotique.

Ce chapitre propose une exploration des bases des réseaux de neurones artificiels, comprenant les définitions de base, et leurs fonctionnement et les différentes techniques d'apprentissage.

I.2 Définition du neurone biologique

Un neurone biologique, également connu sous le nom de cellule nerveuse, est l'unité fondamentale du système nerveux et du cerveau. Il est spécialisé dans le traitement, la réception et la transmission des informations électrochimiques vers d'autres cellules nerveuses du cerveau central ou périphérique. Chaque neurone est composé d'un corps cellulaire **(Soma)** comportant un noyau, de nombreuses ramifications dendritiques **(Dendrites)** pour recevoir les informations, et d'un axone **(Axon)** pour diffuser les informations (voir Figure (I.1)). La transmission des informations entre les neurones se fait via des **Synapses**, qui sont des jonctions spécialisées permettant le transfert des signaux électrochimiques. Les synapses peuvent être chimiques ou électriques, facilitant la communication entre les neurones. Les neurones sont fortement inter connectés, formant ainsi des réseaux de neurones complexes qui activent les fonctions du système nerveux. Ils sont impliqués dans des processus tels que l'apprentissage, la mémoire, la compréhension et la concentration (Kandel, Schwartz, Jessell, Siegelbaum et Hudspeth, 2012).

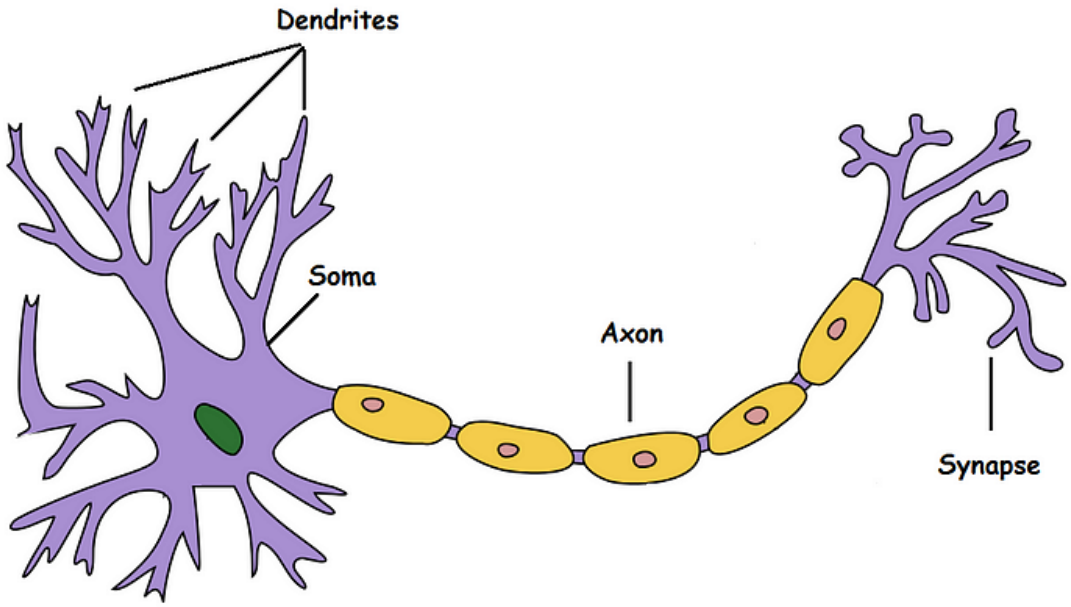

FIGURE I.1 – Neurone biologique

I.3 Définition du neurone artificiel

La définition d'un neurone artificiel est celle d'une unité de base ou d'une unité de calcul élémentaire d'un réseau de neurones artificiels. Il reçoit généralement plusieurs valeurs d'entrée et calcule une valeur de sortie en appliquant une fonction d'activation à une somme pondérée des valeurs d'entrée (voir Figure (I.2)). Inspiré du fonctionnement d'un neurone biologique, le neurone artificiel est conçu comme un automate doté d'une fonction de transfert qui transforme ses entrées en sortie selon des règles précises (Haykin, 2009). Ces neurones sont associés en réseaux dont la topologie des connexions est variable, tels que les réseaux proactifs, récurrents, etc. De plus, l'efficacité de la transmission des signaux d'un neurone à l'autre peut varier, ce qui est déterminé par les poids synaptiques, modulables par des règles d'apprentissage, imitant ainsi la plasticité synaptique des réseaux biologiques. Les réseaux de neurones artificiels sont utilisés dans le domaine de l'intelligence artificielle pour résoudre des problèmes complexes tels que la vision par ordinateur, le traitement du langage naturel, et bien d'autres applications (Haykin, 2009).

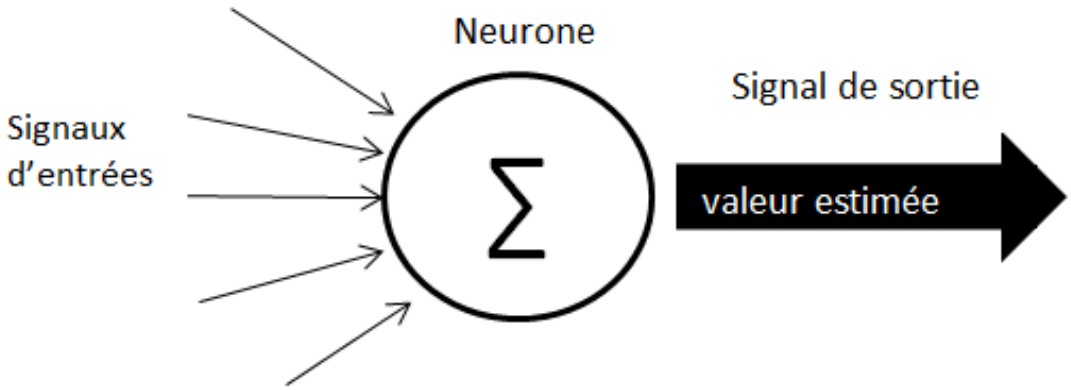

FIGURE I.2 – Neurone artificiel

I.4 Définition du réseau de neurone

Les réseaux de neurones artificiels, également appelés réseaux neuronaux, sont des algorithmes d'apprentissage automatique inspirés du fonctionnement des neurones biologiques (voir Figure (I.4)). Ils sont composés de nombreuses unités inter connectées appelées neurones artificiels, qui travaillent ensemble pour résoudre des problèmes spécifiques. Ces réseaux sont organisés en couches, comprenant une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque neurone de la couche d'entrée est connecté à chaque neurone de la première couche cachée, et ainsi de suite entre les couches cachées et la couche de sortie (voir Figure (I.3)). Les connexions entre les neurones ont des poids numériques qui sont ajustés au cours de l'entraînement du réseau. Les réseaux de neurones sont largement exploités dans une multitude de domaines, incluant la reconnaissance faciale, le traitement du langage naturel, la synthèse et l'analyse d'images, ainsi que dans le secteur financier (Haykin, 2009).

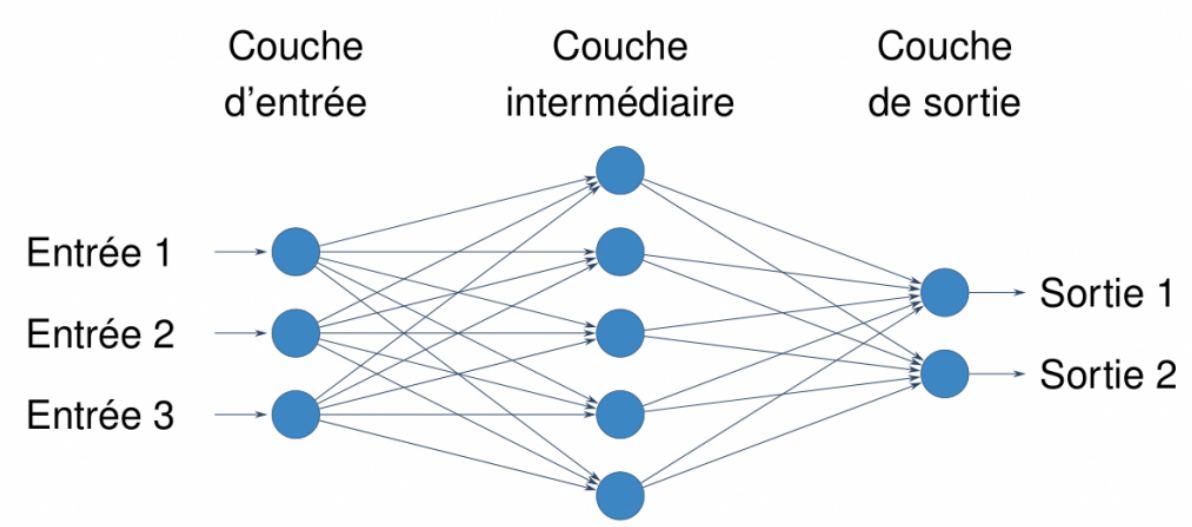

FIGURE I.3 – Réseau de neurones artificiels

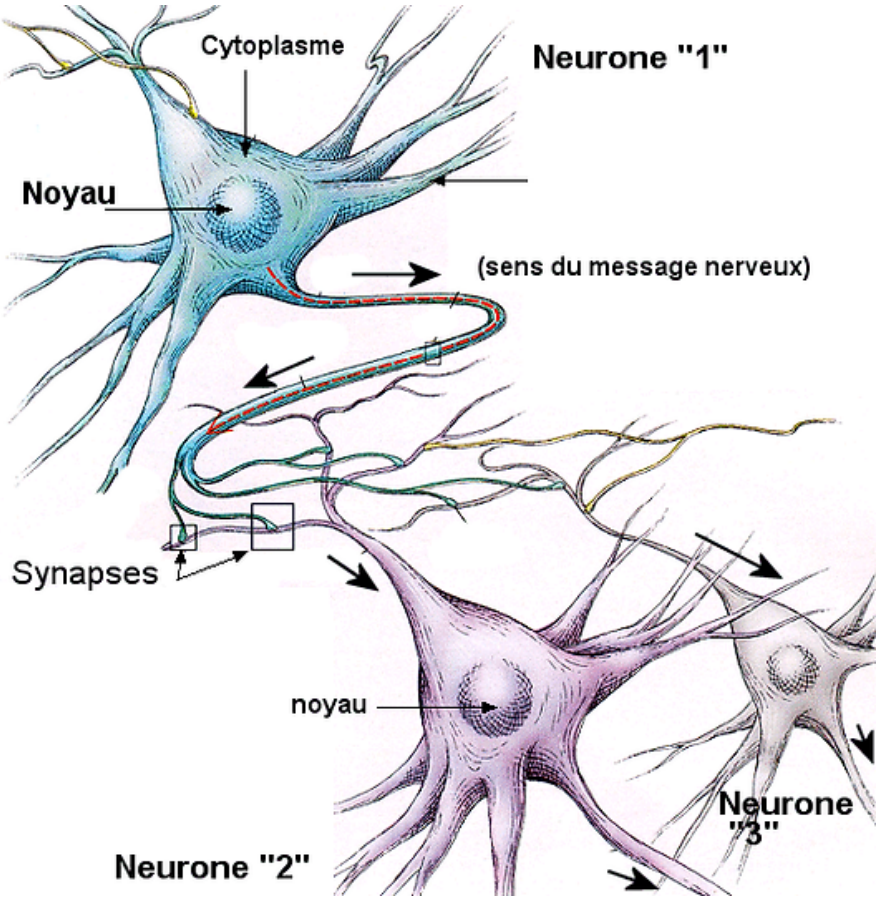

FIGURE I.4 – Réseau de neurones biologiques

I.5 Principe de fonctionnement du neurone artificiel

Un neurone artificiel commence par effectuer une combinaison linéaire des entrées qu'il reçoit. Chaque entrée est pondérée par un poids correspondant, ces produits sont ensuite additionnés. À cette somme est ajoutée une valeur appelée biais, qui permet de décaler la fonction d'activation pour améliorer la capacité du modèle à représenter les données. Le biais est très important car il permet au neurone de s'activer (ou de se déclencher) même en l'absence de toute entrée, ou de modifier le seuil d'activation (Haykin, 2009) (voir Figure (I.5)).

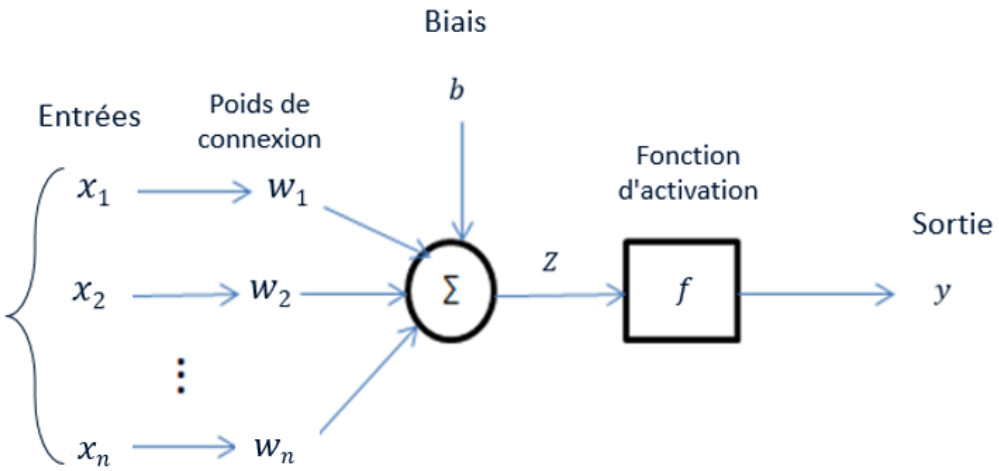

FIGURE I.5 – Neurone mathématique

La combinaison linéaire peut être représentée par la formule suivante :

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{I.1}$$

Où :

z représente la combinaison linéaire pondérée des entrées avec le biais.

w_i sont les poids synaptiques associés à chaque entrée x_i .

x_i sont les valeurs d'entrée.

n est le nombre d'entrées du neurone.

b est le terme de biais.

La sortie du neurone sera donc :

$$y = f(z) = f(\sum_{i=1}^{n} w_i x_i + b) \tag{I.2}$$

Tel que f est la fonction d'activation.

— **La fonction d'activation**

La fonction d'activation dans un neurone artificiel est une fonction mathématique appliquée à un signal en entrée pour produire un signal de sortie. Elle est utilisée pour déterminer si le neurone doit être activé ou non. Les fonctions d'activation peuvent varier, mais elles sont généralement non-linéaires. Elles peuvent être aussi simples qu'une fonction de seuil qui renvoie 0 si l'entrée est inférieure à un certain seuil et 1 si elle est supérieure, ou aussi

complexes qu'une fonction sigmoïde ou tangente hyperbolique qui renvoie une valeur entre 0 et 1 ou -1 et 1 respectivement.

Il existe différentes formes de fonctions d'activation (voir Table (I.1)), les plus courantes sont les sigmoïdes unipolaire et bipolaire ainsi que la tangente hyperbolique.

Sigmoïde Unipolaire

La sigmoïde unipolaire est souvent utilisée dans les réseaux de neurones pour modéliser des comportements binaires, comme la classification ou la prise de décision binaire. Elle est efficace pour des sorties probabilistes où les valeurs doivent être comprises entre 0 et 1. La fonction est la suivante : $f_1(x) = \frac{1}{1+\exp(-x)}$.

Sigmoïde Bipolaire

Contrairement à la sigmoïde unipolaire, cette fonction varie entre -1 et 1, offrant une plage de sortie symétrique autour de zéro. Cette symétrie peut être avantageuse dans certaines applications de modélisation, car elle permet de mieux équilibrer les signaux positifs et négatifs. La fonction est la suivante : $f_2(x) = \frac{1-\exp(-x)}{1+\exp(-x)}$.

Tangente Hyperbolique

La tangente hyperbolique partage des propriétés similaires avec la fonction sigmoïde bipolaire, mais elle est souvent préférée car elle centre les données de sortie autour de zéro, ce qui peut accélérer la convergence du réseau pendant l'apprentissage. Elle est également utilisée pour normaliser les valeurs d'entrée. Bien que 'tanh' puisse souffrir du problème du gradient qui disparaît, elle évite le problème de saturation associé à la sigmoïde grâce à sa symétrie autour de zéro. La fonction 'tanh' peut remplacer la sigmoïde dans la dernière couche d'un modèle de classification binaire. La fonction est la suivante : $f_3(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$.

La Table (I.1) présente les différentes fonctions d'activation utilisées dans les réseaux de neurones, détaillant leurs expressions mathématiques ainsi que leurs correspondances sur MATLAB.

Nom de la fonction	Relation entrée/sortie	Icône	Nom sur MATLAB
Seuil	$y = 0$ si $s < 0$ $y = 1$ si $s \geq 0$		hardlim
Seuil symétrique	$y = -1$ si $s < 0$ $y = 1$ si $s \geq 0$		hardlims
Linéaire	$y = s$		purelin
Linéaire saturée	$y = 0$ si $s \leq 0$ $y = s$ si $0 \leq s \leq 1$ $y = 1$ si $s \geq 1$		satlin
Linéaire positive	$y = 0$ si $s \leq 0$ $y = s$ si $s \geq 0$		poslin
Sigmoïde	$y = \frac{1}{1+\exp(-s)}$		logsig

TABLE I.1 – Fonctions d'activation

I.6 Classification et types des réseaux de neurones

La classification des réseaux de neurones artificiels est un domaine vaste qui englobe diverses architectures et méthodes d'apprentissage. Ces réseaux sont utilisés pour traiter et classifier des données complexes, notamment des images, des séries temporelles ou des données textuelles. elle peut être divisée en deux catégories principales : les réseaux de neurones bouclés (ou récurrents) et les réseaux de neurones non bouclés (ou à propagation avant).

I.6.1 Réseaux de Neurones Non Bouclés (ou à Propagation Avant)

Les réseaux de neurones à propagation avant, également connus sous le nom de réseaux de neurones non bouclés, sont des réseaux de neurones artificiels acycliques, se distinguant ainsi des réseaux de neurones récurrents (Haykin, 2009). Le plus connu est le Perceptron multicouche, qui est une extension du premier réseau de neurones artificiel.

Dans ce type de réseau, l'information ne se déplace que dans une seule direction, vers

l'avant, à partir des nœuds d'entrée, en passant par les couches cachées (le cas échéant) et vers les nœuds de sortie. Il n'y a pas de cycles ou de boucles dans le réseau. Lorsque le réseau de neurones à propagation avant comporte plusieurs couches cachées, on parle habituellement d'un Perceptron multicouche (Rumelhart, Hinton et Williams, 1986).

Ce type de réseaux est utilisé pour effectuer des taches d'approximation de fonction non linéaire, de la classification ou de la modélisation de processus statiques non linéaires .

Un réseau de neurones non bouclé réalise une fonction non linéaire de ses entrées et peut être représenté mathématiquement par des équations qui décrivent le processus de propagation des données à travers les différentes couches du réseau (Nielsen, 2015).

Voici une représentation mathématique d'un réseau de neurones non bouclé réalisant une fonction non linéaire de ses entrées paramétrée par les coefficients W du réseau :

Soit $O(k)$ le vecteur de sortie à l'instant k, $I(k)$ est le vecteur des entrées, et ψ la fonction non linéaire réalisée par les neurones du réseau. Alors, la sortie peut être représentée par :

$$O(k) = \psi(I(k); W) \tag{I.3}$$

La Figure (I.6) présente la forme canonique d'un réseau de neurones non bouclé.

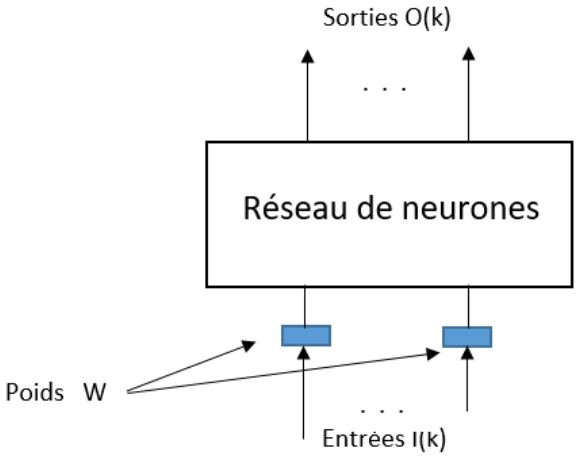

FIGURE I.6 – Forme canonique d'un réseau de neurones non bouclé

— **Réseaux de neurones monocouche**

La structure d'un réseau monocouche consiste en des neurones d'entrée entièrement connectés à des neurones de sortie. Chaque connexion entre les neurones est associée à un poids qui peut être modifié (Haykin, 2009).

La Figure (I.7) illustre le réseau de neurones monocouche.

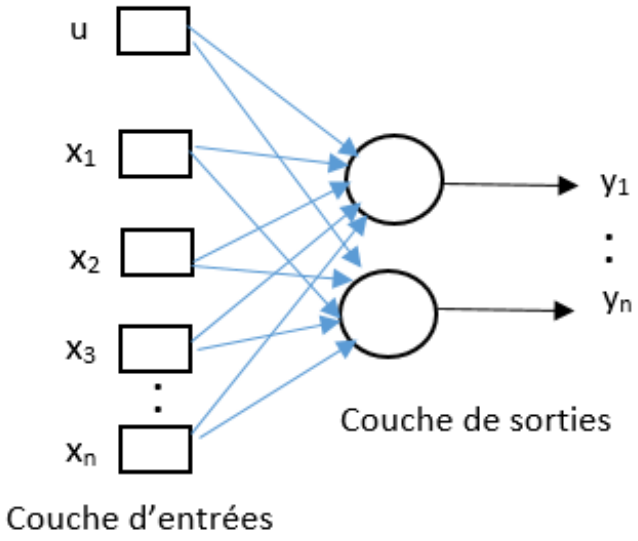

FIGURE I.7 – Réseau de neurones monocouche

— **Réseaux de neurones multicouche**

Il s'agit du réseau de neurones statique le plus couramment utilisé, caractérisé par des neurones organisés en couches successives, comme le montre la Figure (I.8). Les neurones de la première couche reçoivent le vecteur d'entrée et calculent leurs sorties, qui sont ensuite transmises aux neurones de la couche suivante. Ce processus se poursuit de couche en couche jusqu'à la couche de sortie. Chaque neurone dans une couche cachée est connecté à tous les neurones des couches précédente et suivante, sans connexions entre les neurones d'une même couche (Nielsen, 2015).

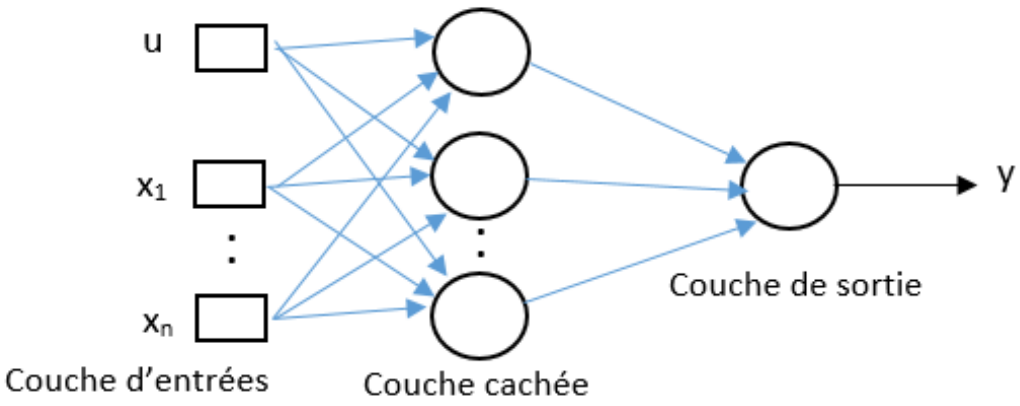

FIGURE I.8 – Réseau de neurones multicouche

I.6.2 Réseaux de Neurones Bouclés (ou Récurrents)

Les réseaux de neurones bouclés, ou réseaux récurrents (Reccurent Neural Network), se distinguent par leurs boucles de rétroaction qui leur permettent de traiter des séquences de données temporelles et d'apprendre des dépendances entre les éléments successifs. Fonctionnant élément par élément, ils actualisent leur état interne en fonction de l'entrée courante et de l'état précédent, puis calculent la sortie.

Les avantages des RNN incluent la capacité à traiter des séquences temporelles, l'apprentissage des dépendances et la modélisation de systèmes dynamiques.

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. Cette forme est décrite par les équations (I.4) et (I.5) :

$$S(k-1) = f(S(k); I(k); W(k)) \tag{I.4}$$

$$O(k) = g(S(k); I(k); W(k)) \tag{I.5}$$

Où :

$O(k)$ est le vecteur des sorties à l'instant k.

$I(k)$ est le vecteur des entrées.

Les fonctions f et g sont des fonctions non linéaires qui peuvent être réalisées par deux réseaux de neurones non bouclés.

$S(k-1)$ est la variable d'état à l'instant $(k-1)$.

$S(k)$ est la variable d'état à l'instant k.

$W(k)$ est le vecteur des poids à l'instant k.

La Figure (I.9) présente la forme canonique d'un réseau de neurones bouclé.

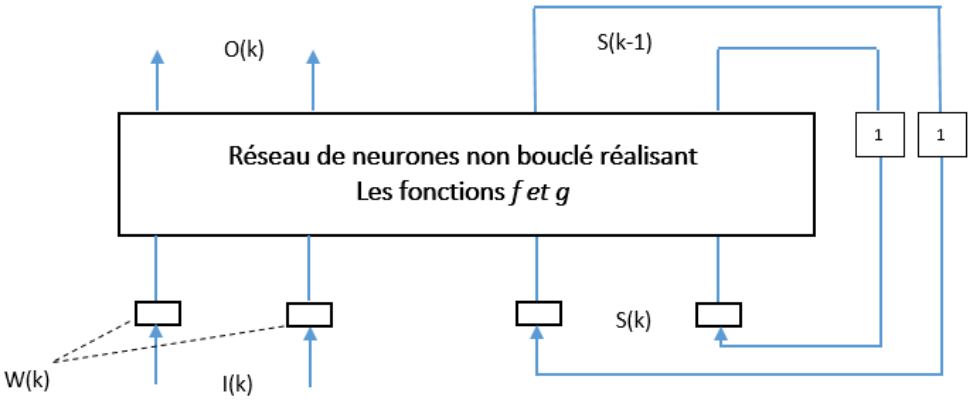

FIGURE I.9 – Forme canonique d'un réseau de neurones bouclé

Les modèles les plus performants et les plus connus incluent **l'architecture de Jordan**, qui est multicouche. Dans ce modèle, les cellules de la première couche sont divisées en deux couches : la couche de contexte et la couche d'entrée. De plus, la dernière couche est bouclée sur la couche de contexte (Jordan, 1986) (voir Figure (I.11)). D'autre part, **l'architecture d'Elman** présente quelques différences avec celle de Jordan. C'est également une structure multicouche où les boucles de rétroaction connectent la couche cachée aux cellules d'entrée. Il y a autant de cellules cachées que de cellules de contexte, et chaque cellule cachée est liée à une seule cellule de contexte (Elman, 1990) (voir Figure (I.10)).

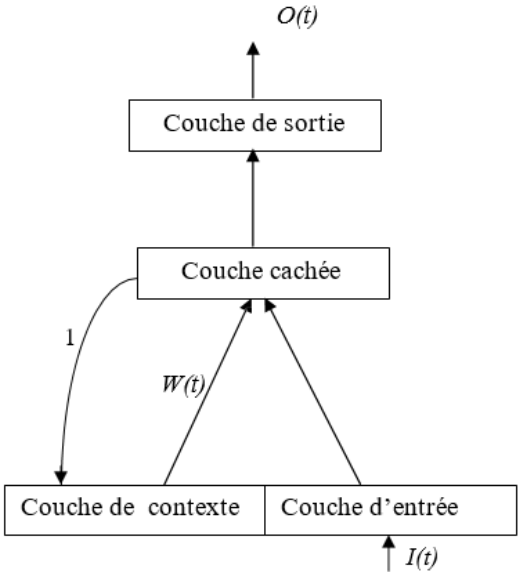

FIGURE I.10 – Architecture d'Elman

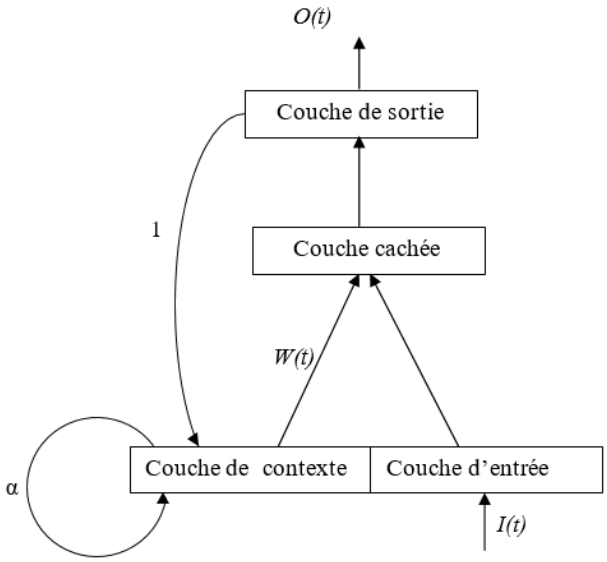

FIGURE I.11 – Architecture de Jordan

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

Le passage d'une modélisation d'état à une modélisation entrée/sortie est un outil puissant pour l'analyse et la commande des systèmes dynamiques. Il permet de simplifier la représentation du système et de se concentrer sur les relations entre les entrées et les sorties. On parle d'une modélisation entrée/sortie lorsqu'on définit comme vecteur d'état le vecteur composé des sorties de processus aux instants précédents. C'est une méthode utile dans l'asservissement, le filtrage et la prédiction (Ljung, 1999).

I.7 Propriétés des réseaux de neurones

Les réseaux de neurones sont des outils de modélisation puissants qui possèdent des propriétés **d'approximation universelle** et de **parcimonie**. Ces propriétés leur permettent de modéliser une grande variété de processus dynamiques non linéaires avec une efficacité remarquable.

Sans se plonger dans les aspects mathématiques détaillés, on peut exprimer la propriété d'approximation de la manière suivante : **"toute fonction bornée suffisamment régulière peut être approchée 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". Il convient de noter que cette caractéristique n'est pas exclusive aux réseaux de neurones, d'autres familles de fonctions paramétrées, telles que les ondelettes, les fonctions radiales, ou encore les fonctions splines, partagent cette propriété (Dreyfus, 1998).

Ce qui rend les réseaux de neurones uniques, c'est leur capacité à réaliser des approximations de façon parcimonieuse : pour une précision équivalente, ces réseaux demandent moins de paramètres ajustables (c'est-à-dire les poids des connexions) que les estimateurs universels classiquement utilisés. Pour être plus précis, le nombre de poids dans un réseau de neurones augmente de manière linéaire avec le nombre de variables de la fonction à approximer, alors qu'il croît de façon exponentielle pour la plupart des autres estimateurs. En ce qui concerne leur attrait industriel, celui-ci découle précisément de leur aptitude à être parcimonieux. Concrètement, lorsqu'un problème implique plus de deux variables, les réseaux de neurones sont généralement préférés à d'autres approches (Dreyfus, 1998).

D'un point de vue conceptuel, la propriété de parcimonie peut être envisagée comme suit : lorsqu'une approximation est constituée d'une combinaison linéaire de fonctions de base prédéfinies (comme des monômes ou des gaussiennes avec des centres et des écarts-types fixes), seuls les coefficients de cette combinaison peuvent être modifiés. En revanche, lorsque l'approximation repose sur une combinaison linéaire de fonctions non linéaires avec des paramètres ajustables (comme dans un Perceptron multicouche), à la fois les coefficients de la combinaison et la forme des fonctions utilisées peuvent être adaptés (Dreyfus, 1998). Ainsi, dans un Perceptron multicouche, les poids de la première couche influent sur la forme de chaque sigmoïde générée par les neurones cachés, tandis que les poids de la deuxième couche déterminent une combinaison linéaire de ces fonctions. On comprend alors que cette flexibilité supplémentaire, permettant d'ajuster la forme des fonctions combinées, conduit à l'utilisation d'un nombre moins important de fonctions de base et donc à un nombre réduit de paramètres à ajuster.

I.8 Apprentissage des réseaux de neurones

La particularité majeure des réseaux de neurones réside dans leur aptitude à l'apprentissage, qui est le mécanisme d'ajustement des paramètres d'un système afin de réaliser de manière optimale la tâche qui lui est assignée. Le type d'apprentissage est spécifié par la façon dont ces paramètres sont adaptés, et diverses méthodes et algorithmes existent pour effectuer ces ajustements.

Il est possible de catégoriser l'apprentissage en deux grandes catégories : l'apprentissage supervisé et l'apprentissage non supervisé.

a. **Apprentissage supervisé**

Dans le cadre des réseaux utilisant un apprentissage supervisé, les données d'entrée ainsi que les sorties souhaitées correspondantes sont fournies au réseau. Le réseau doit ensuite ajuster ses poids pour réduire au minimum la disparité entre la sortie souhaitée et sa propre sortie. Cette procédure est répétée jusqu'à ce qu'un critère de performance soit satisfait. L'algorithme le plus fréquemment employé à cette fin est la rétropropagation du Gradient (Bengio, 2018).

b. **Apprentissage non supervisé**

Pour les réseaux utilisant un apprentissage non supervisé, aucune indication sur la réponse désirée n'est donnée. Une entrée leur est présentée, et ils évoluent librement jusqu'à atteindre un état stable. Ce processus est appelé 'auto-organisation' (Boureau et Bach, 2019).

L'ajustement des poids des neurones est le but principal de l'apprentissage. Celui-ci nécessite des exemples, autrement appelés échantillons d'apprentissage, ainsi qu'un algorithme d'apprentissage.

On considère un ensemble de N mesures de sortie Y_k^p (pour k allant de 1 à N), qui correspondent à N valeurs du vecteur d'entrée X. L'association de ces mesures de sortie et valeurs d'entrée constitue l'ensemble d'apprentissage.

Pour optimiser le vecteur de paramètres W, il est nécessaire de minimiser la fonction de coût. Cette fonction mesure l'écart entre les valeurs prédictives Y_k produites par le modèle

(réseau de neurones) et les mesures réelles Y_k^p. La fonction de coût $J(W)$ communément choisie est de nature quadratique et est formulée comme suit dans l'équation (I.6) :

$$J(W) = \sum_{k=1}^{N} \frac{1}{2}(Y_k^p - Y_k(W))^2 \tag{I.6}$$

Les algorithmes d'apprentissage

Ils sont utilisés pour ajuster les poids et les biais du réseau afin de minimiser une fonction de coût, ce qui permet d'améliorer les performances du réseau dans des tâches telles que la classification ou la prédiction.

a. **Algorithme de minimisation itératif**

Un algorithme itératif de minimisation vise à trouver le minimum du coût en ajustant les paramètres à chaque itération, en se fondant sur le calcul du coût global. Parmi ces méthodes, on compte l'algorithme du Gradient et l'algorithme de Newton.

b. **Algorithme de minimisation récursif**

Un algorithme récursif de minimisation ajuste les paramètres en se fondant sur un coût associé à un nombre restreint d'exemples (coût partiel), procédant à une mise à jour des paramètres après la présentation de chaque exemple. Parmi ces algorithmes, on compte l'algorithme de Windrow-Hoff, la méthode des moindres carrés récursifs, et la récursion du filtre de Kalman étendu.

L'estimation des paramètres peut se faire soit de manière hors ligne (dans un système non adaptatif), soit en ligne (dans un système adaptatif) :

Dans le contexte de **l'apprentissage en ligne**, les séquences d'apprentissage sont typiquement infinies, car le modèle continue à assimiler de nouvelles données en temps réel. Cette caractéristique implique que le processus d'apprentissage peut théoriquement se prolonger indéfiniment (Mé et Oudin, 2017). Dans de telles circonstances, l'emploi d'un algorithme récursif s'avère nécessaire. En revanche, dans **l'apprentissage hors ligne**, les séquences d'apprentissage sont finies, car le modèle est formé sur un ensemble de données statique et préalablement collecté. Une fois que toutes les données d'entraînement ont été utilisées pour ajuster les paramètres du modèle, le processus d'apprentissage est terminé, et le modèle est

ensuite évalué sur des données de test distinctes. Dans cette situation, l'algorithme d'apprentissage peut être soit itératif, soit récursif, selon les besoins du problème et les caractéristiques des données (Mé et Oudin, 2017).

I.9 Conclusion

Dans ce chapitre, nous avons exploré les bases des réseaux de neurones artificiels, en abordant les définitions essentielles, les différentes architectures neuronales et les techniques d'apprentissage associées. Cette étude nous a permis de mieux comprendre le fonctionnement des réseaux de neurones artificiels, ainsi que leur utilisation dans la résolution de divers problèmes, notamment la modélisation et l'identification des processus statiques ou dynamiques. Nous avons également souligné l'importance de l'apprentissage des réseaux de neurones, qui consiste à ajuster les paramètres du réseau sur la base d'un ensemble d'exemples d'entraînement, afin de minimiser l'erreur quadratique moyenne entre les prédictions du modèle et les valeurs réelles observées. Les réseaux de neurones offrent l'avantage de pouvoir résoudre des problèmes complexes sans nécessiter de règles mathématiques explicites, et ils peuvent souvent atteindre une meilleure approximation avec moins de paramètres ajustables que les méthodes traditionnelles.

Dans le prochain chapitre, nous explorerons un nouveau type de réseau dynamique, les réseaux de neurones à espace d'état, qui présentent des caractéristiques intéressantes pour la modélisation et la prédiction de séquences temporelles.

Chapitre II

Réseaux de neurones à espace d'état

II.1 Introduction

Les Réseaux de Neurones à Espace d'État (SSNN) ont émergé dans les années 1940 et 1950, lorsque les premières recherches sur les réseaux de neurones biologiques ont inspiré les premiers modèles informatiques de neurones artificiels, établis par des chercheurs comme Warren McCulloch et Walter Pitts (McCulloch et Pitts, 1943). Au fil des décennies suivantes, les SSNN ont continué à évoluer grâce aux progrès technologiques dans les domaines de l'informatique, de l'apprentissage automatique et de la neurobiologie. Aujourd'hui, les SSNN sont largement utilisés dans diverses applications telles que la reconnaissance vocale, la traduction automatique, la prédiction de séries chronologiques et la modélisation de systèmes dynamiques.

Ce chapitre présente un type spécifique de réseaux de neurones récurrents utilisé pour la modélisation et la commande numérique des processus, connu sous le nom de Réseaux de Neurones à Espace d'État, ou State Space Neural Network (SSNN) en anglais. Avant d'aborder en détail les SSNN, nous rappelons quelques notions fondamentales de la représentation d'état des systèmes. Nous nous situons dans le cadre général des modèles à temps discret des processus, cadre qui est approprié tant pour la commande numérique que pour l'utilisation de réseaux de neurones formels.

II.2 Représentation d'état

La représentation d'état en automatique constitue un outil essentiel pour la modélisation des systèmes dynamiques. Elle repose sur l'utilisation de variables d'état et peut prendre des formes linéaires ou non linéaires, continues ou discrètes. Cette approche permet de prédire l'état du système à tout moment ultérieur, pour autant que l'état initial et l'évolution des variables extérieures influant sur le système soient connus (Jaulin, 2005). La représentation d'état d'un système n'est pas unique, bien au contraire pour un système donné il en existe une infinité (Granjon, 2015). La forme générale des équations d'état en temps continu est la suivante :

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

Où :

$\vec{x}(t)$ est le vecteur d'état du système, représentant les variables internes du système qui évoluent dans le temps.

$\vec{u}(t)$ est le vecteur d'entrée ou de commande du système, représentant les signaux externes qui influencent le comportement du système.

f et g sont des fonctions qui décrivent comment les variables d'état évoluent dans le temps en fonction des états actuels et des entrées.

$\vec{y}$ est le vecteur de sortie du système.

La première équation, qui est une équation différentielle, s'appelle équation d'état ; la deuxième s'appelle équation d'observation (ou équation de sortie).

Dans le cas des systèmes linéaires f et g sont linéaires, ainsi la représentation d'état devient :

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

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

II.2.1 Commandabilité

C'est une propriété fondamentale des systèmes dynamiques, en particulier des systèmes linéaires. Elle reflète la capacité à influencer l'ensemble des états d'un système par le biais de ses entrées (ou commandes). Si un système est commandable, cela signifie qu'il est possible de le conduire de n'importe quel état initial à n'importe quel état final dans un intervalle de temps fini, en utilisant une séquence appropriée de commandes (Rugh, 1996).

Le critère de Kalman est le plus connu pour tester la commandabilité d'un système linéaire. Selon ce critère, un système est commandable si et seulement si la matrice de commandabilité est de plein rang. La matrice de commandabilité est construite à partir de la matrice d'état A et de la matrice de commande B du système. Elle est définie comme suit :

$$P_c = \left[[B \; AB \; A^2B \cdots A^{(n-1)}B]\right]$$

où n est le nombre d'états du système. Si le rang de la matrice P_c est égal à n, alors le système est commandable.

II.2.2 Observabilité

L'observabilité est une propriété fondamentale des systèmes dynamiques qui détermine si l'état complet d'un système peut être déduit à partir de ses sorties (des mesures) sur un intervalle de temps fini. Si un système est observable, cela signifie que, connaissant les sorties et les entrées sur une période donnée, il est possible de calculer l'état initial du système.

Le critère de Kalman pour l'observabilité est similaire à celui de la commandabilité. Un système est observable si et seulement si la matrice d'observabilité est de plein rang. La matrice d'observabilité est construite à partir de la matrice d'état A et de la matrice de sortie C du système. Elle est définie comme suit :

$$P_o = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{(n-1)} \end{bmatrix}$$

où n est le nombre d'états du système. Si le rang de la matrice P_o est égal à n, alors le système est observable.

II.2.3 Discrétisation et l'influence de la période d'échantillonnage

Travailler dans le domaine discret est souvent nécessaire, notamment lorsque le temps qui régit le système est discrétisé, telle que dans le contexte informatique où le temps est mesuré en unités discrètes (k). La discrétisation est un processus permettant de convertir une représentation d'état continue en une représentation échantillonnée ou discrète du système.

Dans cette conversion, deux éléments clés sont impliqués :

— Les échantillonneurs, qui captent périodiquement la valeur du signal selon une période d'échantillonnage définie.

— Les bloqueurs, généralement d'ordre zéro, qui maintiennent la commande u constante sur des intervalles définis, tels que $[kTe, (k+1)Te]$, comme illustré dans la Figure (II.1) suivante :

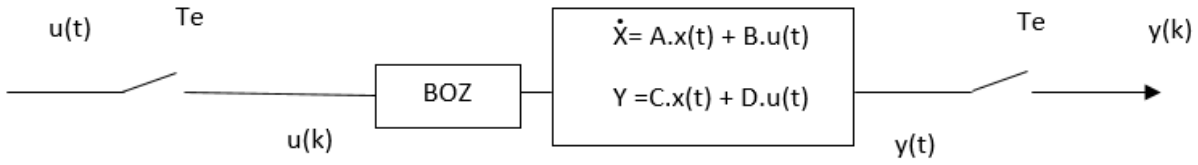

FIGURE II.1 – Système continu échantillonné

L'état échantillonné est représenté comme ceci :

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

Où :

$A_D = e^{At} = \phi(t)$

$B_D = \int_0^t e^{At} B d\alpha$

$C_D = C$

Pour obtenir une représentation précise du comportement d'un système en passant du domaine continu au domaine discret, il est crucial de choisir soigneusement la fréquence d'échantillonnage, qui détermine le taux auquel les mesures sont prises. Conformément au théorème de Shannon, si un signal continu possède une fréquence maximale $Fmax$, il peut être échantillonné sans perte d'information si la fréquence d'échantillonnage Fe est choisie de telle sorte à ce que Fe soit supérieure à deux fois $Fmax$.

Un mauvais choix de fréquence d'échantillonnage peut conduire à des problèmes tels que la perte de stabilité ou la perte d'observabilité pour le système échantillonné (Van der Smagt, 2001). En d'autres termes, si la fréquence d'échantillonnage est trop basse par rapport à la fréquence maximale du signal, des informations importantes peuvent être perdues, ce qui compromet la capacité à reconstruire précisément le signal continu à partir de ses échantillons discrets. En revanche, une fréquence d'échantillonnage trop élevée peut entraîner une surcharge de calcul et de ressources, ainsi que des effets indésirables. Ainsi, le choix de la fréquence d'échantillonnage est critique pour assurer la fidélité de la représentation du système échantillonné.

Dans une représentation d'état en temps discret, l'objectif est d'exprimer l'état du système à un instant donné en fonction du signal d'entrée et de son état précédent. Cette représentation est particulièrement utile pour modéliser et analyser le comportement dynamique des systèmes.

— Pour un système non linéaire, l'évolution de l'état du système est généralement décrite par une équation de la forme :

$$\begin{cases} x(k+1) = h(x(k), u(k)) \\ y(k) = g(x(k)) \end{cases} \tag{II.4}$$

Ici, $x(k)$ représente l'état du système à l'instant k, $u(k)$ est le signal d'entrée au même instant, et $x(k+1)$ est l'état du système à l'instant suivant. Les fonctions h et g sont des fonctions non linéaires qui modélisent l'évolution du système dans le temps en tenant compte de l'état actuel $x(k)$ et du signal d'entrée $u(k)$.

— Pour un système linéaire, les équations d'état prennent généralement la forme standard

suivante :

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

Ici, A est la matrice d'état qui définit l'évolution de l'état du système en fonction de son état actuel $x(k)$, B est la matrice de commande qui représente l'effet du signal d'entrée $u(k)$ sur l'évolution de l'état, et $x(k+1)$ est l'état du système à l'instant suivant.

L'équation $y(k) = Cx(k)$ représente la relation entre la sortie observable $y(k)$ d'un système à un instant donné k. La matrice C spécifie comment les variables d'état contribuent à la sortie observée. Cette équation est essentielle pour modéliser et comprendre le comportement dynamique des systèmes dans le cadre de la représentation d'état en temps discret.

Il est important de noter que la modélisation linéaire est un cas particulier de la modélisation non linéaire, où les fonctions h et g sont remplacées par des opérations linéaires représentées par les matrices A, B et C.
La représentation graphique de ces équations d'état est illustrée sur la Figure (II.2) suivante :

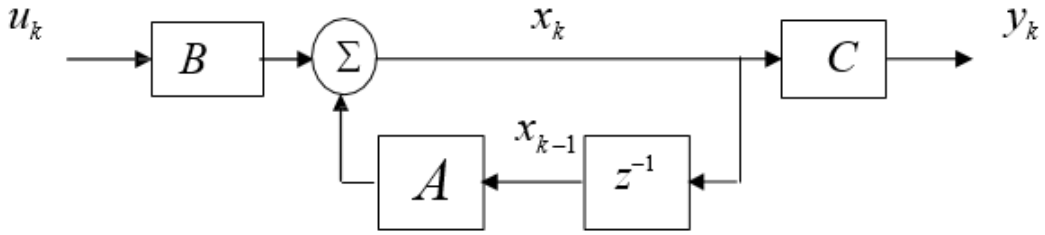

FIGURE II.2 – Représentation schématique d'une modélisation d'état en temps discret

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

Le réseau neuronal à espace d'états, communément appelé State Space Neural Network (SSNN), est une variante de réseau de neurones récurrents spécialement conçue pour représenter avec précision la dynamique complexe d'un système non linéaire dans son espace d'état

(Haykin, 2009). Son utilisation dans l'identification de systèmes non linéaires est bien établie, ayant démontré son efficacité dans divers domaines industriels (santé, finance, économie, environnement...etc) (Douglas et Szu, 1993).

Ces modèles neuronaux à espace d'état sont configurés pour représenter fidèlement les interactions complexes entre les variables d'état d'un système dynamique, ce qui les rend particulièrement adaptés à la prédiction et à la modélisation de comportements temporels, en optimisant les processus d'apprentissage grâce à leur structure spécifique, les SSNN offrent une approche robuste et efficace pour relever les défis liés à la modélisation et à l'identification des systèmes dynamiques non linéaires.

La structure générale d'un SSNN est représentée par le schéma bloc dans la Figure (II.3).

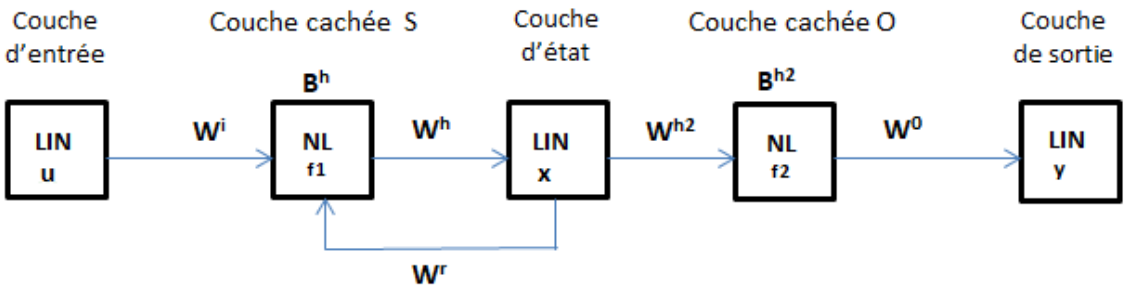

FIGURE II.3 – Schéma bloc d'un réseau de neurones à espace d'état

LIN : éléments linéaires.

NL : éléments non linéaires.

$f1$, $f2$: fonctions non linéaires.

Ce réseau de neurones a la capacité de reproduire le comportement d'un système dynamique décrit par l'équation (II.6).

$$\begin{cases} \vec{x}(k+1) = f(\vec{x}, \vec{u}) \\ \vec{y}(k) = g(\vec{x}(k)) \end{cases} \tag{II.6}$$

Où :

f et g sont deux fonctions non linéaires statiques.

$\vec{x}$ est le vecteur d'état , $\vec{u}$ le vecteur d'entrée, et $\vec{y}$ est le vecteur de sortie.

— **Chaque bloc dans la Figure (II.3) correspond à une couche spécifique du SSNN :**

Couche d'entrée : C'est la première couche du réseau, où les données initiales sont introduites pour être traitées par le réseau. Elle est composée de neurones qui reçoivent les valeurs d'entrée et les transmettent aux couches suivantes du réseau.

Couche cachée (S) : La lettre (S) vient du mot "State", C'est une couche intermédiaire qui ne communique pas directement avec l'extérieur. Elle est responsable de mettre en évidence les relations entre les variables et de traiter les informations reçues de la couche d'entrée. Cette couche joue un rôle crucial dans le traitement des données et l'extraction des caractéristiques pertinentes. Elle précède la couche d'état et représente la fonction non linéaire qui décrit le comportement des états.

Couche d'état : Chaque neurone de cette couche représente un état du système, et la sortie du neurone est la valeur de l'état $x(k)$.

Couche cachée (O) : La couche cachée (O) d'un réseau de neurones à espace d'état est une autre couche intermédiaire qui ne communique pas directement avec l'extérieur. Elle reçoit des informations de la couche d'état ou de la couche cachée (S) et les traite pour préparer la sortie du réseau. Cette couche contribue à la transformation des données et à la représentation intermédiaire des informations. La lettre (O) provient du mot "Output" et désigne la couche précédant la couche de sortie. Elle représente la fonction non linéaire qui relie les sorties du réseau aux états.

Couche de sortie : La couche de sortie d'un réseau de neurones à espace d'état est la dernière couche du réseau, où les résultats finaux ou les prédictions sont générés. Elle communique les résultats du traitement effectué par les couches précédentes et constitue la sortie finale du réseau de neurones.

— Ces cinq couches jouent des rôles distincts mais complémentaires dans un réseau de neurones à espace d'état, permettant ainsi de capturer et de traiter efficacement des

données séquentielles.

Le fonctionnement du SSNN peut être décrit par l'équation (II.7), qui est capable de reproduire exactement le comportement de la représentation d'état du système donnée par l'équation (II.6).
Le modèle (II-7) peut être représenté comme suit :

$$\begin{cases} \hat{x}(k+1) = W^h f1(W^r \hat{x}(k) + W^i \vec{u}(k) + B^h) \\ \hat{y}(k) = W^0 f2(W^{h2} \hat{x}(k) + B^{h2}) \end{cases} \tag{II.7}$$

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 contenant les éléments h et $h2$, qui servent à ajuster l'entrée nette de la fonction d'activation en la faisant augmenter ou diminuer.

$f1$ et $f2$ sont deux fonctions non-linéaires et elles sont généralement de forme sigmoïdales.

Le Réseau de neurones de la Figure (II.3) peut être représente d'une manière plus détaillée par le réseau de neurones de la Figure (II.4).

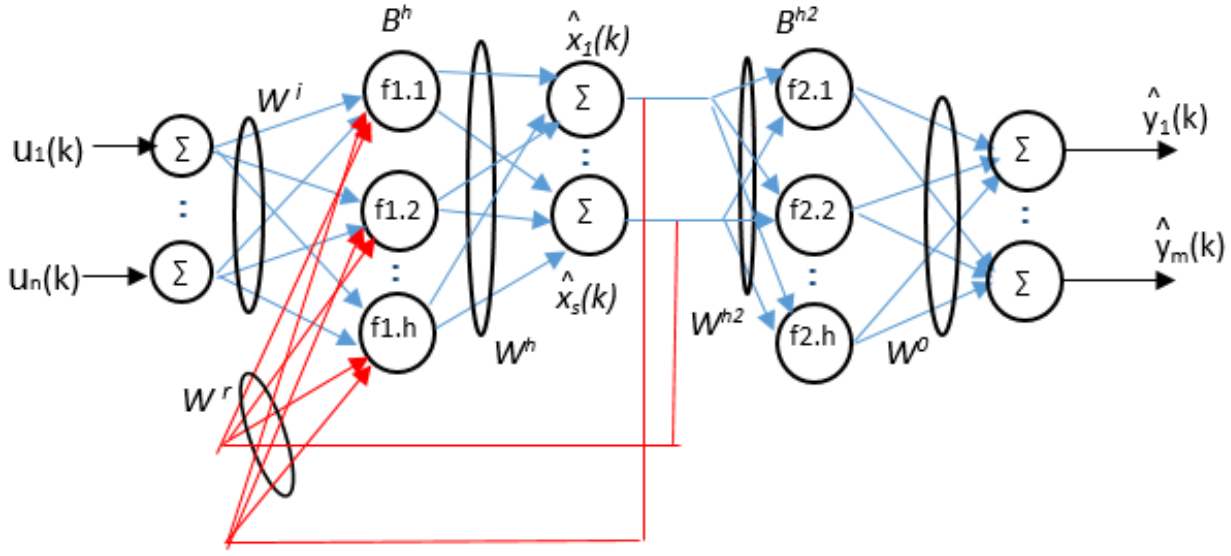

FIGURE II.4 – Réseau de neurones à espace d'état

II.4 Avantages et Inconvénients :

Les réseaux de neurones à espace d'état (SSNN) se distinguent des réseaux de neurones classiques par leur capacité à modéliser et prédire des systèmes dynamiques en utilisant une représentation d'état échantillonnée. La structure d'un SSNN est identique à la représentation schématique d'une représentation d'état en temps discret, ce qui permet d'intégrer efficacement les informations antérieures grâce à des représentations internes. Ces représentations, sous forme de vecteurs ou d'ensembles de variables, résument l'état actuel et les états passés du système, capturant ainsi les dynamiques temporelles et les dépendances séquentielles des données.

Les SSNN offrent une grande flexibilité en représentant les différents états du réseau comme les points d'un hypercube dans un espace multidimensionnel. Cela permet une exploration approfondie des configurations possibles et une analyse précise des données. En intégrant le contexte temporel et les dépendances séquentielles, les SSNN améliorent la précision et la pertinence des prédictions, rendant leur utilisation bénéfique dans de nombreuses applications.

L'inconvénient principal des réseaux de neurones à espace d'état (SSNN) est leur complexité de modélisation et de mise en œuvre. Ces réseaux peuvent être plus complexes à modéliser et à mettre en œuvre que les réseaux multicouches (MLP), nécessitant une compréhension approfondie de la théorie des systèmes dynamiques et des équations d'état (Van der Smagt, 2001).

II.5 Fonctionnement et conditions initiales

Le fonctionnement d'un réseau de neurones à espace d'état (SSNN) est étroitement lié à la représentation d'état utilisée en automatique pour modéliser un système dynamique. La représentation d'état permet de déterminer l'état du système à un moment donné en se basant sur les informations de l'instant précédent. Dans le contexte d'un SSNN, cela se traduit par la mise à jour itérative de l'état du système à chaque itération du réseau. Par exemple, la première sortie du réseau dépend de l'état initial, et une fois cette sortie obtenue, l'état suivant peut être calculé en fonction de cet état initial et de l'entrée de commande. Ce

processus de mise à jour de l'état est répété de manière itérative à chaque pas de temps du système.

Les conditions initiales du SSNN sont cruciales pour son bon fonctionnement. L'état initial du système, noté $x(0)$, est un choix crucial et peut être déterminant pour les performances du réseau. Si les valeurs initiales de l'état du système sont connues, le SSNN peut être initialisé avec ces valeurs, ce qui facilite la convergence et l'adaptation du réseau aux dynamiques du système. Cependant, lorsque les valeurs initiales ne sont pas connues, il est possible de les fixer à des valeurs nulles (Zamarreno, Vega, Garcia et Francisco, 2000). Cette approche permet d'initialiser le réseau de manière appropriée en l'absence d'informations sur l'état initial du système. En résumé, le SSNN fonctionne en mettant à jour itérativement l'état du système en fonction des informations de l'instant précédent, et les valeurs initiales de cet état sont essentielles pour son bon fonctionnement, qu'elles soient connues ou fixées de manière arbitraire.

La Figure (II.5) montre le principe itératif du SSNN.

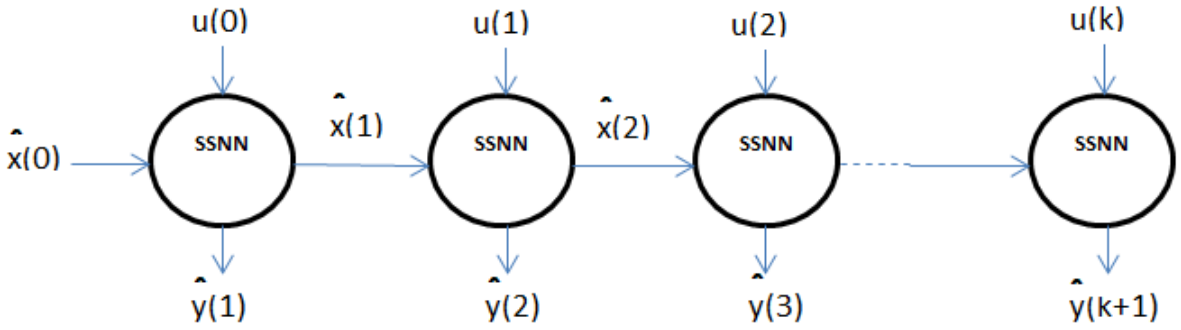

FIGURE II.5 – Principe itératif du SSNN

II.6 Apprentissage des réseaux de neurones à espace d'état

Le SSNN se compose de deux sous-réseaux à une seule couche cachée : le premier est un réseau bouclé (dynamique) correspondant à l'équation d'état, tandis que le deuxième est

un réseau non bouclé (statique) correspondant à l'équation de sortie. Cette structure est représentée graphiquement sur les Figures (II.6) et (II.7) pour une meilleure compréhension.

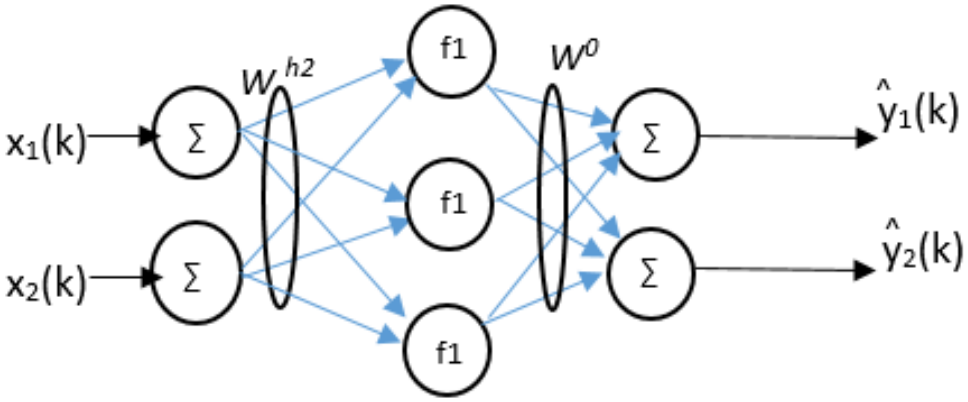

FIGURE II.6 – Réseau de neurones statique

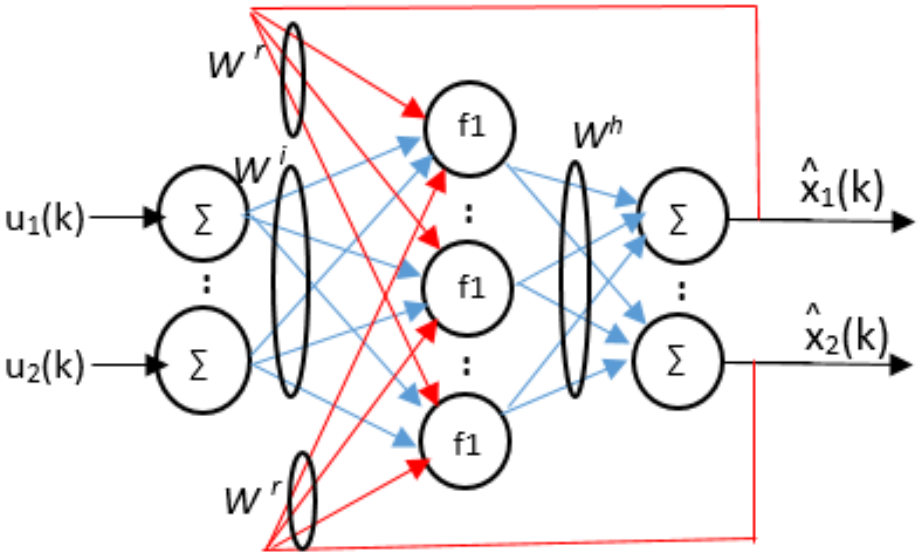

FIGURE II.7 – Réseau de neurones dynamique

II.6.1 Apprentissage semi dirigé

Dans ce cas les variables d'état ne sont pas mesurées (Haykin, 2009). Dans ce scénario, les seules données disponibles pour l'apprentissage sont la séquence des entrées de commande $u(k)$ et la séquence des sorties mesurées $y_p(k)$ (Jang et al, 1997). Cette configuration est courante dans de nombreux systèmes où toutes les variables d'état ne peuvent pas être directement mesurées (Bengio, Goodfellow et Courville, 2016). Dans ce cas, l'apprentissage est dit semi-dirigé, car bien que les sorties mesurées soient disponibles, les variables d'état ne le sont pas (Haykin, 2009). Pour réaliser cet apprentissage semi-dirigé, différentes méthodes peuvent être utilisées, notamment la rétropropagation à travers le temps, qui est une technique couramment utilisée pour entraîner les réseaux de neurones récurrents. Cette approche permet d'adapter les poids du réseau afin de minimiser l'erreur entre les sorties prédites par le réseau et les sorties mesurées dans les données d'apprentissage.

La Figure (II.8) montre l'apprentissage semi dirigé.

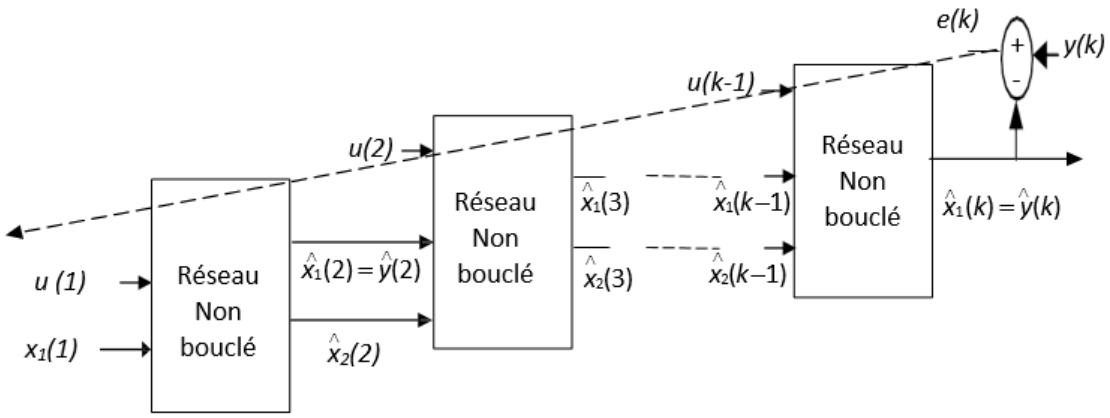

FIGURE II.8 – Apprentissage semi dirigé

II.6.2 Apprentissage dirigé

Nous abordons une situation où les variables d'état sont mesurées, ce qui signifie que l'apprentissage est dirigé (supervisé) (Haykin, 2009). Les données d'apprentissage comprennent les séquences d'entrées de commande $u(k)$, les séquences des variables d'état $x_p(k)$, et les séquences de sorties mesurées $y_p(k)$ (Jang, Sun et Mizuo, 1997).

Dans ce système, deux sous-réseaux distincts sont impliqués : le premier est responsable de l'approximation de l'état à partir de l'équation d'état (Bengio, Goodfellow et Courville, 2016), tandis que le deuxième s'attaque à l'approximation de la sortie à partir de l'équation de sortie. L'apprentissage de ces deux sous-réseaux est dirigé par l'état xp du processus (Haykin, 2009). Les erreurs ex et ey sont intégrées dans les deux fonctions de coût à minimiser, et chacune de ces erreurs sert d'entrée à un réseau utilisant la rétropropagation pour calculer ses poids (Mozer, 2000).

L'apprentissage dirigé est représenté par la figure (II.9).

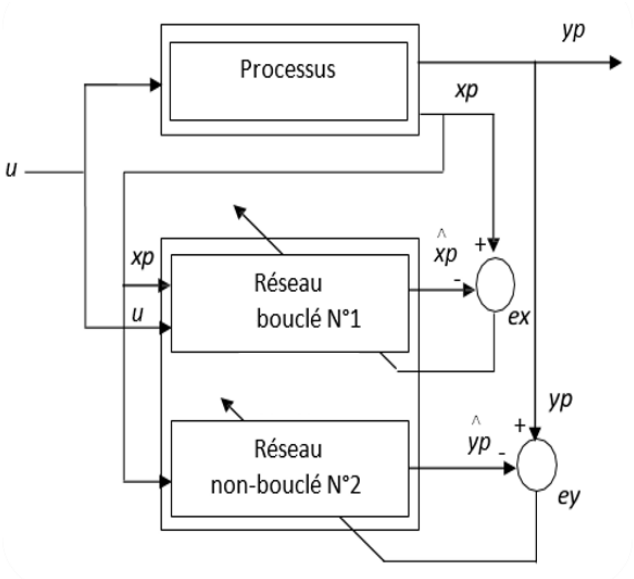

FIGURE II.9 – Apprentissage dirigé

II.6.3 Méthodes d'apprentissage

Pour l'apprentissage de ces sous-réseaux, différentes méthodes peuvent être utilisées :

- **Méthodes du premier ordre** : Reposent sur le Gradient de la fonction de coût. Elles visent à atteindre le minimum de cette fonction en ajustant les poids jusqu'à ce que le Gradient (ou dérivée) de la fonction soit nul. Parmi ces méthodes, on trouve le Gradient stochastique.

- **Méthodes du deuxième ordre** : Nécessitent la dérivée seconde de la fonction de coût. Elles incluent des approches telles que la méthode de Levenberg-Marquardt, connue

pour sa capacité à converger plus rapidement grâce à sa prise en compte de la courbure de la fonction de coût.

a/ L'algorithme du Gradient stochastique

C'est une méthode d'optimisation itérative utilisée pour entraîner des réseaux de neurones à espace d'état (SSNN). Il vise à minimiser une fonction de coût, généralement l'erreur quadratique moyenne (EQM), en ajustant progressivement les paramètres du réseau (poids et biais) à chaque itération.

1. Choisir la taille du réseau et initialiser les poids et les seuils du réseau (selon le système à modéliser).

2. Calculer les sorties des différentes couches, où la sortie du neurone i est déterminée par l'équation suivante :

$$y_i = g\left(\sum_{j=1}^{n_1} W_{ij} \cdot y_j\right) \tag{II.8}$$

   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.

   W_{ij} s'agit du poids de la connexion entre le neurone j de la couche précédente et le neurone i de la couche actuelle.

   y_j est la sortie du neurone j dans la couche précédente.

   $\sum_{j=1}^{n_1}$ correspond à la somme des produits des sorties y_j de chaque neurone j de la couche précédente, pondérés par les poids respectifs W_{ij}.

   n_1 représente le nombre de neurones dans la couche précédente.

3. Calculer la fonction de coût partiel, généralement on choisit une fonction de coût quadratique. L'erreur quadratique sur la sortie est :

$$J = \frac{1}{2}\sum_{i=1}^{n_2}(y_i - y_{di})^2 \tag{II.9}$$

   J est la fonction de coût, qui mesure à quel point les prédictions du réseau de neurones s'éloignent des valeurs réelles ou désirées.

y_i est la sortie du réseau de neurones (prédite), y_{di} est la sortie désirée (mesurée).

$\sum_{i=1}^{n2}$ correspond à la somme des erreurs pour chaque neurone i dans la couche de sortie.

n_2 est le nombre de neurones dans la couche de sortie.

4. Calculer le Gradient de la fonction de coût par la rétropropagation du Gradient 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 Liang, 2007).

5. Modifier les paramètres selon la relation suivante :

$$W(k) = W(k-1) - \mu(k)(\partial J/\partial W)\backslash_{W(k-1)} \tag{II.10}$$

Avec $\mu(k) > 0$ le pas de progression dans la direction de descente qui est équivalente à l'inverse du Gradient. La valeur de $\mu(k)$, généralement choisie entre 0 et 1, est d'une importance capitale. Elle peut favoriser une convergence plus rapide si elle est proche de 1, ou la ralentir si elle est proche de 0 (Personnaz, Gluck et Rumelhart, 2003).

$\partial J/\partial W$ c'est le Gradient de la fonction de coût J par rapport aux poids W, évalué avec les poids de l'itération précédente $k-1$. Ce Gradient indique la direction dans laquelle il faut ajuster les poids pour minimiser la fonction de coût.

b/ L'algorithme de Levenberg-Marquardt

Il est également connu sous le nom de méthode des moindres carrés amortis, est une technique d'optimisation puissante utilisée pour entraîner les réseaux de neurones à espace d'état (SSNN). Contrairement au Gradient stochastique qui se déplace uniquement dans la direction du Gradient, le Levenberg-Marquardt combine les forces du Gradient et de la méthode de Newton pour trouver un minimum plus rapidement et plus efficacement dans des situations complexes.

1. Définir une valeur initiale pour les paramètres du modèle W, et choisir un paramètre de régularisation λ initial.

2. Calcule des sorties des différentes couches.

3. Calculer la fonction de coût partiel.

4. Calculer le Gradient de la fonction de coût $J(W)$ par rapport aux paramètres du modèle. Ce gradient est souvent noté $\nabla J(W)$.

5. Calculer la matrice hessienne $H(W)$, qui est une approximation de la dérivée seconde de la fonction de coût par rapport aux paramètres du modèle.

   Avec : $H = Ja^T Ja$

   Ja représente la matrice jacobienne (Un calcul approfondi de la matrice jacobienne est présenté dans la thèse de doctorat de Mahul (Mahul, 2005)).

6. Mettre à jour les paramètres du modèle en utilisant la formule suivante :

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

   ou $W(i)$ est la nouvelle valeur des paramétrés a l'itération i, $W(i-1)$ est la valeur précédente des paramètres, I est la matrice identité, λ est le paramètre de régularisation, $\nabla J(W(i-1))$ est le Gradient de la fonction de cout par rapport aux paramètres à l'itération $i-1$, et $H(W(i-1))$ est la matrice hessienne à l'itération $i-1$.

7. Mise à jour du paramètre de régularisation : Si l'itération a conduit à une amélioration de la fonction de coût, diminuer le paramètre de régularisation λ. Sinon, l'augmenter.

8. Répéter les étapes 2 à 7 jusqu'à ce qu'un critère d'arrêt soit satisfait. Ce critère peut être un nombre maximal d'itérations, une tolérance sur l'amélioration de la fonction de coût, ou tout autre critère approprié.

II.7 Utilisations et applications des SSNN

Les réseaux de neurones à espace d'état (SSNN) sont des modèles de machine learning qui peuvent être utilisés comme simulateurs ou prédicteurs à un pas. Ils sont conçus pour modéliser des relations complexes et non linéaires, ce qui les rend particulièrement efficaces dans de nombreux domaines.

- **Les modèles de simulation** jouent un rôle crucial dans la prédiction à long terme des processus qu'ils représentent, opérant de manière autonome pour imiter fidèlement le

comportement de ces derniers. Souvent déployés pour valider la conception d'un système avant sa production ou pour anticiper les tendances sur le long terme, ces modèles ne peuvent, dans le cas des modèles neuronaux, utiliser les sorties mesurées directement du processus comme entrées. Au lieu de cela, ils se basent sur leurs propres prédictions antérieures pour leurs entrées. Cette caractéristique rend l'estimation des paramètres pour ces modèles non adaptative.

- **Les modèles prédicteurs à un-pas** travaillent de concert avec le processus qu'ils représentent, en incluant une partie des sorties mesurées de ce dernier en tant qu'entrées. Leur objectif est de prédire la sortie du processus juste après un échantillonnage. Par exemple, en se basant sur les sorties précédentes du processus à l'instant k, il est possible d'anticiper sa sortie à l'instant $k+1$.

Les Figures (II.10) et (II.11) représentent le modèle simulateur et le modèle prédicteur du SSNN respectivement.

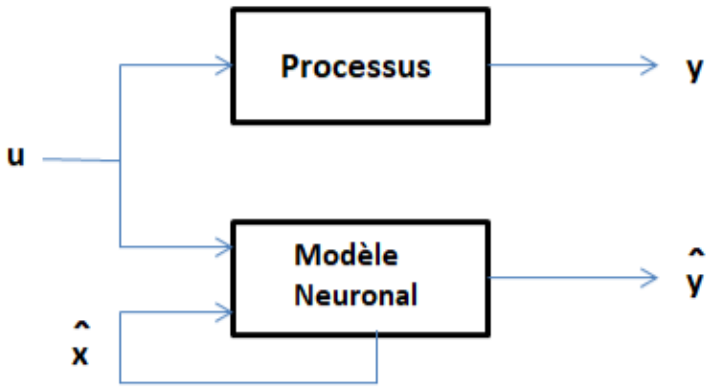

FIGURE II.10 – Modèle simulateur du SSNN

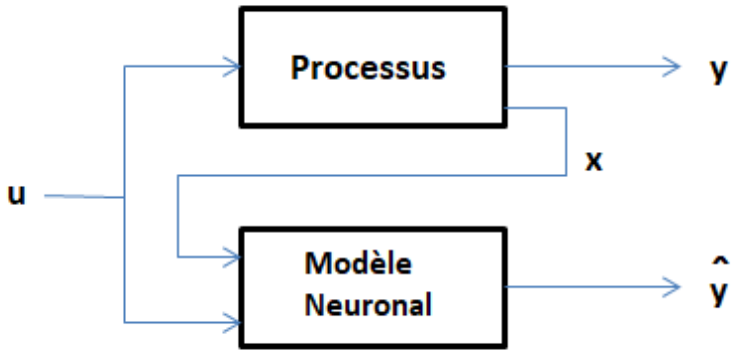

FIGURE II.11 – Modèle prédicteur du SSNN

II.7.1 Choix du nombre de neurones

Dans un réseau de Neurones à Espace d'État (SSNN), il est essentiel d'adapter le nombre de neurones dans les couches d'entrée et de sortie aux dimensions du système à modéliser, afin de capturer toutes ses caractéristiques pertinentes. Concernant les couches cachées, le nombre de neurones est déterminé par la nécessité d'assurer une bonne capacité de généralisation de l'estimateur neuronal. Contrairement aux couches d'entrée et de sortie, ce choix n'est pas imposé par la structure du système, mais vise à trouver un équilibre entre une modélisation précise et une capacité de généralisation adéquate. Un nombre insuffisant de neurones dans la couche cachée peut entraîner une modélisation inexacte du comportement dynamique du système, tandis qu'un nombre excessif peut provoquer un sur-ajustement. Il est donc crucial de sélectionner judicieusement ce paramètre pour garantir à la fois la performance de modélisation et la robustesse de l'estimateur neuronal (Gil, Henriques, Dourado et Duarte-Ramos, 2005).

II.7.2 Applications des SSNN

Les réseaux de neurones à espace d'état (SSNN) sont utilisés dans divers domaines pour prédire et optimiser le comportement des systèmes dynamiques, tels que les robots ou les machines, en appliquant des techniques de commande prédictive. Ils servent également à modéliser des systèmes physiques, comme les moteurs ou les réseaux électriques, afin de mieux comprendre et contrôler leur fonctionnement. Dans le secteur industriel, les SSNN sont

particulièrement efficaces pour la détection d'anomalies, permettant d'anticiper les pannes avant qu'elles ne surviennent.

Les SSNN jouent également un rôle crucial dans la navigation des véhicules autonomes, tels que les voitures ou les drones, en fournissant des directives en temps réel. En outre, ils sont capables d'ajuster automatiquement les réglages des machines dans des systèmes adaptatifs en fonction des conditions changeantes. Enfin, ces réseaux trouvent des applications dans des domaines comme la reconnaissance vocale et l'analyse de données, par exemple pour l'interprétation de la parole ou l'analyse de signaux médicaux.

II.8 Conclusion

En conclusion, les réseaux de neurones à espace d'état (SSNN) se révèlent être des outils extrêmement efficaces pour l'étude des systèmes dynamiques. Leur structure mathématique, qui ressemble aux équations d'état non linéaires de l'automatique classique, leur permet de représenter de manière précise ces systèmes. Cette similitude offre plusieurs avantages notables, notamment une représentation intuitive qui facilite l'analyse, une capacité à estimer directement les propriétés du système grâce aux paramètres du réseau, une polyvalence dans l'identification des systèmes dynamiques linéaires et non linéaires, ainsi qu'une capacité à apprendre à partir de données réelles pour modéliser des systèmes concrets. De plus, le fait d'utiliser le même espace d'état et des grandeurs similaires rapproche les SSNN de l'automatique classique, ouvrant ainsi la voie à une collaboration fructueuse entre ces deux domaines. Cette convergence offre des perspectives prometteuses pour l'analyse, la modélisation, le contrôle et la commande des systèmes dynamiques, en exploitant la puissance de l'apprentissage automatique pour répondre aux problématiques de l'automatique classique.

Chapitre III

Applications

III.1 Introduction

Dans la continuité du chapitre précédent, qui explorait les réseaux de neurones à espace d'état (SSNN), nous allons maintenant examiner leur application à la modélisation de processus dynamiques simulés. Deux algorithmes d'apprentissage sont utilisés à cette fin : la méthode du second ordre, également connue sous le nom de méthode de Levenberg-Marquardt, technique courante pour l'apprentissage des réseaux dynamiques, et la méthode du premier ordre, appelée méthode du Gradient stochastique, bien qu'elle soit souvent associée à d'autres types de réseaux comme les réseaux de neurones récurrents, elle peut être appliquée aux SSNN pour optimiser les paramètres en utilisant des Gradients calculés de manière itérative. Les données d'entraînement utilisées pour les SSNN proviennent du modèle mathématique du processus dynamique étudié. Cette approche permet d'assurer que le réseau est correctement entraîné pour reproduire les comportements dynamiques du système simulé, garantissant ainsi une modélisation précise et représentative. En outre, pour cette modélisation, deux applications linéaires et deux applications non linéaires ont été employées, offrant une vue d'ensemble des performances des SSNN dans des contextes variés.

III.2 Identification de systèmes linéaires

III.2.1 Identification d'un système linéaire d'ordre 1

Considérons un système linéaire de premier ordre dont la représentation d'état est la suivante :

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) \end{cases} \tag{III.1}$$

Avec :

$$A = 0.9429 \qquad B = 0.01942 \qquad C = 1.17$$

Le modèle SSNN est un réseau neuronal qui est construit avec une entrée de commande unique, deux neurones dans la première couche cachée, un neurone d'état unique, deux neurones dans la deuxième couche cachée, et enfin un neurone de sortie unique. On peut représenter cette structure par la séquence (1,2,1,2,1). La Figure (III.1) présente cette architecture de manière illustrative.

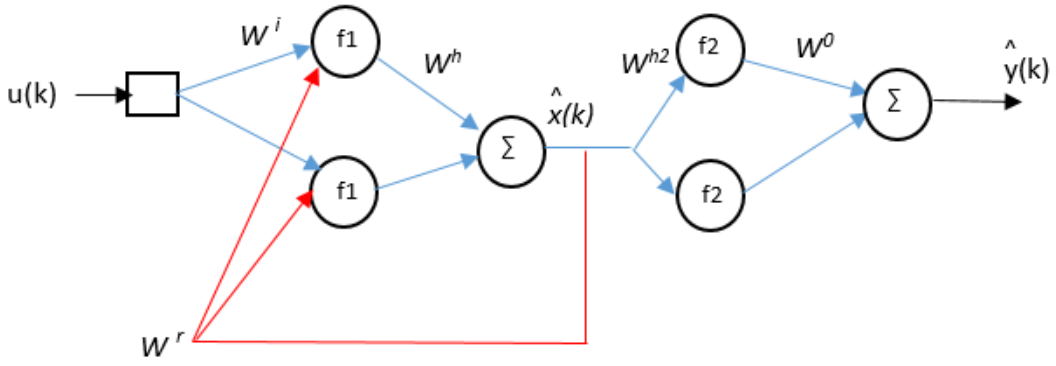

FIGURE III.1 – Architecture du SSNN

La représentation d'état qui peut être obtenue à partir de cette structure neuronale est la suivante :

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

Avec :

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

Pour l'apprentissage, les variables d'état et de sortie sont obtenues à partir du modèle mathématique du système. Pour cela, une séquence de commande est utilisée, composée de créneaux de durée et d'amplitude aléatoires, totalisant 400 pas d'échantillonnage, comme le montre la Figure (III.2).

Afin d'évaluer les performances du SSNN, une séquence de commandes de même type est utilisée, c'est-à-dire des créneaux d'amplitude et de durée aléatoires, mais cette fois-ci, elle couvre 300 pas d'échantillonnage, comme illustré sur la Figure (III.3).

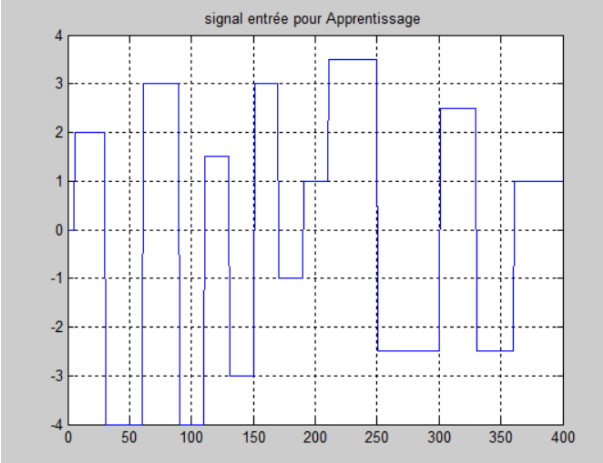

FIGURE III.2 – Séquence d'apprentissage

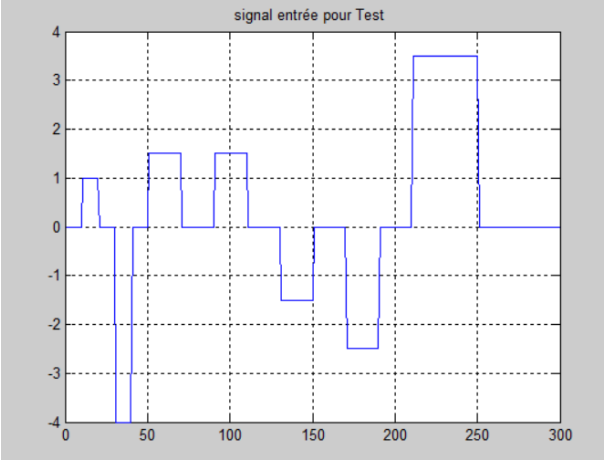

FIGURE III.3 – Séquence de test

Les résultats d'apprentissage et de test obtenus en utilisant l'algorithme de Levenberg-Marquardt en mode hors ligne sont présentés sur les Figures (III.4) et (III.5) respectivement, tandis que les Figures (III.6) et (III.7) illustrent les résultats obtenus avec l'algorithme du

Gradient en mode hors ligne.

Afin de quantifier l'erreur d'estimation, nous avons utilisé l'erreur quadratique moyenne (EQM), Ces erreurs sont regroupées dans la Table (III.1).

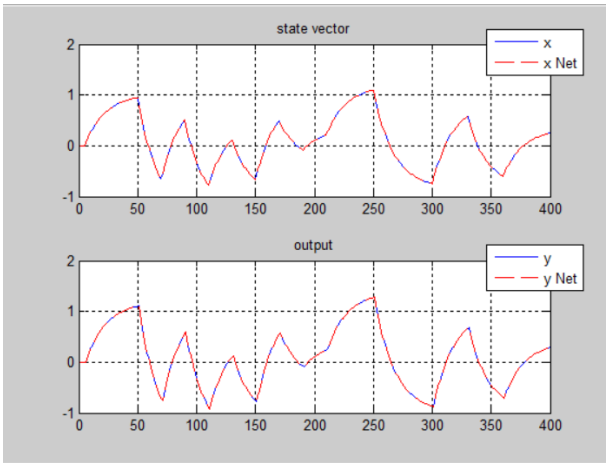

FIGURE III.4 – Séquence d'apprentissage des résultats obtenus avec la méthode de L-M

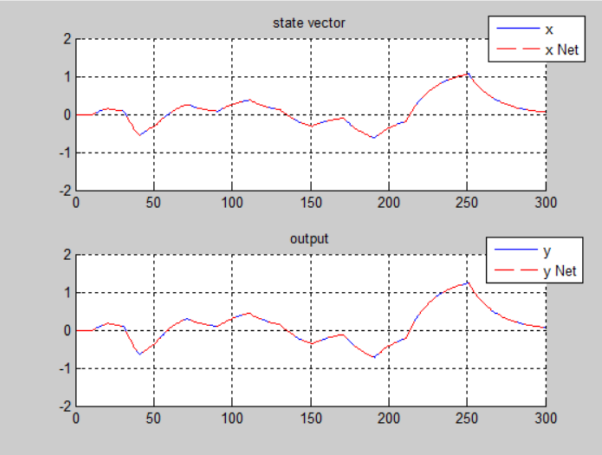

FIGURE III.5 – Séquence de test des résultats obtenus avec la méthode de L-M

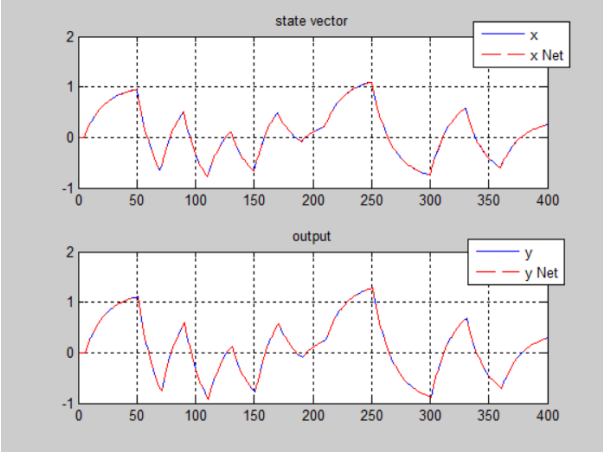

FIGURE III.6 – Séquence d'apprentissage des résultats obtenus avec la méthode du Gradient

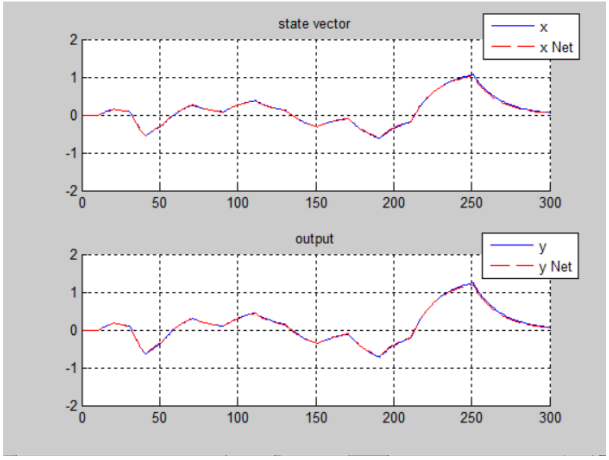

FIGURE III.7 – Séquence de test des résultats obtenus avec la méthode du Gradient

	EQMAx	EQMAy	EQMTx	EQMTy
Levenberg-Marquardt	4.3335×10^{-27}	5.0517×10^{-18}	2.4914×10^{-24}	3.5170×10^{-16}
Gradient	2.1504×10^{-6}	2.1875×10^{-11}	2.4688×10^{-6}	3.3870×10^{-6}

TABLE III.1 – Erreurs quadratiques moyennes entre l'état et sortie réels et l'état et sortie estimés par le SSNN

Avec :

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

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

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

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

Les Figures (III.4), (III.5), (III.6), (III.7) et la Table (III.1) révèlent que l'algorithme de Levenberg-Marquardt brille par sa performance exceptionnelle et avec son efficacité remarquable. En effet, avec seulement deux neurones dans les couches cachées et après 3 itérations seulement, il atteint des performances exceptionnelles. Les valeurs de l'EQMTx et de l'EQMTy s'établissent autour de 10^{-24} et 10^{-16} respectivement, bien inférieures aux erreurs quadratiques moyennes de 10^{-6} obtenues par la méthode du Gradient stochastique.

Les matrices des poids résultant du processus d'apprentissage sont les suivantes :

$$W^i = \begin{bmatrix} 0.4215 \\ 0.7320 \end{bmatrix} \qquad W^r = \begin{bmatrix} 0.6948 \\ -0.4292 \end{bmatrix} \qquad W^h = \begin{bmatrix} 1.0131 & -0.5568 \end{bmatrix}$$

$$W^{h2} = \begin{bmatrix} 0.7258 \\ -0.9879 \end{bmatrix} \qquad W^0 = \begin{bmatrix} 0.6290 & -0.7222 \end{bmatrix}$$

La représentation d'état du modèle neuronal, établie à partir des valeurs numériques issues de l'apprentissage est :

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

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

Les équations (III.1) et (III.3) montrent que les valeurs numériques des matrices (Φ, Γ et Ψ) obtenues avec le réseau de neurones sont identiques aux valeurs numériques des matrices (A, B et C) de la représentation d'état du système à identifier.

III.2.2 Identification d'un système linéaire multivariable d'ordre 1

Examinons un système linéaire de premier ordre dont la représentation d'état se présente comme suit :

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) \end{cases} \tag{III.4}$$

Avec :

$$A = 0.972 \qquad B = \begin{pmatrix} 0.5 & 0.1 \end{pmatrix} \qquad C = \begin{pmatrix} 0.32 \\ 0.29 \end{pmatrix}$$

Le réseau neuronal SSNN se caractérise par une structure spécifique : deux entrées de commande, suivie de trois neurones dans la première couche cachée, un neurone d'état, trois neurones supplémentaires dans la seconde couche cachée, et enfin deux neurones de sortie. Cette architecture est résumée par la séquence (2,3,1,3,2). La Figure (III.8) offre une représentation visuelle de cette architecture.

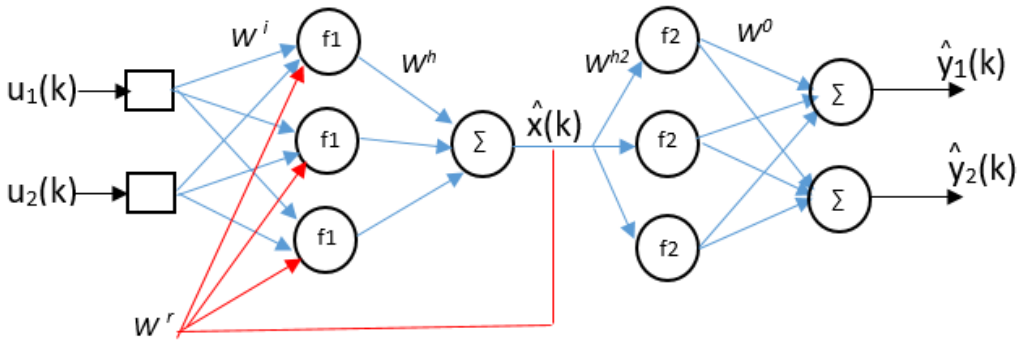

FIGURE III.8 – Architecture du SSNN

La représentation d'état déductible de cette structure neuronale se présente ainsi :

$$\begin{cases} \hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) \\ \hat{y}(k) = \Psi\hat{x}(k) \end{cases} \tag{III.5}$$

Avec :

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

Pour l'apprentissage, les variables d'état et de sortie sont obtenues à partir du modèle mathématique du système. À cet effet, deux signaux de commande $u_1(k)$ et $u_2(k)$ ont été utilisés : $u_1(k)$ est une suite de créneaux précédemment employée (signal 1), tandis que $u_2(k)$ est un signal sinusoïdal. La séquence d'apprentissage se compose de 400 pas d'échantillonnage, comme illustré à la Figure (III.9).

Pour évaluer les performances du SSNN, une nouvelle séquence de test a été mise en place. Celle-ci utilise également deux signaux de commande : un échelon d'amplitude 0.5 et un nouveau signal sinusoïdal différent de celui utilisée lors de l'apprentissage. Cette séquence de test couvre aussi 300 pas d'échantillonnage, tel que montré à la Figure (III.10).

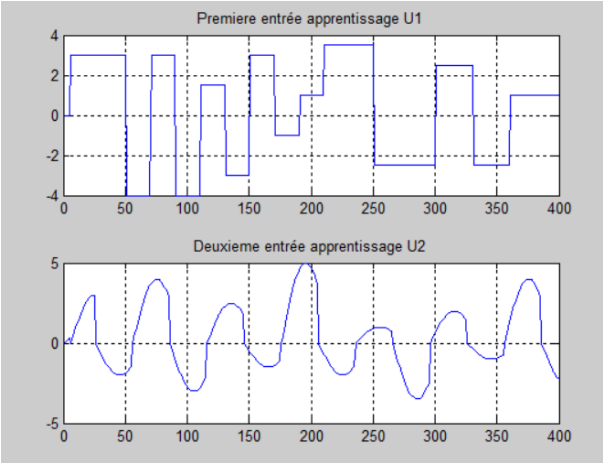

FIGURE III.9 – Séquence d'apprentissage

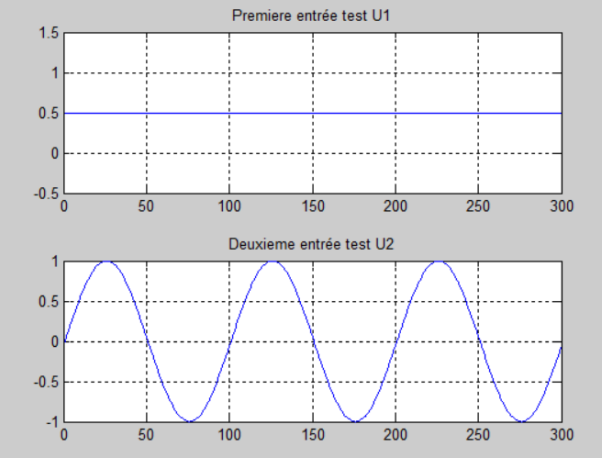

FIGURE III.10 – Séquence de test

Les résultats d'apprentissage et de test sont présentés dans les Figures (III.11) et (III.12), respectivement, en utilisant l'algorithme de Levenberg-Marquardt pour l'apprentissage hors ligne. Les Figures (III.13) et (III.14) illustrent également les résultats d'apprentissage et de test, mais cette fois-ci avec l'algorithme du Gradient.

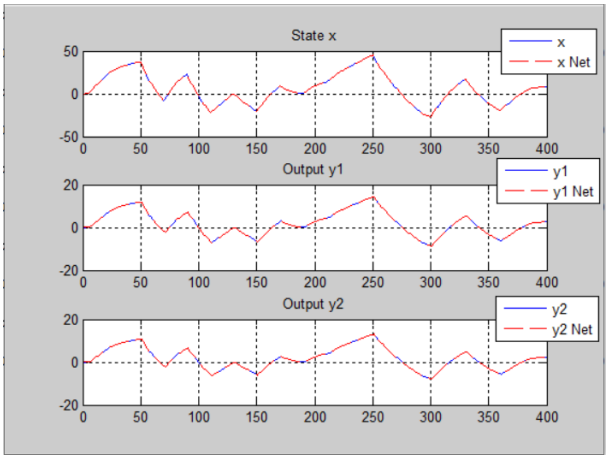

FIGURE III.11 – Séquence d'apprentissage des résultats obtenus avec la méthode de L-M

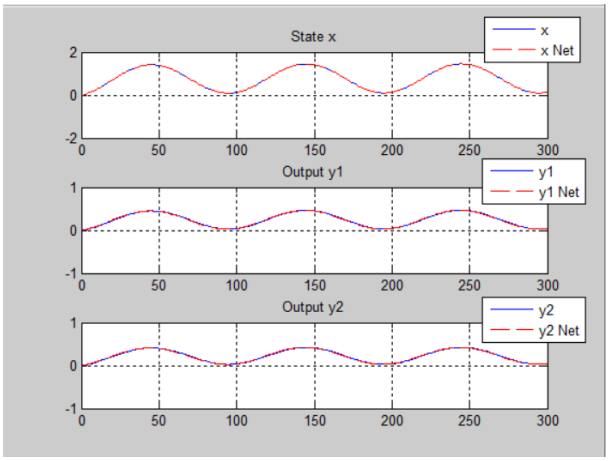

FIGURE III.12 – Séquence de test des résultats obtenus avec la méthode de L-M

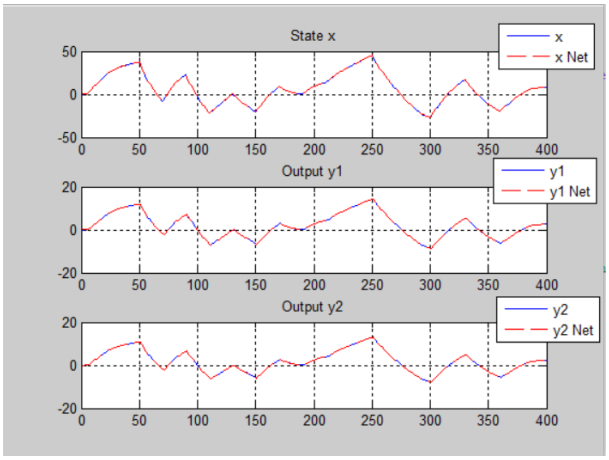

FIGURE III.13 – Séquence d'apprentissage des résultats obtenus avec la méthode du Gradient

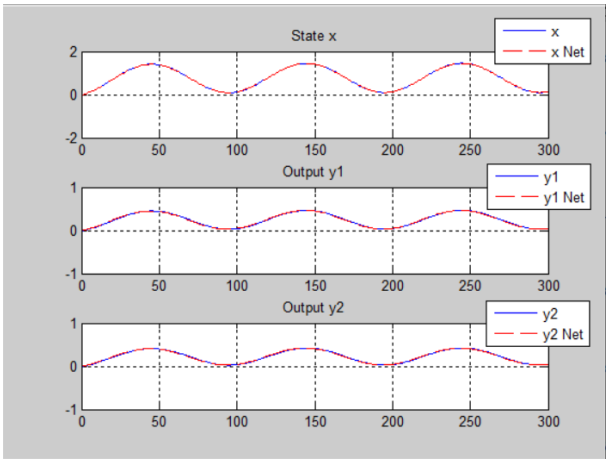

FIGURE III.14 – Séquence de test des résultats obtenus avec la méthode du Gradient

Les Tables (III.2) et (III.3) montrent respectivement les résultats des erreurs quadratiques moyennes (EQM) sur l'ensemble d'apprentissage et de test.

	EQMAx	EQMAy_1	EQMAy_2
Levenberg-Marquardt	3.0216×10^{-29}	2.0501×10^{-15}	1.3727×10^{-15}
Gradient	4.1922×10^{-3}	6.6499×10^{-6}	4.052×10^{-6}

TABLE III.2 – Comparaison des erreurs quadratiques moyennes sur l'ensemble d'apprentissage (EQMA) pour les méthodes de Levenberg-Marquardt et du Gradient

	EQMTx	EQMTy_1	EQMTy_2
Levenberg-Marquardt	1.0075×10^{-25}	9.5023×10^{-7}	7.8042×10^{-7}
Gradient	1.4615×10^{-3}	2.4553×10^{-3}	2.0119×10^{-3}

TABLE III.3 – Comparaison des erreurs quadratiques moyennes sur l'ensemble de test (EQMT) pour les méthodes de Levenberg-Marquardt et du Gradient

Avec :

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

EQMAy_1 : Évalue l'erreur quadratique moyenne de la première sortie y_1 sur l'ensemble d'apprentissage.

EQMAy_2 : Évalue l'erreur quadratique moyenne de la deuxième sortie y_2 sur l'ensemble d'apprentissage.

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

EQMTy_1 : Calcule l'erreur quadratique moyenne de la première sortie y_1 sur l'ensemble de test.

EQMTy_2 : Calcule l'erreur quadratique moyenne de la deuxième sortie y_2 sur l'ensemble de test.

Les résultats illustres par les Figures (III.11), (III.12), (III.13), (III.14) ainsi que les données des Tables (III.2) et (III.3) démontrent l'efficacité de l'algorithme de Levenberg-Marquardt dans l'identification des systèmes linéaires multivariables. Les erreurs quadratiques moyennes observées lors des tests sont de l'ordre de 10^{-25} pour l'état et de 10^{-7} pour les sorties, comparativement à à 10^{-3} pour l'état et de 10^{-3} pour les sorties avec la méthode du Gradient stochastique.

Le modèle a été entraîné en utilisant 4 itérations, avec 3 neurones dans les couches cachées.

Voici les matrices de poids obtenues à la suite du processus d'apprentissage :

$$W^i = \begin{bmatrix} -0.4939 & 0.6184 \\ 0.4335 & -1.1175 \\ -0.6829 & -0.6520 \end{bmatrix} \qquad W^r = \begin{bmatrix} -1.2056 \\ 0.4162 \\ -0.2379 \end{bmatrix} \qquad W^h = \begin{bmatrix} -0.8670 & -0.3707 & -0.3404 \end{bmatrix}$$

$$W^{h2} = \begin{bmatrix} -0.1858 \\ 0.1226 \\ 0.8780 \end{bmatrix} \qquad W^0 = \begin{bmatrix} -0.9355 & 0.5937 & 0.0836 \\ 0.6123 & -0.9156 & 0.5877 \end{bmatrix}$$

La représentation d'état du modèle neuronal, construite à partir des valeurs numériques obtenues lors de l'apprentissage, est la suivante :

$$\Phi = W^h * W^r = 0.972 \quad \Gamma = W^h * W^i = \begin{pmatrix} 0.5 & 0.1 \end{pmatrix} \quad \Psi = W^0 * W^{h2} = \begin{pmatrix} 0.32 \\ 0.29 \end{pmatrix}$$

$$\begin{cases} x(k+1) = 0.972x(k) + \begin{pmatrix} 0.5 & 0.1 \end{pmatrix} u(k) \\ y(k) = \begin{pmatrix} 0.32 \\ 0.29 \end{pmatrix} x(k) \end{cases} \tag{III.6}$$

Les résultats observés montrent que le SSNN, avec ses poids optimaux, peut parfaitement reproduire la représentation d'état d'un système linéaire multivariable, comme illustré par les comparaisons des représentations (III-4) et (III-6).

L'étude des systèmes linéaires à une ou plusieurs entrées et sorties montre que la méthode de Levenberg-Marquardt (L-M) offre des résultats très satisfaisants pour les systèmes monovariables. Bien que les résultats obtenus avec la méthode du Gradient soient également acceptables, ils sont plus sensibles au nombre d'entrées, ce qui augmente les erreurs d'estimation. En revanche, la méthode L-M maintient de très bonnes performances, la rendant ainsi robuste pour l'identification de systèmes linéaires complexes.

III.3 Identification de systèmes non linéaires

III.3.1 Identification d'un système non linéaire d'ordre 2

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

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Ctanh(x(k)) \end{cases} \tag{III.7}$$

Avec :

$$A = \begin{pmatrix} 0.8 & 0.1 \\ 0.1 & 0.7 \end{pmatrix} \qquad B = \begin{pmatrix} 0.005 \\ 0.008 \end{pmatrix} \qquad C = \begin{pmatrix} 1.2 & 0.8 \end{pmatrix}$$

Le réseau de neurones SSNN se distingue par sa structure : il commence par une seule entrée de commande, suivies par quatre neurones dans la première couche cachée. Ensuite, il intègre deux neurones d'état, puis quatre autres neurones dans la deuxième couche cachée, et se termine par un neurone de sortie. La séquence (1,4,2,4,1) résume cette configuration. La Figure (III.15) montre une représentation visuelle de cette architecture.

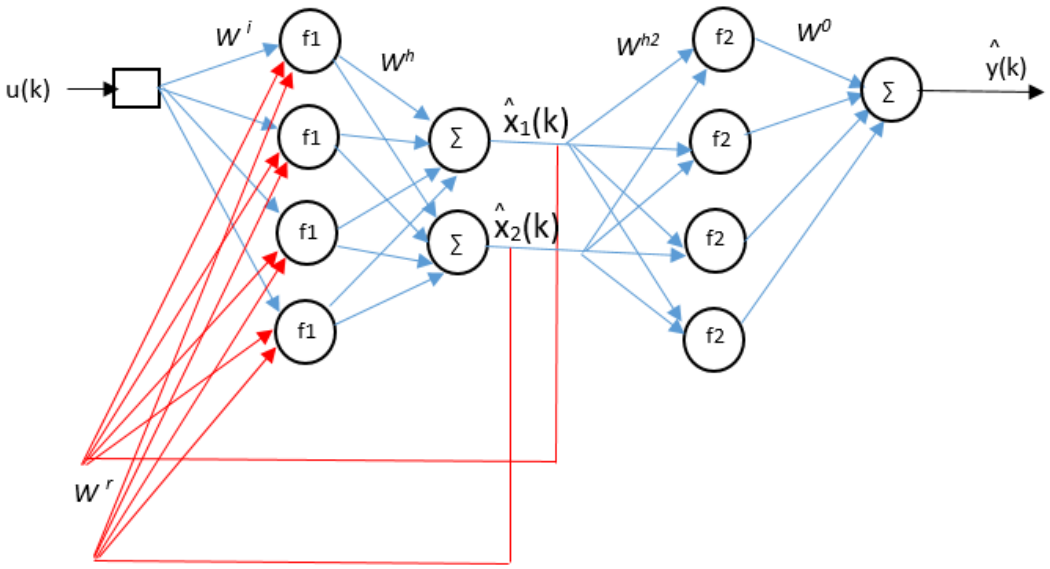

FIGURE III.15 – Architecture du SSNN

Voici la représentation d'état dérivée de cette architecture neuronale :

$$\begin{cases} \hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) \\ \hat{y}(k) = \Psi\hat{x}(k) \end{cases} \tag{III.8}$$

Avec :

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

Les séquences des variables d'état et des sorties employées pour l'apprentissage sont générées à partir du modèle mathématique du système.

La séquence de commande utilisée pour l'apprentissage est composée de créneaux d'amplitudes et de durées aléatoires. Cette séquence comporte 400 pas d'échantillonnage et est illustrée à la Figure (III.16).

La séquence de commande utilisée pour évaluer la performance du SSNN (séquence de test) est également constituée de créneaux d'amplitudes et de durées aléatoires. Cette séquence comporte 300 pas d'échantillonnage et est illustrée à la Figure (III.17).

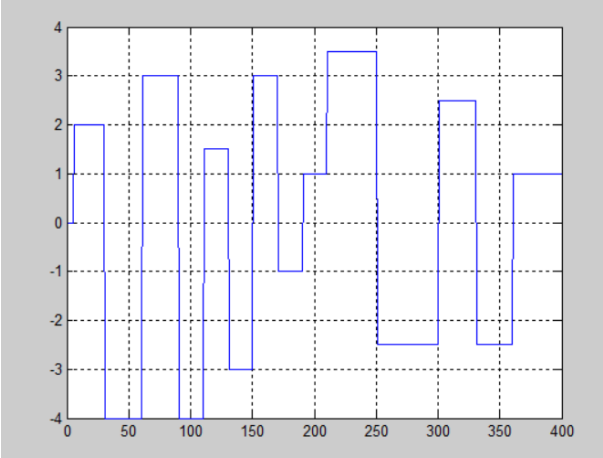

FIGURE III.16 – Séquence d'apprentissage

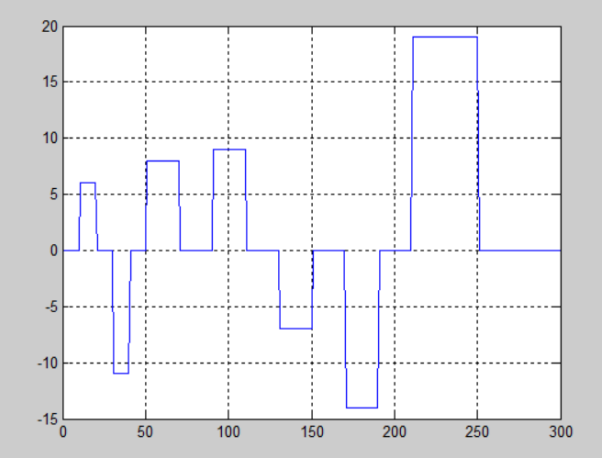

FIGURE III.17 – Séquence de test

Les résultats de l'apprentissage et des tests sont illustrés dans les Figures (III.18) et (III.19), respectivement, en utilisant l'algorithme de Levenberg-Marquardt pour l'apprentissage hors ligne. Les Figures (III.20) et (III.21) montrent également les résultats de l'apprentissage et des tests, mais cette fois avec l'algorithme du Gradient.

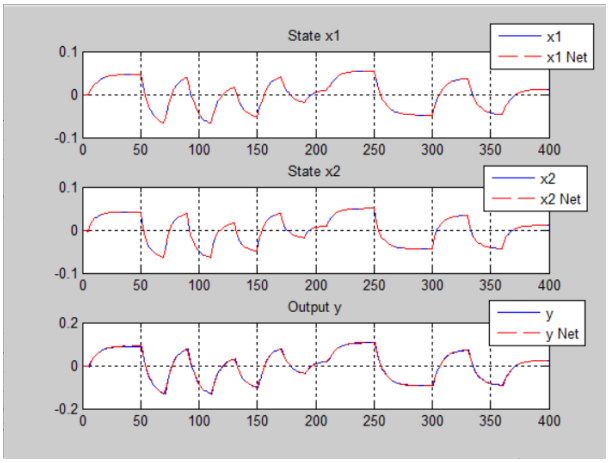

FIGURE III.18 – Séquence d'apprentissage des résultats obtenus avec la méthode de L-M

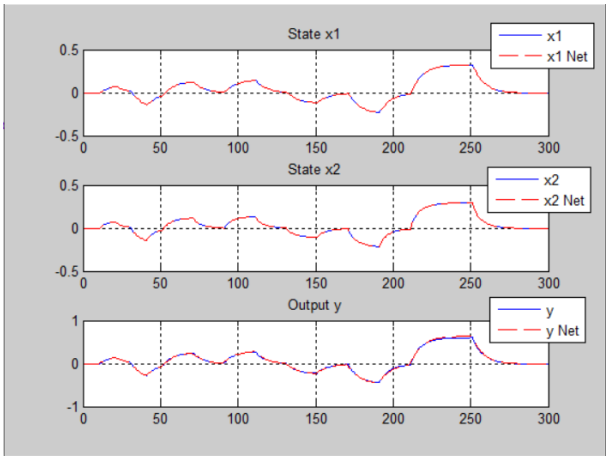

FIGURE III.19 – Séquence de test des résultats obtenus avec la méthode de L-M

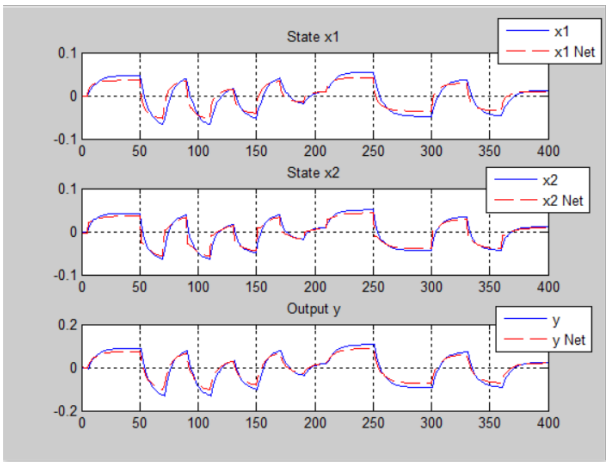

FIGURE III.20 – Séquence d'apprentissage des résultats obtenus avec la méthode du Gradient

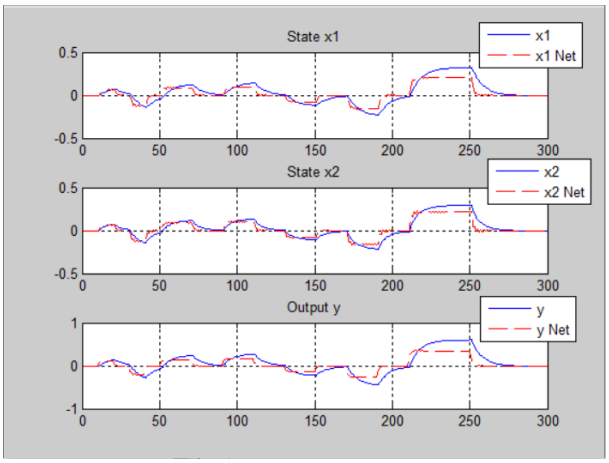

FIGURE III.21 – Séquence de test des résultats obtenus avec la méthode du Gradient

Les Tables (III.4) et (III.5) présentent respectivement les résultats des erreurs quadratiques moyennes (EQM) sur les ensembles d'apprentissage et de test.

	EQMAx_1	EQMAx_2	EQMAy
Levenberg-Marquardt	1.6528×10^{-5}	2.0713×10^{-5}	1.0096×10^{-5}
Gradient	2.4186×10^{-3}	1.7924×10^{-3}	2.7168×10^{-4}

TABLE III.4 – Comparaison des erreurs quadratiques moyennes sur l'ensemble d'apprentissage (EQMA) pour les méthodes de Levenberg-Marquardt et du Gradient

	EQMTx_1	EQMTx_2	EQMTy
Levenberg-Marquardt	2.3444×10^{-11}	7.1745×10^{-11}	1.3614×10^{-5}
Gradient	3.6646×10^{-4}	2.0179×10^{-4}	6.7526×10^{-4}

TABLE III.5 – Comparaison des erreurs quadratiques moyennes sur l'ensemble de test (EQMT) pour les méthodes de Levenberg-Marquardt et du Gradient

Avec :

EQMAx_1 : Représente la mesure de l'erreur quadratique moyenne du premier état x_1 sur l'ensemble d'apprentissage. EQMAx_2 : Évalue l'erreur quadratique moyenne du deuxième état x_2 sur l'ensemble d'apprentissage.

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

EQMTx_1 : Quantifie l'erreur quadratique moyenne du premier état x_1 sur l'ensemble de test.

EQMTx_2 : Calcule l'erreur quadratique moyenne du deuxième état x_2 sur l'ensemble de test.

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

Encore une fois, l'algorithme de Levenberg-Marquardt démontre son efficacité dans l'identification des systèmes. Les résultats, présentés dans les Figures (III.18) à (III.21) ainsi que dans les Tables (III.4) et (III.5), le confirment. Les erreurs quadratiques moyennes observées lors des tests sont de l'ordre de 10^{-11} pour l'état et de 10^{-5} pour la sortie.

Voici les matrices de poids obtenues à la suite du processus d'apprentissage :

$$W^i = \begin{bmatrix} -0.0138 \\ -0.1511 \\ -0.0848 \\ -0.2548 \end{bmatrix} \qquad W^0 = \begin{bmatrix} -0.3259 & 0.4524 & -1.2468 & -0.7703 \end{bmatrix}$$

$$W^h = \begin{bmatrix} 0.5028 & -0.4589 & -0.6783 & 0.4510 \\ 0.0804 & -0.3542 & 0.3087 & 0.0716 \end{bmatrix} \qquad W^r = \begin{bmatrix} 0.6132 & 0.1668 \\ -0.3351 & -1.2801 \\ -0.2922 & 0.7780 \\ 0.3098 & -0.0968 \end{bmatrix}$$

$$W^{h2} = \begin{bmatrix} -0.6249 & -0.4895 \\ 0.6921 & 0.4304 \\ -1.2217 & 0.5369 \\ -0.6452 & 0.3983 \end{bmatrix}$$

La représentation d'état du modèle neuronal, développée à partir des valeurs numériques acquises durant l'apprentissage, se présente comme suit :

$$\Phi = W^h * W^r = \begin{pmatrix} 0.8 & 0.1 \\ 0.1 & 0.7 \end{pmatrix} \quad \Gamma = W^h * W^i = \begin{pmatrix} 0.005 \\ 0.008 \end{pmatrix} \quad \Psi = W^0 * W^{h2} = \begin{pmatrix} 1.2 & 0.8 \end{pmatrix}$$

$$\begin{cases} x(k+1) = \begin{pmatrix} 0.8 & 0.1 \\ 0.1 & 0.7 \end{pmatrix} x(k) + \begin{pmatrix} 0.005 \\ 0.008 \end{pmatrix} u(k) \\ y(k) = \begin{pmatrix} 1.2 & 0.8 \end{pmatrix} tanh(x(k)) \end{cases} \tag{III.9}$$

Les équations (III.7) et (III.9) révèlent que les valeurs numériques des matrices (Φ, Γ et Ψ) calculées à l'aide du réseau de neurones sont équivalentes à celles des matrices (A, B et C) utilisées dans la modélisation d'état du système à identifier.

III.3.2 Identification d'un système non linéaire d'ordre 4

Considérons un système non linéaire de quatrième ordre, représenté par l'état suivant :

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Ctanh(x(k)) \end{cases} \tag{III.10}$$

Avec :

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ psinx_1(k) + p & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & p & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad C = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\text{et :} \quad p = 0.85$$

Remarques :

Le paramètre p est constant et induit une non linéarité. Si $p = 0$, le système est linéaire. Si $p \geq 1$, le système devient instable.

Le SSNN utilise deux réseaux de neurones, le premier pour estimer les états $x_1(k)$, $x_2(k)$, $x_3(k)$, $x_4(k)$ et le deuxième pour estimer les sorties $y_1(k)$, $y_2(k)$, $y_3(k)$, $y_4(k)$ (Figure (III.22)).

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 réseau utilisent une autre fonction d'activation $f_2(.) = tansig(.)$.

La Figure (III.22) montre l'architecture du SSNN (1,10,4,10,4) utilisée pour la modélisation du système.

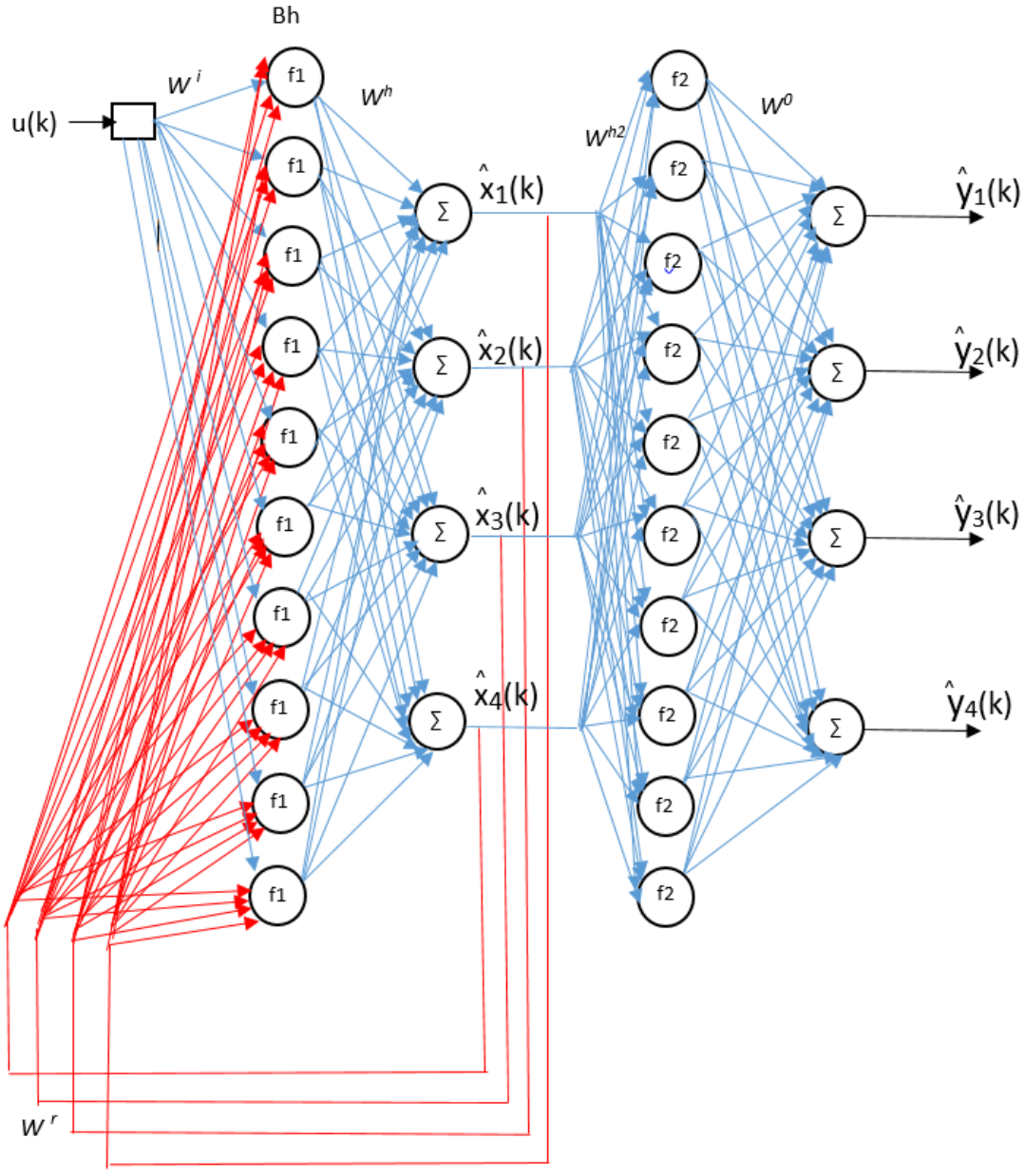

FIGURE III.22 – Architecture du SSNN

Le système est simulé avec des conditions initiales nulles, c'est-à-dire que :

$$\mathbf{x}(0) = [0 \quad 0 \quad 0 \quad 0]^T$$

Le système est excité par un signal de commande $u(k)$ sinusoïdal montré sur la Figure (III.23).

Les performances d'estimation des états et des sorties du système par un SSNN sont évaluées en comparant les réponses à un échelon unitaire qui est représenté dans la Figure (III.24).

La base d'apprentissage comprend 1800 mesures, et la condition d'arrêt de l'apprentissage est définie à 500 itérations.

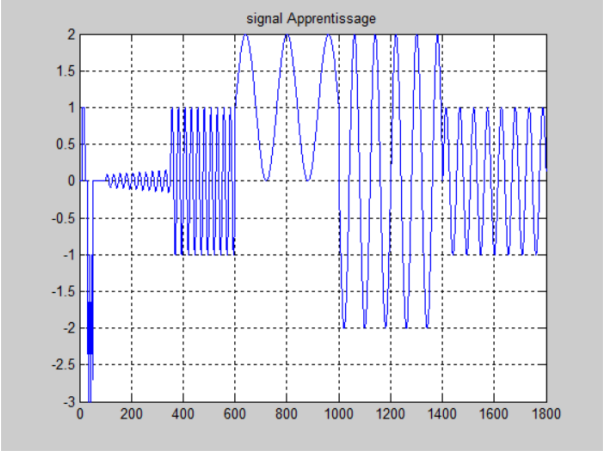

FIGURE III.23 – Séquence d'apprentissage

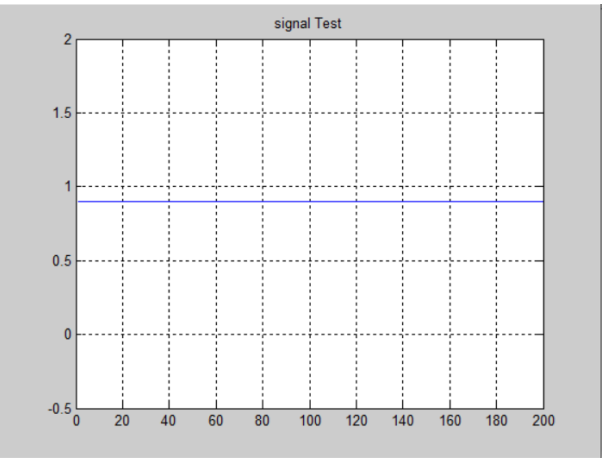

FIGURE III.24 – Séquence de test

Les Figures (III.25) et (III.26) illustrent les quatre états x_1, x_2, x_3, x_4 et les quatre sorties y_1, y_2, y_3, y_4 du système respectivement en utilisant l'algorithme de Levenberg-Marquardt pour l'apprentissage hors ligne. Ces figures présentent également les quatre états et les quatre sorties estimés par le SSNN.

Les Figure (III.27) et (III.28) illustrent les quatre états x_1, x_2, x_3, x_4 et les quatre sorties y_1, y_2, y_3, y_4 du système respectivement en utilisant l'algorithme du Gradient pour l'apprentissage hors ligne. Ces figures présentent également les quatre sorties estimés par le SSNN.

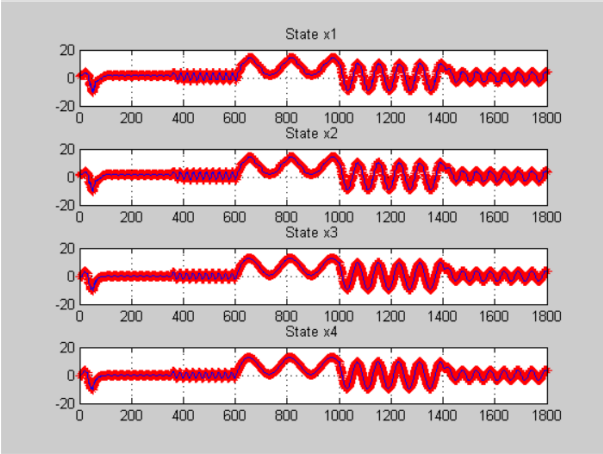

FIGURE III.25 – Résultats de l'apprentissage des états du SSNN avec la méthode de L-M (en bleu, les états réels (x_1, x_2, x_3, x_4) ; en rouge, les états estimés (x_1 Net, x_2 Net, x_3 Net, x_4 Net), respectivement présentés dans les graphes de haut en bas)

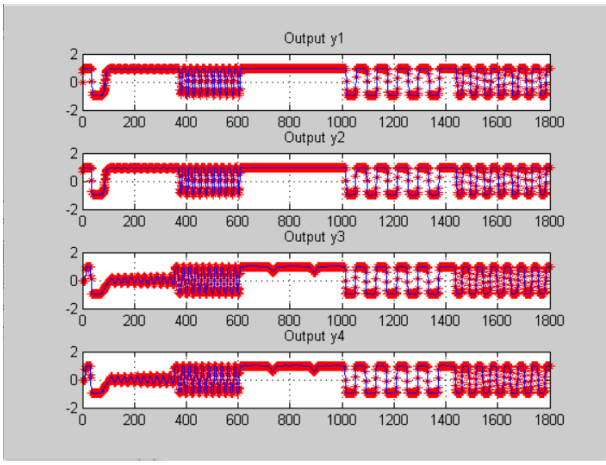

FIGURE III.26 – Résultats d'apprentissage des sorties du SSNN avec la méthode de L-M (en bleu, les sorties réelles (y_1, y_2, y_3, y_4) ; en rouge, les sorties estimées (y_1 Net, y_2 Net, y_3 Net, y_4 Net), respectivement présentés dans les graphes de haut en bas)

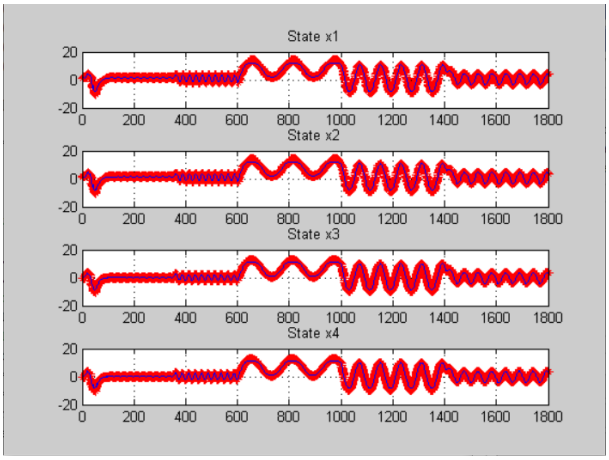

FIGURE III.27 – Résultats d'apprentissage des états du SSNN avec la méthode du Gradient (en bleu, les états réels (x_1, x_2, x_3, x_4) ; en rouge, les états estimés (x_1 Net, x_2 Net, x_3 Net, x_4 Net), respectivement présentés dans les graphes de haut en bas)

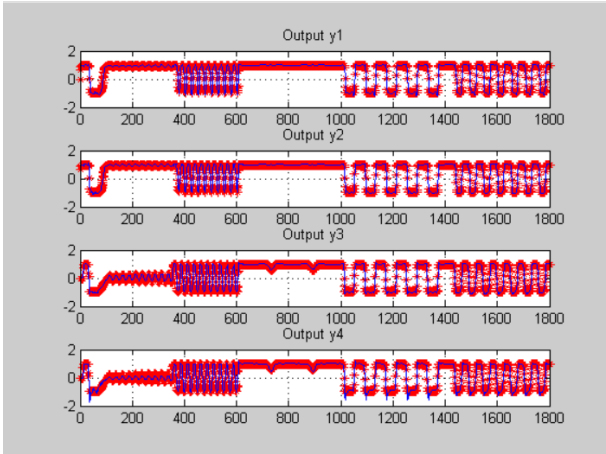

FIGURE III.28 – Résultats d'apprentissage des sorties du SSNN avec la méthode du Gradient (en bleu, les sorties réelles (y_1, y_2, y_3, y_4) ; en rouge, les sorties estimées (y_1 Net, y_2 Net, y_3 Net, y_4 Net), respectivement présentés dans les graphes de haut en bas)

Les Figures (III.29) et (III.30) illustrent les résultats du test pour les quatre états x_1, x_2, x_3, x_4 et les quatre sorties y_1, y_2, y_3, y_4 du système respectivement en utilisant

l'algorithme de Levenberg-Marquardt, lorsque le système est soumis à une excitation par un échelon d'amplitude 0.9. Ces figures présentent également les quatre états et les quatre sorties estimés par le SSNN.

Les Figures (III.31) et (III.32) illustrent les résultats du test pour les quatre états x_1, x_2, x_3, x_4 t les quatre sorties y_1, y_2, y_3, y_4 du système respectivement en utilisant l'algorithme du Gradient, lorsque le système est soumis à une excitation par un échelon d'amplitude 0.9. Ces figures présentent également les quatre états et les quatre sorties estimés par le SSNN.

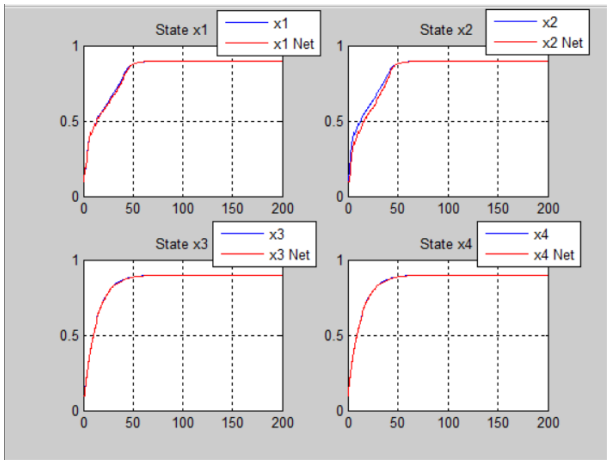

FIGURE III.29 – Réponses du SSNN à un signal échelon pour les états x_1, x_2, x_3, x_4 du système avec la méthode de L-M

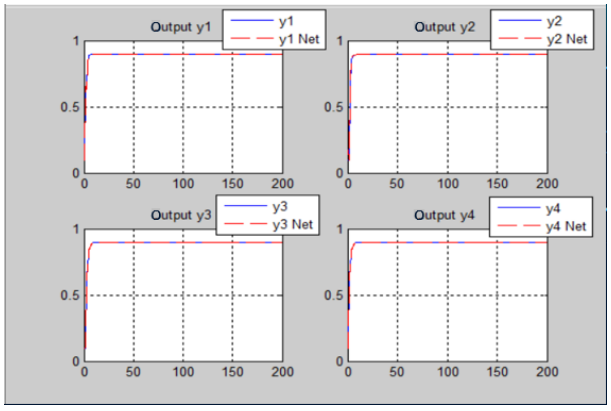

FIGURE III.30 – Réponses du SSNN à un signal échelon pour les sorties y_1, y_2, y_3, y_4 du système avec la méthode de L-M

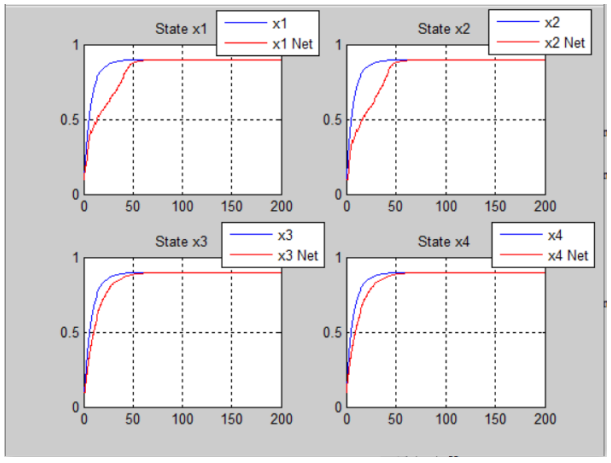

FIGURE III.31 – Réponses du SSNN à un signal échelon pour les états x_1, x_2, x_3, x_4 du système avec la méthode du Gradient

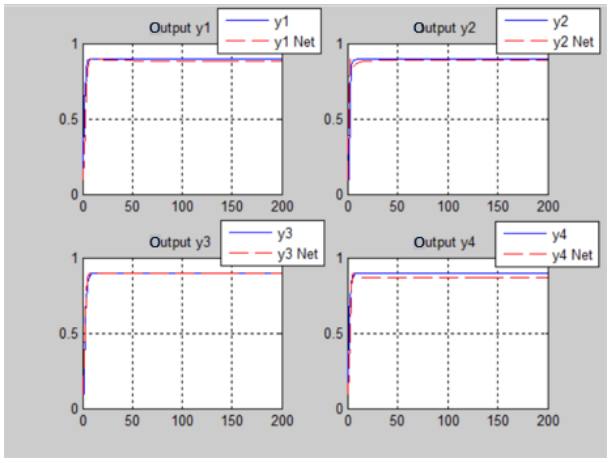

FIGURE III.32 – Réponses du SSNN à un signal échelon pour les sorties y_1, y_2, y_3, y_4 du système avec la méthode du Gradient

Les Tables (III.6) et (III.7) présentent les résultats des erreurs quadratiques moyennes (EQM) sur l'ensemble d'apprentissage, tandis que les Tables (III.8) et (III.9) présentent les résultats sur l'ensemble de test.

	EQMAx_1	EQMAx_2	EQMAx_3	EQMAx_4
L-M	2.093×10^{-1}	2.082×10^{-1}	1.354×10^{-1}	1.356×10^{-1}
Gradient	7.337×10^{-1}	5.740×10^{-1}	3.208×10^{-1}	3.080×10^{-1}

TABLE III.6 – Comparaison des erreurs quadratiques moyennes entre les états sur l'ensemble de l'apprentissage (EQMA) pour les méthodes de Levenberg-Marquardt et du Gradient

	EQMAy_1	EQMAy_2	EQMAy_3	EQMAy_4
L-M	1.5×10^{-6}	1.5×10^{-6}	8.838×10^{-7}	1.5×10^{-7}
Gradient	2.25×10^{-2}	1.82×10^{-2}	2.46×10^{-2}	3.03×10^{-2}

TABLE III.7 – Comparaison des erreurs quadratiques moyennes entre les sorties sur l'ensemble de l'apprentissage (EQMA) pour les méthodes de Levenberg-Marquardt et du Gradient

	EQMTx_1	EQMTx_2	EQMTx_3	EQMTx_4
L-M	2.44×10^{-6}	5.168×10^{-5}	1.2×10^{-7}	1×10^{-7}
Gradient	9.6×10^{-1}	1.24×10^{-1}	2.2×10^{-1}	2.3×10^{-1}

TABLE III.8 – Comparaison des erreurs quadratiques moyennes entre les états sur l'ensemble de test (EQMT) pour les méthodes de Levenberg-Marquardt et du Gradient

	EQMTy_1	EQMTy_2	EQMTy_3	EQMTy_4
L-M	3.01×10^{-6}	4.071×10^{-5}	1.6×10^{-5}	4×10^{-5}
Gradient	1.3×10^{-1}	2.1×10^{-1}	4×10^{-1}	2.2×10^{-1}

TABLE III.9 – Comparaison des erreurs quadratiques moyennes entre les sorties sur l'ensemble de test (EQMT) pour les méthodes de Levenberg-Marquardt et du Gradient

Avec :

EQMAx_i : Évalue la mesure de l'erreur quadratique moyenne du quatrième état x_i sur l'ensemble de l'apprentissage.

EQMAy_i : Évalue l'erreur quadratique moyenne de la sortie y_i sur l'ensemble de l'apprentissage.

EQMTx_i : Évalue l'erreur quadratique moyenne de l'état x_i sur l'ensemble de test.

EQMTy_i : Évalue l'erreur quadratique moyenne de la sortie y_i sur l'ensemble de test.

L'algorithme de Levenberg-Marquardt est plus efficace pour l'apprentissage et le test des états et sorties du système SSNN. Les résultats, présentés dans les Figures (III.25) à (III.32) ainsi que dans les Tables (III.6) à (III.9), le confirment.

Les Figures (III.33) et (III.34) montrent la différence entre les sorties y_1, y_2, y_3, y_4 et leurs estimés par le SSNN. On voit que la méthode de Levenberg-Marquardt (L-M) est plus performante que la méthode du Gradient pour le modèle SSNN (1,10,4,10,4) en termes de précision et de stabilité des sorties. Avec la méthode de Levenberg-Marquardt, le SSNN semble avoir réussi à reproduire presque parfaitement le comportement du système en régime permanent. Les graphiques des erreurs des sorties y_1, y_2, y_3 et y_4 dans la Figure (III.33) montrent que les erreurs convergent rapidement vers zéro et restent stables, indiquant une précision élevée du modèle une fois la phase transitoire passée. Cela suggère que le modèle a bien capturé la dynamique du système en régime permanent, avec des écarts très faibles entre les sorties réelles et estimées. L'erreur d'estimation a considérablement diminué en régime transitoire avec la méthode de Levenberg-Marquardt. Les graphiques montrent que les erreurs initiales, bien que présentes au début, convergent rapidement vers zéro, indiquant une réduction rapide de l'erreur d'estimation pendant la phase transitoire, ce qui permet au modèle de s'ajuster efficacement et de fournir des prédictions précises à long terme. Avec L-M, les erreurs initiales des sorties sont plus faibles, convergent rapidement vers zéro et présentent moins de fluctuations, ce qui indique une calibration plus efficace et une meilleure performance globale du modèle. En revanche, la méthode du Gradient montre des erreurs initiales des sorties plus élevées et des fluctuations plus importantes avant de se stabiliser, révélant une convergence plus lente et une précision initiale moindre.

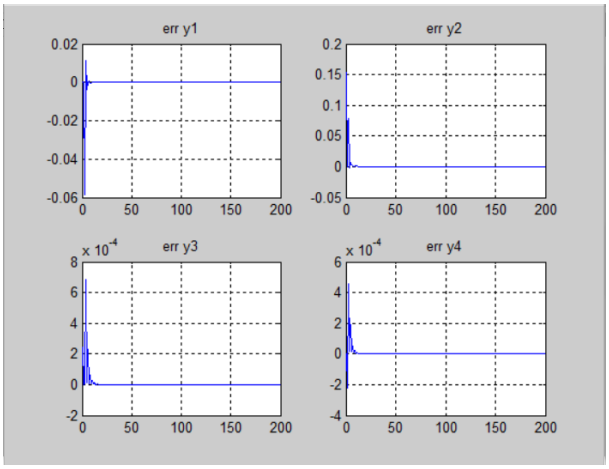

FIGURE III.33 – Différence entre les sorties réelles et estimées obtenues avec le SSNN(1,10,4,10,4) avec la méthode de L-M

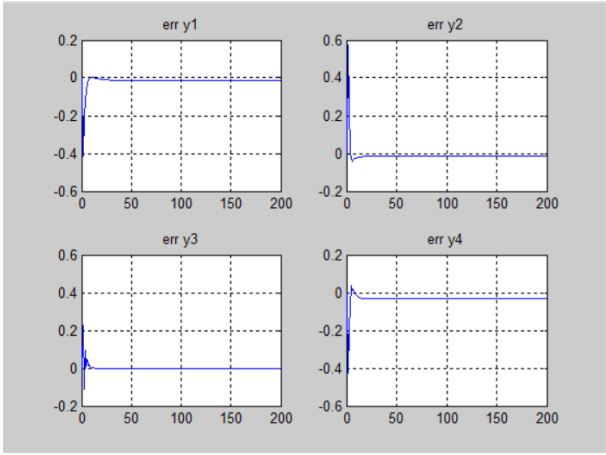

FIGURE III.34 – Différence entre les sorties réelles et estimées obtenues avec le SSNN(1,10,4,10,4) avec la méthode du Gradient

Voici les matrices de poids obtenues à la suite du processus d'apprentissage :

$$W^i = \begin{bmatrix} -0,0214 \\ -0,0230 \\ -0,0240 \\ 0,3006 \\ -0,0193 \\ -0,0043 \\ -0,3589 \\ -0,0224 \\ 0,0192 \\ 0,0182 \end{bmatrix} \qquad W^r = \begin{bmatrix} -0,4956 & -0,0242 & -0,0436 & 0,0496 \\ 0,0030 & -0,0604 & -0,0192 & 0,0854 \\ -0,9498 & 0,0156 & -0,1499 & 0,0860 \\ -0,0227 & 0,0798 & 0,2560 & -0,3390 \\ -0,4707 & -0,0176 & -0,0388 & 0,0429 \\ -0,2701 & 0,0082 & -0,0144 & 0,0092 \\ -0,3377 & -0,6623 & -0,4537 & 0,7951 \\ 0,0047 & -0,0444 & -0,0191 & 0,0370 \\ -0,1834 & 0,0703 & 0,0225 & -0,0488 \\ 0,4676 & 0,0172 & 0,0388 & -0,0423 \end{bmatrix}$$

$$W^h = \begin{bmatrix} 0,0626 & 28,272 & -0,0735 & -8,788 & 14,3907 & 1,2736 & -0,6336 & -149,8385 & -3,8681 & 14,6045 \\ 147.5615 & 77.6304 & -18.2380 & 9.2380 & 79.1182 & -692.7670 & -61.0550 & 302.2365 & 390.0939 & -101.7731 \\ -0,1883 & 92,6618 & -0,0284 & -2,7931 & 10,141 & 2,0383 & -0,4891 & -135,8467 & -3,3408 & 10,5839 \\ -0,3417 & 96,7390 & -0,0244 & 12,4343 & 20,3124 & 3,5234 & -0,4523 & -117,4711 & -5,9176 & 21,1234 \end{bmatrix}$$

$$W^{h2} = \begin{bmatrix} 0,1272 & -0,3398 & 0,3740 & -0,2001 \\ 0,0012 & 0,0001 & 0,9622 & 0,0063 \\ -0,0573 & 0,0397 & 0,9083 & -0,0616 \\ -0,0002 & 0,0001 & -0,9980 & -0,0003 \\ 0,0000 & 1,0000 & -0,0000 & 0,0000 \\ -0,0000 & 0,0000 & -0,0000 & -1,0000 \\ -1,0000 & -0,0000 & -0,0000 & 0,0000 \\ -0,0009 & 0,0006 & -0,9896 & -0,0015 \\ 0,9794 & -1,3232 & -0,7479 & 1,1856 \\ 0,0000 & -0,0000 & -0,0000 & 0,0000 \end{bmatrix} \qquad B^h = \begin{bmatrix} 1.2591 \\ 0,0551 \\ 12.3442 \\ -0,0102 \\ -1,1359 \\ -0,0349 \\ 13.1801 \\ 0,0137 \\ 0,0130 \\ 1.1185 \end{bmatrix}$$

$$W^0 = \begin{bmatrix} -0,0000 & 0,0000 & -0,0000 & -0,0000 & 0,0000 & 0,0000 & -1,0000 & 0,0000 & 0,0000 & -0,2986 \\ 0,0000 & -0,0000 & 0,0000 & 0,0000 & 1,0000 & -0,0000 & 0,0000 & -0,0000 & -0,0000 & 0,0579 \\ -0,0000 & -0,0000 & -00001 & -1,2460 & 0,0000 & 0,0000 & 0,0000 & 0,2459 & 0,0000 & 0,1401 \\ 0,0000 & -0,0000 & 0,0000 & 0,0000 & 0,0000 & -1,0000 & 0,0000 & -0,0000 & -0,0000 & 0,8321 \end{bmatrix}$$

$$B^{h_2} = \begin{bmatrix} 3.1088 \\ -0,3473 \\ 0,1851 \\ -0,0014 \\ -0,0000 \\ -0,0000 \\ -0,0000 \\ -0,0073 \\ 0,0707 \\ -2.6307 \end{bmatrix} \qquad B^1 = \begin{bmatrix} 52.7563 \\ -29.8945 \\ 13.0757 \\ -15.6153 \end{bmatrix} \qquad B^2 = \begin{bmatrix} -0,2955 \\ 0,0573 \\ 0,1387 \\ 0,8235 \end{bmatrix}$$

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

$$\begin{cases} \hat{x}(k+1) = W^h f_1(W^r \hat{x}(k) + W^i \vec{u}(k) + B^h) + B^1 \\ \hat{y}(k) = W^0 f_2(W^{h_2} \hat{x}(k) + B^{h_2}) + B^2 \end{cases} \tag{III.11}$$

et en choisissant les fonctions d'activation, elle devient :

$$\begin{cases} \hat{x}(k+1) = W^h logsig(W^r \hat{x}(k) + W^i \vec{u}(k) + B^h) + B^1 \\ \hat{y}(k) = W^0 tansig(W^{h_2} \hat{x}(k) + B^{h_2}) + B^2 \end{cases} \tag{III.12}$$

Notre étude a mis en évidence la performance constante de la méthode de Levenberg-Marquardt, quel que soit l'ordre du système, tout en comparant cette méthode à celle du Gradient. Cette fois, nous avons examiné deux cas de systèmes non linéaires : un système d'ordre deux et un autre d'ordre quatre avec quatre sorties.

Pour le système d'ordre deux, l'efficacité de l'approche L-M a été démontrée. Les résultats obtenus avec la méthode du Gradient sont acceptables malgré quelques limitations. Cependant, pour le système d'ordre quatre, la méthode de Levenberg-Marquardt a de nouveau prouvé son efficacité et sa constance. Elle a su gérer la complexité accrue de ce système, produisant des résultats précis et fiables. En comparaison, la méthode du Gradient a vu sa fiabilité diminuer, entraînant une augmentation de l'erreur d'estimation au-delà de l'ordre quatre.

III.4 Conclusion

Après cette étude et les résultats d'identification obtenus, il est clair que la méthode du Gradient n'est pas adaptée à l'identification des systèmes dynamiques fortement non linéaires avec un SSNN, notamment en régime transitoire où le SSNN peine à suivre la dynamique du système. En revanche, les performances obtenues avec la méthode de Levenberg-Marquardt demeurent très satisfaisantes malgré la forte non-linéarité du système et ne sont pas vraiment affectées par l'ordre du système.

Conclusion générale

Dans cette étude complète sur les réseaux de neurones à espace d'état, nous avons examiné une architecture qui combine les techniques d'apprentissage de l'intelligence artificielle avec la représentation d'état utilisée en automatique. Les SSNN sont particulièrement efficaces pour estimer et modéliser des processus dynamiques complexes, qu'ils soient linéaires ou non linéaires. Cette recherche a permis d'évaluer leurs capacités, avantages et limites dans ce contexte. Nous avons d'abord expliqué le fonctionnement et l'architecture des SSNN, puis montré comment les utiliser pour modéliser et identifier des processus dynamiques simulés. Deux méthodes d'apprentissage supervisé ont été appliquées pour modéliser différents systèmes dynamiques : la méthode du gradient stochastique et la méthode Levenberg-Marquardt. Le Gradient stochastique est une méthode de premier ordre qui ajuste les paramètres du réseau de manière efficace en fonction des données. Cependant, la méthode Levenberg-Marquardt s'est avérée supérieure. En tant que méthode de deuxième ordre, elle optimise la convergence du modèle avec une précision accrue et plus rapidement.

En conclusion, les SSNN sont très prometteurs pour modéliser des systèmes dynamiques complexes. La méthode Levenberg-Marquardt, en particulier, s'est révélée la plus efficace pour améliorer les performances des SSNN. Pour exploiter pleinement leur potentiel, il est essentiel de continuer à perfectionner les techniques d'apprentissage et d'optimisation des paramètres des SSNN.

Références bibliographiques

[1] McCulloch, W. S., & Pitts, W. H. (1943). A logical calculus of ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), 115-133.

[2] Rosenblatt, F. (1958). *The perceptron : A probabilistic model for information storage and organization in the brain*. Psychological Review, 65(6), 386–408.

[3] Bengio, Y., Goodfellow, I., & Courville, A. (2016). *Deep learning*. Cambridge, MA : MIT Press.

[4] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533-536.

[5] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

[6] Russell, S. J., & Norvig, P. (2020). *Artificial intelligence : A modern approach* (4th edition). Pearson.

[7] Haykin, S. (2009). *Réseaux de neurones et machines apprenantes* (3e édition). Upper Saddle River, NJ : Prentice Hall.

[8] Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow*. O'Reilly Media.

[9] Kandel, E. R., Schwartz, J. H., Jessell, T. M., Siegelbaum, S. A., & Hudspeth, A. J. (2012). *Principles of neural science* (5th edition). New York : McGraw-Hill.

[10] Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press.

[11] Jordan, M. I. (1986). Serial order : A parallel distributed processing approach. Institute for Cognitive Science, University of California, San Diego.

[12] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179-210.

[13] Ljung, L. (1999). *System identification : Theory for the user* (2ème édition). Upper Saddle River, NJ : Prentice Hall.

[14] Dreyfus, G. (1998). *Neural Networks : Methodology and Applications*. Springer.

[15] Bengio, Y. (2018). *Apprendre à apprendre : Vers une intelligence artificielle générale*. Editions Odile Jacob.

[16] Boureau, Y., & Bach, F. (2019). *Apprentissage profond : fondements théoriques et applications*. Editions Odile Jacob.

[17] Mé, D., & Oudin, N. (2017). *Apprentissage automatique : théorie et applications*. Editions Eyrolles.

[18] Jaulin, L. (2005). *Analyse et commande des systèmes dynamiques*. Hermes Science Publications.

[19] Granjon, D. (2015). *Modélisation et commande des systèmes dynamiques*. Éditions Lavoisier.

[20] Rugh, W. J. (1996). *Linear System Theory* (2nd edition). Upper Saddle River, NJ : Prentice Hall.

[21] Van der Smagt, P. P. (2001). *Identification de systèmes dynamiques à l'aide de réseaux de neurones récurrents*. Amsterdam, Pays-Bas : IOS Press.

[22] Douglas, S. C., & Szu, H. H. (1993). *Nonlinear System Identification and Control*. John Wiley & Sons.

[23] Zamarreno, J.M., Vega, P., Garcia, L.D., & Francisco, M. (2000). State-space neural network for modelling, prediction and control. *Control Engineering Practice*, 8(9), 1063-1075.

[24] Jang, J. S., Sun, C. T., & Mizuo, Y. (1997). *Neuro-fuzzy et soft computing : Une approche computationnelle de l'apprentissage et de la prise de décision*. Englewood Cliffs, NJ : Prentice Hall.

[25] Mozer, M. C. (2000). Au-delà des gradients : Le défi de l'apprentissage de systèmes de contrôle complexes. In *Advances in neural information processing systems* (pp. 261-268).

[26] Bose, N. K., & Liang, P. (2007). *Neural Network Fundamentals with Graphs, Algorithms, and Applications*. Springer.

[27] Personnaz, L., Gluck, M. A., & Rumelhart, D. E. (2003). *Learning representations by back-propagating errors*. In *Neural Networks for Pattern Recognition*. Cambridge University Press.

[28] Mahul, A. (2005). *Apprentissage de la qualité de service dans les réseaux multiservices : application au routage optimal sous contraintes*. Thèse de Doctorat, Université de Sciences Pour l'Ingénieur de Clermont Ferrand.

[29] Gil, P., Henriques, J., Dourado, A., & Duarte-Ramos, H. (2005). Order estimation in affine state space neural networks. In *IEEE Mid-Summer Workshop on Soft Computing in Industrial applications*, Espoo, Finland, June 28-30.