

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

UNIVERSITE ABDERAHMANE MIRA -BEJAIA-

Faculté de la Technologie

Département d' Electronique

Mémoire

**En vue d'obtention du diplôme de Magister en
Automatique et Traitement du Signal**

Option : Système

Présenté par :

Zahir ASRADJ

Thème :

Identification des systèmes non linéaires par les réseaux de neurones

Soutenu le :

Jury

M. S. Radjef
D. Aissani
A. Oukaour
B. Mendil
K. Mokrani

Pr. Université de Béjaia
Pr. Université de Béjaia
MC. Université de Caen
Pr. Université de Béjaia
MC. Université de Béjaia

Président
Rapporteur
Co/Rapporteur
Examineur
Examineur

REMERCIEMENTS

Mes remerciements vont à, monsieur : D. Aissani, Professeur à l'université de Béjaia, pour avoir accepté d'être le rapporteur de ce travail, sa disponibilité et ses conseils.

Mes sincères remerciements vont aussi à, monsieur : A. Ouğaour, Maître de conférence à l'université de Caen, pour son aide, ses conseils précieux, ses encouragements incessants durant la réalisation de ce travail, car il s'est beaucoup investi pour que ce travail soit un succès espéré.

Je tiens à remercier également monsieur : M. S. Radjef, Professeur à l'université de Bejaia et messieurs : B. Mendil, Professeur à l'université de Bejaia, K. Mokrani, Maître de conférence à l'université de Bejaia, d'avoir accepté de juger ce travail.

Je tiens aussi à témoigner ma reconnaissance à toute personnes ayant aidé de près ou de loin à l'aboutissement de ce travail en particulier Aïda.

A ma famille

Sommaire

Introduction générale.....	01
-----------------------------------	-----------

Chapitre I : Etat de l'art sur les réseaux de neurones

I. Introduction.....	03
I.1 Réseaux de neurones artificiels.....	03
I.1.1 Historique.....	03
I.1.2 Neurone biologique.....	04
I.1.3 Neurone formel	05
I.1.4 Fonction d'activation	06
I.1.5 Réseaux de neurones.....	08
I.1.5.1 Réseaux de neurones non bouclés.....	08
a) Exemple de réseau MLP	09
I.1.5.2 Réseaux de neurones bouclés.....	10
a) Exemple de réseau de neurone à mémoire.....	10
b) Exemple de réseau neuronal récurrent à temps discret.....	11
I.2 Algorithmes d'apprentissage	12
I.2.1 Introduction.....	12
I.2.2 Méthodes d'optimisation	13
I.2.2.1 Algorithme de la rétro-propagation	13
I.2.2.2 Méthode de recherche aléatoire	14
I.2.2.3 Méthode de gradient conjugué.....	15
I.2.2.4 Algorithme génétique.....	15
I.5 Conclusion	15

Chapitre II: Les algorithmes génétiques

II. Introduction	16
II.1 Les algorithmes génétiques	16
II.2 Technologies et éléments de base.....	16
II.3 A quoi sert l'algorithme génétique ?	17
II.4 Conception d'un algorithme génétique.....	18
II.4.1 Codage des chromosomes	19
a) Codage binaire	19
b) Codage réel	20
II.4.2 Initialisation de la population	20

II.4.3 Fonction d'évaluation	21
II.4.3.1 No scaling	21
II.4.3.2 Echelle linéaire	21
II.4.3.3 Le sigma tronqué	22
II.4.3.4 Puissance d'échelle.....	22
II.4.3.5 Rangement	22
II.5 La sélection.....	23
II.5.1 Sélection ordonnée	23
II.5.2 Sélection par la roulette biaisée	23
II.5.3 Sélection par tournoi.....	24
II.5.4 Sélection uniforme.....	24
II.6 Croisement.....	25
II.6.1 Croisement simple	25
II.6.2 Croisement multiple	26
II.6.3 Croisement uniforme	26
II.6.4 Croisement arithmétique.....	27
II.6.5 Croisement BLX- α	27
II.6.6 Croisement linéaire.....	28
II.6.7 Croisement discret	28
II.6.8 Croisement étendu	28
II.7 Mutation.....	29
II.7.1 Mutation aléatoire.....	30
II.7.2 Mutation non uniforme	30
II.8 Réinsertion.....	31
II.9 Les paramètres de base d'un AG	31
II.9.1 Population.....	31
II.9.2 Taille de la population	31
II.9.3 Taux de croisement.....	32
II.9.4 Taux de mutation	32
II.9.5 Critère d'arrêt	32
II.10 Classification des données.....	32
II.10.1 Description de la méthode de classification utilisée.....	33
II.10.2 Apprentissage	34
a) Mise en œuvre de l'algorithme	34

b) Fonction objective.....	35
c) Les paramètres utilisés dans l'AG.....	35
II.10.3 Application avec le réseau MLP.....	36
a) Instruction de l'AG implémenté	36
a) La génération des données	37
II.11 Conclusion.....	40

Chapitre III: Identification par réseaux de neurones optimisés par les AG

III Introduction	41
III.1 Identification par réseaux de neurones.....	41
III.1.1 Identification	41
III.1.2 Type d'identification	41
III.1.2.1 Identification série-parallèle.....	41
III.1.2.2 Identification parallèle.....	42
III.2 Perceptron multicouche.....	43
III.2.1 Mise en œuvre du MLP.....	43
III.2.2 Propriétés fondamentales du MLP	44
III.2.3 Equations du réseau.....	44
III.2.4 Apprentissage du réseau MLP.....	45
III.2.5 Application du réseau MLP à l'identification.....	46
III.3 Réseau de neurones à mémoire (MNN).....	50
III.3.1 Equations du réseau.....	51
III.3.2 Apprentissage du réseau MNN	52
III.3.3 Algorithme MNN	54
III.3.4 Application du réseau MNN à l'identification.....	55
III.4 Réseau neuronal récurrent à temps discret DTRNN	57
III.4.1 Equations du réseau.....	58
III.4.2 Apprentissage du réseau DTRNN	59
III.4.3 Algorithme RTRL	60
III.4.4 Application du réseau DTRNN à l'identification	61
III.5 Identification avec les réseaux de neurones entraînés avec les AG.....	64
III.5.1 Réseau MLP.....	64
III.5.2 Réseau MNN	66
III.5.3 Réseau DTRNN.....	67
III.5.4 Commentaires.....	69

III.6 Conclusion.....	70
-----------------------	----

Chapitre IV: Application à la machine asynchrone

IV Introduction.....	71
IV.1 Description de la machine.....	71
IV.1.1 Constitution.....	71
IV.1.2 Principe de fonctionnement	72
IV.2 Modélisation de la machine	73
IV.2.1 Hypothèse simplificatrice	74
IV.2.2 Equations électriques	74
IV.2.3 Equations magnétiques	75
IV.2.4 Equations mécaniques	76
IV.3 Modèle biphasé de la machine	76
IV.3.1 Transformation de Park	76
IV.3.2 Transformation de Park appliquée à la MAS	77
IV.3.3 Expression du couple électromagnétique instantané	79
IV.3.4 Choix du référentiel	79
IV.4 Représentation d'état de la MAS	80
IV.5 Onduleur de tension à MLI	81
IV.5.1 Principe de la MLI sinus-triangle.....	81
IV.5.2 Modélisation de l'onduleur de tension à M.L.I.....	82
IV.5.3 Résultats de simulation	84
IV.6 Identification de l'onduleur-MAS.....	86
IV.7 Résultats de simulation	86
IV.7.1 Identification série-parallèle	87
IV.7.2 Identification parallèle	88
IV.8 Conclusion	89

Conclusion générale	90
----------------------------------	-----------

Bibliographie.....	91
---------------------------	-----------

INTRODUCTION GENERALE

Introduction générale

Au cours des vingt dernières années, les techniques de l'intelligence artificielle, telle que la logique floue, les algorithmes génétiques et les réseaux de neurones, ont été de plus en plus appliquées au contrôle et à l'identification des systèmes complexes. Ces derniers sont caractérisés par des non linéarités, une très grande dimension et une imprécision des modèles mathématiques [1], [2].

L'identification de ces systèmes est aujourd'hui un domaine de recherche d'intérêt majeur pour l'industrie. Il est souvent nécessaire de construire des modèles à partir de données réelles (entrées-sorties), capables d'apprendre la dynamique du système en question pour la mise au point des lois de commande. Cette démarche est réalisée en deux phases : d'abord la caractérisation (choix de la structure du modèle), puis l'estimation des paramètres [1].

Récemment, parmi les modèles utilisés pour l'identification, on trouve les réseaux de neurones, que ce soit statiques ou dynamiques. Un problème que nous devons relativement résoudre avant d'utiliser un réseau de neurones est la définition de sa structure. Cette dernière n'est souvent que partiellement imposée par la tâche à réaliser : les entrées et les sorties du réseau peuvent être fixées en fonction de celle-ci par les concepteurs [2]. Mais le nombre de couches cachées et le nombre optimal de neurones dans chaque couche ne peuvent être fixés a priori [2], [3], [4]. Pour résoudre ce problème, des techniques stochastiques sont proposées. Parmi celles-ci, les algorithmes génétiques (AG) qui font partie du champ de l'intelligence artificielle. Ce sont des algorithmes d'optimisation dérivés de la génétique naturelle et des mécanismes d'évolution (croisement, mutation et sélection) [5].

Holland [33], Man et Al [34] sont les premiers à avoir donné les principes de base des AG. Leurs applications aux problèmes d'optimisation ont été formulées par Goldberg [6] en 1989.

Les travaux [7], [8], [9], [10] ont montré que les AG peuvent être utilisés comme dispositif d'apprentissage de Perceptron multicouche (MLP). Les AG semblent donc être une bonne solution de rechange car elle permet de couvrir un espace de recherche important pour des problèmes à plusieurs minimums locaux.

Il existe différentes méthodes pour intégrer les AG dans les réseaux de neurones (RN). Ces méthodes touchent généralement soit la topologie du réseau (nombre de couches cachées

ou nombre de neurones), soit la phase d'apprentissage en adaptant certains paramètres propres aux unités du réseau ou bien de leurs connexions.

Globalement, l'intérêt de l'hybridation des RN avec les AG au niveau de l'apprentissage se base sur l'observation d'une recherche globale par une méthode de descente de gradient. Cette dernière sera bien complétée ou même remplacée par une recherche locale effectuée par les AG.

Dans le cadre de ce travail, on a opté pour une hybridation au niveau de l'apprentissage afin de minimiser l'effet de l'approximation sur l'adaptation des paramètres des RN. Nous appliquons l'AG sur les RN statiques et dynamiques. Cet algorithme sera comparé à d'autres algorithmes, à savoir : l'algorithme de la rétro-propagation (BP) et l'algorithme d'apprentissage en temps réel (RTRL).

Dans ce mémoire, l'identification des systèmes non linéaires consiste à réaliser un modèle neuronal qui définit le mieux le comportement du système à identifier. Le critère d'optimisation utilisé est l'erreur quadratique moyenne entre la sortie du système et la sortie du modèle des réseaux de neurones. Lors de l'application des algorithmes BP et RTRL, le résultat des réseaux de neurones dépend de l'initialisation des paramètres du départ, en plus la convergence du modèle n'est pas assuré [2], [3]. Alors, nous avons suggéré d'effectuer l'apprentissage en utilisant les AG.

Ce mémoire est organisé comme suit :

Le premier chapitre présente un état de l'art sur les trois types de réseaux de neurones utilisés, à savoir le perceptron multicouche (MLP), réseau de neurones à mémoire (MNN) et le réseau neuronal récurrent à temps discret (DTRNN). Nous commençons par la présentation des concepts fondamentaux au sujet des réseaux de neurones, une description de l'apprentissage et les méthodes d'optimisation les plus utilisées.

Dans le second chapitre, nous rappelons les notions de base sur les algorithmes génétiques, les différents opérateurs génétiques, à savoir le croisement, la sélection et la mutation. Et à la fin du chapitre, nous illustrons l'efficacité de l'AG mis en œuvre en les appliquant sur la classification des données.

Le troisième chapitre sera consacré à l'identification des systèmes non linéaires en utilisant les trois types de réseaux de neurones exposés auparavant, optimisés par les AG.

Le quatrième chapitre est une application de tous les algorithmes mis en œuvre pour l'identification du système onduleur machine asynchrone.

CHAPITRE I : ETAT DE L'ART SUR LES RESEAUX DE NEURONES

I Introduction :

Les réseaux de neurones (RN) formels sont des systèmes de traitement de l'information dont la structure s'inspire de celle du système nerveux. Leurs deux grands domaines d'application sont d'une part la modélisation biologique, dont il ne sera pas question ici, et d'autre part, la réalisation de machines destinées à effectuer des tâches auxquelles les ordinateurs et les outils traditionnels semblent moins bien adaptés que les êtres vivants, telles que des tâches perceptives et motrices, ainsi, qu'à la reconnaissance de formes, la classification et à l'identification des systèmes.

Nous commençons donc ce chapitre, en premier lieu, par l'historique des RN, nous présenterons ensuite les définitions essentielles, nous expliquons ce qu'est un neurone formel, ce qu'est un réseau de neurones, ce qu'est l'apprentissage des RN (nous précisons notamment les différences entre l'apprentissage supervisé et l'apprentissage non supervisé).

Nous terminerons par la présentation de quelques méthodes d'optimisation telles que la méthode du gradient conjugué et les Algorithmes génétiques.

I.1 Réseaux de neurones artificiels

I.1.1 Historique

Aujourd'hui, les réseaux de neurones sont utilisés dans de nombreux domaines à cause de leurs propriétés, en particulier leur capacité d'apprentissage.

- 1890 : W. James, célèbre psychologue américain introduit le concept de mémoire associative et propose ce qui deviendra une loi de fonctionnement pour l'apprentissage sur les réseaux de neurones connue plus tard sous le nom de loi de Hebb.
- 1943 : J. Mc Culloch et W. Pitts laissent leurs noms à une modélisation du neurone biologique (un neurone au comportement binaire). Ce sont les premiers à montrer que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes (tout au moins au niveau théorique).
- 1949 : D. Hebb, physiologiste américain explique le conditionnement chez l'animal par les propriétés des neurones eux-mêmes. Ainsi, un conditionnement de type pavlovien tel que, nourrir tous les jours à la même heure un chien, entraîne

chez cet animal la sécrétion de salive à cette heure précise même en l'absence de nourriture. La loi de modification des propriétés des connexions entre neurones qu'il propose explique en partie ce type de résultats expérimentaux [11].

Le premier réseau de neurones artificiel apparaît en 1958, grâce aux travaux de Rosenblatt qui conçoit le perceptron. Ce dernier est inspiré du système visuel (en terme d'architecture neurobiologique) et possède une couche de neurones d'entrée une couche de sortie ("décisionnelle"). Ce réseau parvient à apprendre, à identifier des formes simples et à calculer certaines fonctions logiques [12].

- 1969 : M. Minsky et S. Papert publient un ouvrage qui met en exergue les limitations théoriques du perceptron. Limitations alors connues, notamment concernant l'impossibilité de traiter par ce modèle des problèmes non linéaires. Ils étendent implicitement ces limitations à tout modèle de réseaux de neurones artificiels. Leur objectif est atteint, il y a abandon financier des recherches dans le domaine (surtout aux U.S.A.), les chercheurs se tournent principalement vers l'IA (Intelligence Artificielle) et les systèmes à bases de règles [11].

Il faudra attendre le début des années 80 pour que l'intérêt pour ce domaine soit de nouveau présent [11]. En effet, Hopfield démontre en 1982 tout l'intérêt d'utiliser les réseaux récurrents (dits "feed-back») pour la modélisation des processus. Les réseaux récurrents constituent alors la deuxième grande classe de réseaux de neurones, avec les réseaux type perceptron (dits "feed-forward").

En parallèle des travaux de Hopfield, Werbos conçoit son algorithme de rétro propagation qui ne sera pourtant popularisé qu'en 1986 par Rumelhart.

I.1.2 Neurone biologique

Le neurone est une cellule nerveuse. Elle se compose d'un corps cellulaire appelé « Soma » qui contient le noyau (où se déroule les activités cellulaires vitales) de prolongement appelé « *Neurite* » (figure (I.1)). Ces dernières sont de deux types, les *Dentrites* qui servent de canaux d'entrées et l'axone, unique qui est le canal de sortie [13].

Au point de vu fonctionnel, on considère le neurone comme une entité polarisée, c'est-à-dire que l'information ne se transmet que dans un seul sens : des dentrites vers l'axone. Le neurone va donc recevoir des informations, venant d'autres neurones, grâce à ses dentrites. Il va ensuite y avoir sommation, au niveau du corps cellulaire, de toutes ces informations et via un potentiel d'action (un signal électrique). Le résultat de l'analyse va transiter le long de

l'axone jusqu'aux terminaisons synaptiques. A cet endroit, lors de l'arrivée du signal, des vésicules synaptiques vont venir fusionner avec la membrane cellulaire, ce qui va permettre la libération des neurotransmetteurs (médiateurs chimiques) dans la fente synaptique. Le signal électrique ne pouvant pas passer la synapse (dans le cas d'une synapse chimique), les neurotransmetteurs permettent donc le passage des informations, d'un neurone à un autre.

Les synapses possèdent une sorte de « *mémoire* » qui leur permet d'ajuster leur fonctionnement. En fonction de leur « *histoire* », c'est-à-dire de leur activation répétée ou non entre deux neurones, les connexions synaptiques vont donc se modifier. Ainsi, la synapse va faciliter ou non le passage des influx nerveux. Cette plasticité est à l'origine des mécanismes d'apprentissage.

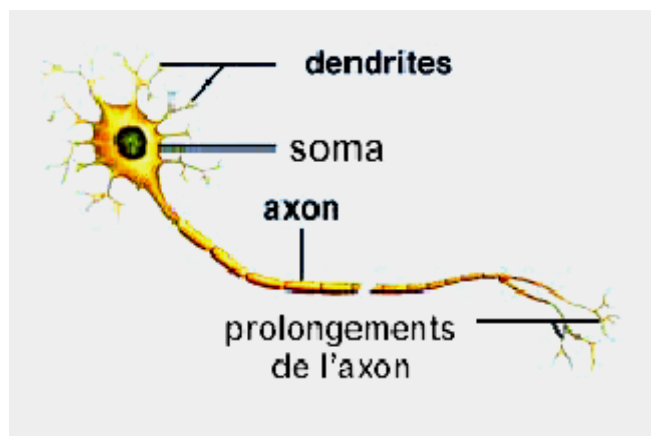


Figure I.1 Neurone biologique [14]

I.1.3 Neurone formel

Le neurone formel figure (I.2) est une modélisation mathématique qui reprend les principes du fonctionnement du neurone biologique, en particulier la sommation des entrées. Sachant qu'au niveau biologique, les synapses n'ont pas toutes la même « valeur » (les connexions entre les neurones étant plus au moins fortes), les chercheurs ont donc créé un algorithme qui pondère la somme de ses entrées par des poids synaptiques (coefficients de pondération). En général, un neurone formel est un élément de traitement possédant n entrées $x_1, x_2, \dots, x_b, \dots, x_n$ (qui sont les entrées externes ou les sorties des autres neurones) et une ou plusieurs sorties. Son traitement consiste à effectuer à sa sortie y_i le résultat d'une fonction de seuillage f (dite aussi la fonction d'activation) de la somme pondérée.

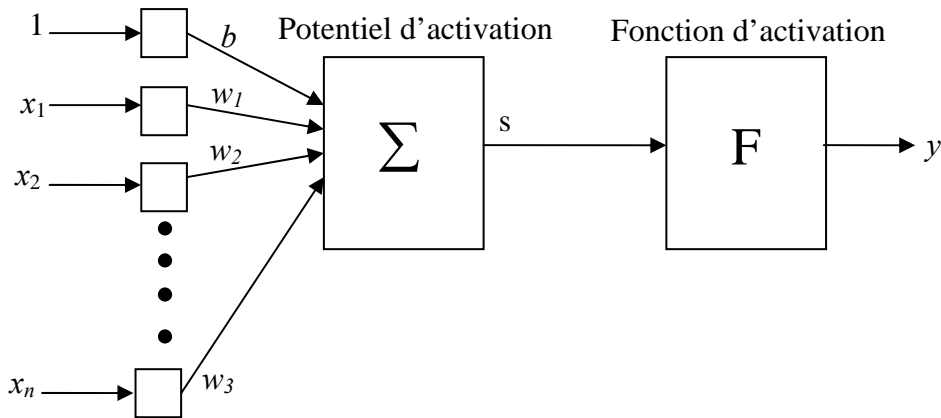


Figure I.2 Modèle d'un neurone formel

Avec :

- Les x_i sont les entrées du réseau.
- S est le potentiel d'activation.
- Les w_i représentent les poids synaptiques.
- y_i la sortie du réseau tels que : $y = f(s)$; $s = \sum_{i=0}^n w_i \cdot x_i + b$

I.1.4 Fonction d'activation [15]

La fonction d'activation (ou fonction de seuillage, ou encore fonction de transfert) sert à introduire une non linéarité dans le fonctionnement du neurone. Les fonctions de seuillage présentent généralement trois intervalles :

- en dessous du seuil, le neurone est non actif (souvent dans ce cas, sa sortie vaut 0 ou -1).
- aux alentours du seuil, une phase de transition.
- au dessus du seuil, le neurone est actif (souvent dans ce cas, sa sortie vaut 1) [16].

Dans sa première version, le neurone formel était implémenté avec une fonction à seuil, mais de nombreuses versions existent. Ainsi, le neurone de McCulloch et Pitts a été généralisé de différentes manières, en choisissant d'autres fonctions d'activations, comme les fonctions énumérées dans le tableau (I.1). Les trois fonctions les plus utilisées sont les fonctions « seuil », « linéaire », « sigmoïdes ».

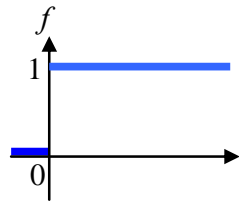
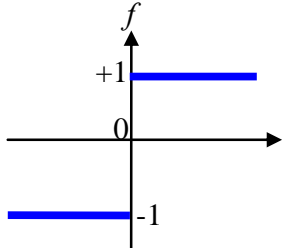
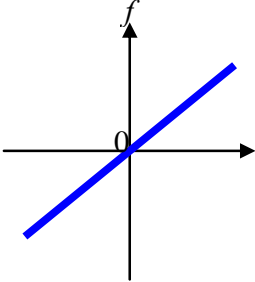
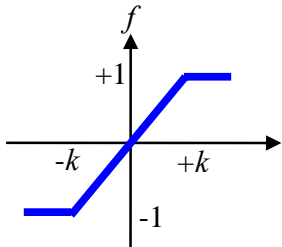
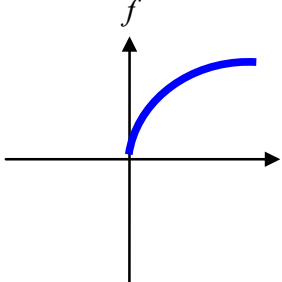
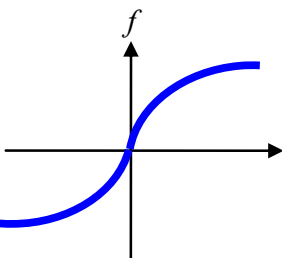
Catégorie	Type	Equation	Allure
Seuil	Binaire (Heaviside)	$f(x)=1$ si $x>0$ $f(x)=0$ si $x\leq 0$	
	Signe	$f(x)=1$ si $x>0$ $f(x)=-1$ si $x\leq 0$	
Linéaire	Identité	$f(x)=x$	
	Saturation	$f(k,x)=-1$ si $x<-1/k$ $f(k,x)=1$ si $x>=1/k$ $f(k,x)=kx$ sinon	
Sigmoide	Positive (logistique)	$f(k,x) = \frac{1}{1+e^{-kx}}$	
	Symétrique (type tanch)	$f(k,x) = \frac{2}{1+e^{-kx}} - 1$	

Tableau I.1 Fonction de transfert

I.1.5 Réseau de neurone

Un réseau neuronal se compose de plusieurs neurones ordonnés dans des plans appelés couches.

Les neurones peuvent être reliés par des connexions pondérées avec les neurones des autres couches ou avec celles de la même couche. On distingue deux types de réseau :

I.1.5.1 Réseau de neurones non bouclés [16]

Un réseau de neurone non bouclé (dit aussi statique) est donc représenté graphiquement par un ensemble de neurones connectés entre eux, l'information circulant des entrées vers les sorties sans « retour en arrière » ; c'est-à-dire à partir d'un neurone quelconque, en suivant les connexions, on ne peut pas revenir au neurone de départ (figure (I.3)).

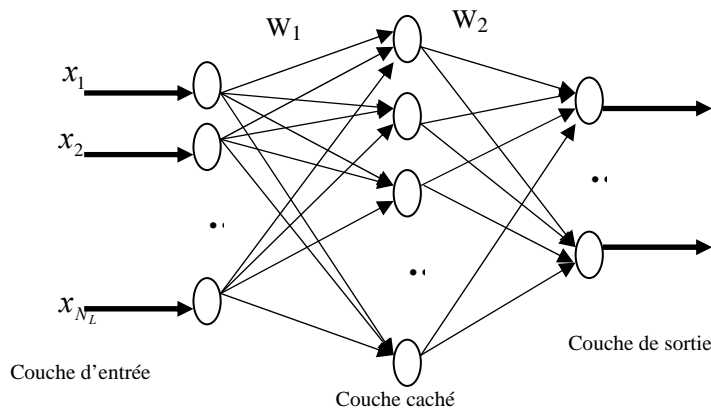


Figure I.3 Un réseau de neurone non bouclé

Le $j^{ème}$ neurone de la couche l cachée à l'instant k reçoit :

$$a_j^l(k) = \sum_{i=1}^{N_{l-1}} w_{ji}^{l-1}(k) x_i^{l-1}(k) \tag{1.1}$$

Sa sortie est donnée par :

$$s_j^l(k) = g(a_j^l(k)) \quad 1 \leq l \leq L \tag{1.2}$$

g : la fonction d'activation du neurone.

Le $j^{ème}$ neurone dans la couche de sortie reçoit

$$x_j^L(k) = \sum_{i=1}^{N_{L-1}} w_{ji}^{L-1}(k) s_i^{L-1}(k) \tag{1.3}$$

Où :

L : nombre total de couches, l : indice de couche.

N_l : Nombre de neurones présents dans la couche l .

$a_j^l(k)$: Entrée au $j^{\text{ème}}$ neurone de la couche l à l'instant k .

$s_j^l(k)$: Sortie du $j^{\text{ème}}$ neurone de la couche l à l'instant k .

$w_{ij}^l(k)$: Poids de la connexion allant du $i^{\text{ème}}$ neurone de la couche l vers le $j^{\text{ème}}$ neurone de la couche $l+1$, à l'instant k .

a) Exemple : Le réseau MLP

Le réseau MLP (Multi Layer Perceptron) ou le perceptron multicouche est un réseau orienté de neurones artificiels organisé (comme son nom l'indique) en couche ; une couche d'entrée, une couche de sortie et une ou plusieurs couches intermédiaires appelées couches cachées. La figure (I.4) donne l'exemple d'un réseau contenant n entrées, deux couches cachées et une couche de sortie.

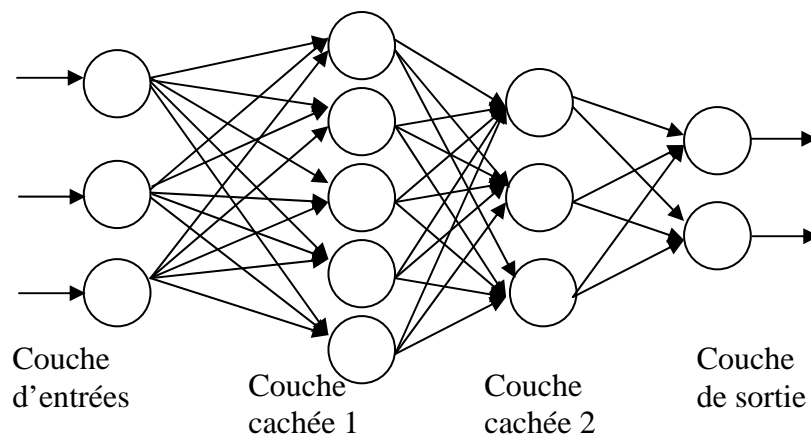


Figure I.4 Exemple d'un réseau MLP

La couche d'entrée représente toujours une couche virtuelle associée aux entrées du système, elle ne contient aucun neurone tandis que les couches suivantes représentent des couches effectives de neurones.

I.1.5.2 Réseau de neurones bouclé

Les réseaux bouclés (ou récurrents ou encore dynamiques) permettent des connexions arbitraires entre les neurones de toutes les couches ; lorsqu'on se déplace dans le réseau en

suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ. La sortie d'un neurone du réseau peut donc être fonction d'elle-même ; cela n'est évidemment concevable que si la notion de temps est explicitement prise en considération. Il peut avoir plusieurs topologies [5]. Parmi les architectures de cette classe ; le réseau de neurones à mémoires (MNN) et le réseau neuronale récurrent à temps discret (DTRNN).

a) Exemple de réseau de neurone à mémoire [2]

Les réseaux de neurones à mémoire sont une classe de réseaux de neurones récurrents obtenue en ajoutant des éléments temporels entraînaables (qui admettent l'apprentissage) à un simple réseau de neurones multicouche. (Qui rend la sortie historiquement sensible). A chaque neurone, on lui associe une mémoire dont la sortie résume l'historique des activations passées des neurones.

➤ **Structure de réseau**

L'architecture du réseau (MNN) est illustrée par la figure (I.5), elle est identique à celle du MLP à part les unités mémoires attachées à chaque neurone.

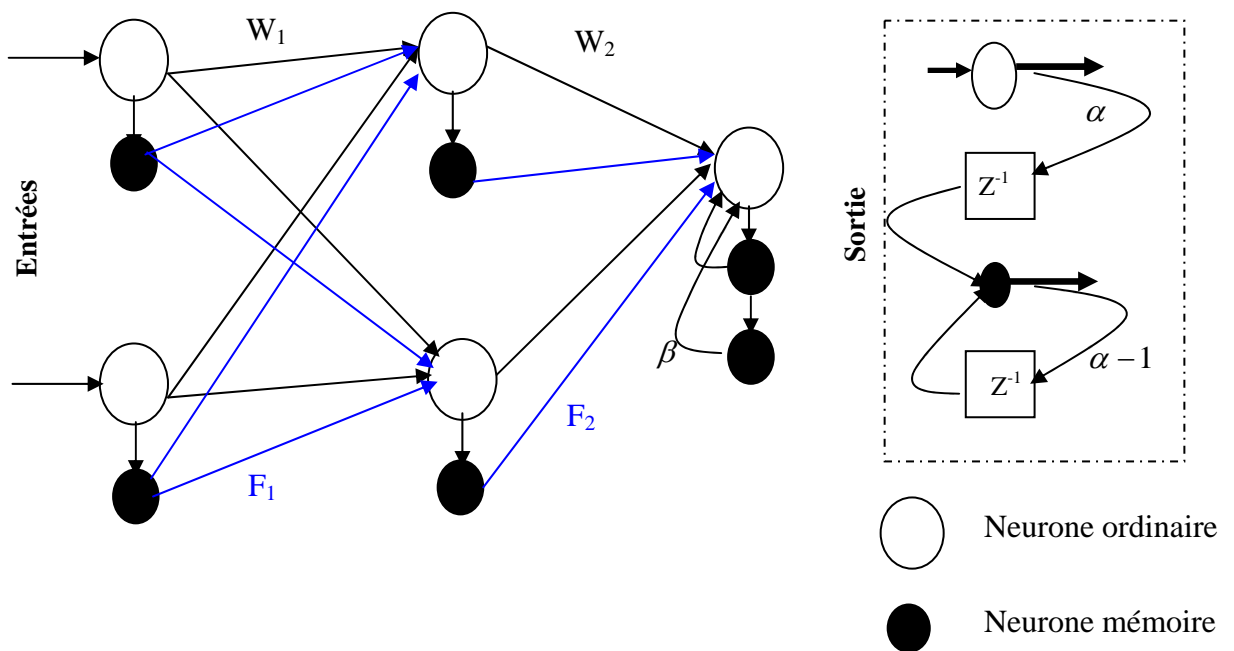


Figure I.5 Architecture d'un réseau de neurones récurrent à mémoires

b) Exemple de réseau neuronal récurrent à temps discret

Un réseau neuronal récurrent à temps discret, dit aussi DTRNN (Discrete Time Recurrent Neural Network), est analogue au réseau de Hopfield. Il est constitué d'une seule couche. Une partie des neurones sont considérés comme des nœuds d'entrée. D'autres peuvent être utilisés comme nœuds de sortie [18].

➤ Structure du réseau

La figure (I.6) montre l'architecture d'un réseau DTRNN. Parmi tous les états du réseau, on choisit une ou plusieurs sorties suivant le besoin.

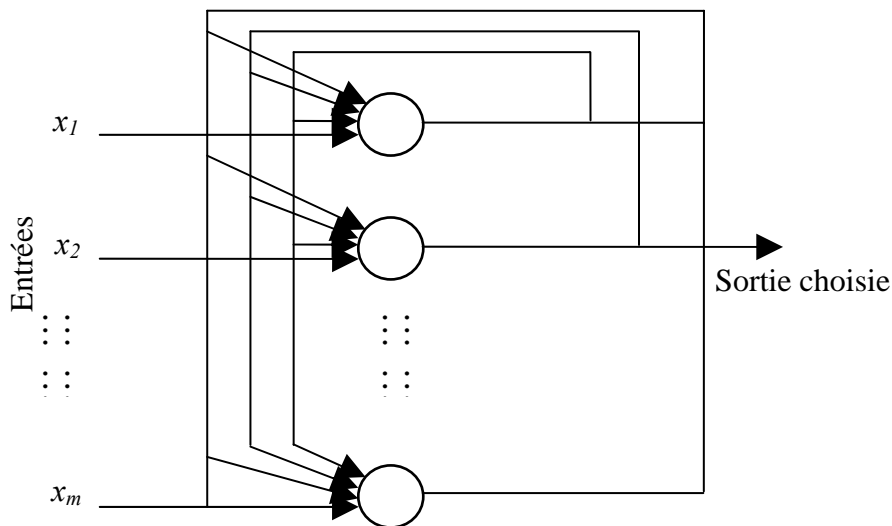


Figure I.6 Architecture d'un réseau de neurone récurrent a temps discret (DTRNN)

Dans le cas des réseaux de neurones artificiels, on ajoute souvent à la description du modèle l'algorithme d'apprentissage. Le modèle sans apprentissage présente en effet peu d'intérêt. Dans la majorité des algorithmes actuels, les variables modifiées pendant l'apprentissage sont les poids des connexions. La section qui suit est réservée aux définitions et l'explication des algorithmes d'apprentissage.

I.2 Algorithmes d'apprentissage

I.2.1 Introduction

Toute l'information que peut contenir un réseau neuronal réside dans les poids synaptiques. L'apprentissage consiste donc à ajuster ces derniers de telle façon qu'il puisse

générer correctement la sortie correspondante à chaque point de l'espace d'entrée. Ainsi, l'apprentissage peut être défini comme une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré [16]:

$$\text{apprentissage} \equiv dW/dt$$

L'une des premières méthodes d'apprentissage a été formulée par Donald Hebb en 1949 pour l'apprentissage des corrélations [13], [19].

On peut distinguer trois types d'apprentissage [19].

- **Apprentissage supervisé** : un professeur fournit au réseau des couples de données (entrée, sortie désirée correspondante). Les paramètres du réseau sont ajustés de manière à minimiser une certaine norme de l'erreur de sortie constituée par la différence entre la sortie réelle du réseau et la valeur désirée correspondante (fournie par le professeur).
- **Apprentissage par renforcement** : est une approche utilisée dans les problèmes de planification à travers le temps. Elle utilise deux réseaux : un *réseau d'action* et un *réseau d'évaluation* qui joue le rôle d'un superviseur et qui génère un signal d'avertissement à chaque fois que les actions prises sont mauvaises. Ce signal sert à entraîner le réseau d'action. Les poids du réseau d'évaluation sont altérés dans le sens de renforcer les bonnes actions et de sanctionner les mauvaises.
- **Apprentissage non supervisé** : en absence de tout professeur, le réseau organise lui-même les formes d'entrée en classes de façon à minimiser un critère de performances. Ceci peut être fait, par exemple, en désignant un certain nombre de neurones gagnants dans une compétition d'activation ou en désignant un certain nombre de bassins d'attraction dans l'espace d'état [20].

I.2.2 Méthodes d'optimisation [21]

Cette section n'a pas pour objectif d'étudier comparativement les techniques d'optimisation. Néanmoins, il est important d'identifier quelques méthodes.

L'optimisation est l'une des branches les plus importantes des mathématiques appliquées modernes et de nombreuses recherches, à la fois pratiques et théoriques, lui sont consacrées. Si on met de côté les problèmes d'optimisation discrète ou multicritère, alors la théorie de l'optimisation peut être séparée en deux grandes branches : l'optimisation locale et l'optimisation globale.

La tâche principale de l'optimisation globale est la recherche de la solution qui minimisera un critère de coût donné, appelée « *optimum global* ». L'optimisation globale vise donc à rechercher non seulement un minimum local, mais surtout le plus petit de ces minima locaux. Il existe deux grandes approches à l'optimisation globale. L'une est dite déterministe ; les algorithmes de recherche utilisent toujours le même cheminement pour arriver à la solution et on peut donc déterminer à l'avance les étapes de la recherche. L'autre est aléatoire ; pour des conditions initiales données, l'algorithme ne suivra pas le même cheminement pour aller vers la solution trouvée et peut même proposer différentes solutions.

I.2.2.1 Algorithme de la rétro-propagation [16]

Plusieurs variantes de cet algorithme ont été développées par des chercheurs travaillant dans des domaines différents. La première formulation de la version actuelle a été faite par Werbos en 1974. Il a été appliqué pour les réseaux multicouches par Rumlehart en 1986.

L'algorithme de la rétro-propagation altère les coefficients synaptiques (w_i) du réseau dans le sens inverse du gradient du critère d'erreur, en utilisant seulement les données d'entrée/sortie ;

A chaque itération, on retire un exemple d'apprentissage (x_i, y_i) et on calcule une nouvelle estimation du poids synaptique w_i .

Cette itération consiste en deux phases :

- **Propagation** : à chaque itération, un élément de l'ensemble d'apprentissage est introduit à travers la couche d'entrée. L'évaluation des sorties du réseau se fait couche par couche, de l'entrée vers la sortie [16].
- **Rétro-propagation** : cette étape est similaire à la précédente. Cependant, les calculs s'effectuent dans le sens inverse (de la sortie vers l'entrée). A la sortie du réseau, on forme le critère de performance E en fonction de la sortie réelle de système et sa valeur désirée. Puis, on évalue le gradient de E par rapport aux différents poids en commençant par la couche de sortie et en remontant vers la couche d'entrée.

I.2.2.2 Méthode de Recherche aléatoire (méthode de Monte-carlo) [21]

C'est la plus simple des méthodes stochastiques. Cette méthode consiste à tirer à chaque itération une solution au hasard. La fonction objectif f est évaluée en ce point. La nouvelle valeur est comparée à la précédente. Si elle est meilleure que la précédente, cette

valeur est enregistrée, ainsi que la solution correspondante. Et le processus continue. Sinon on repart du point précédent et on recommence le procédé, jusqu'à ce que les conditions d'arrêt soient atteintes. L'algorithme est présenté dans la figure suivante :

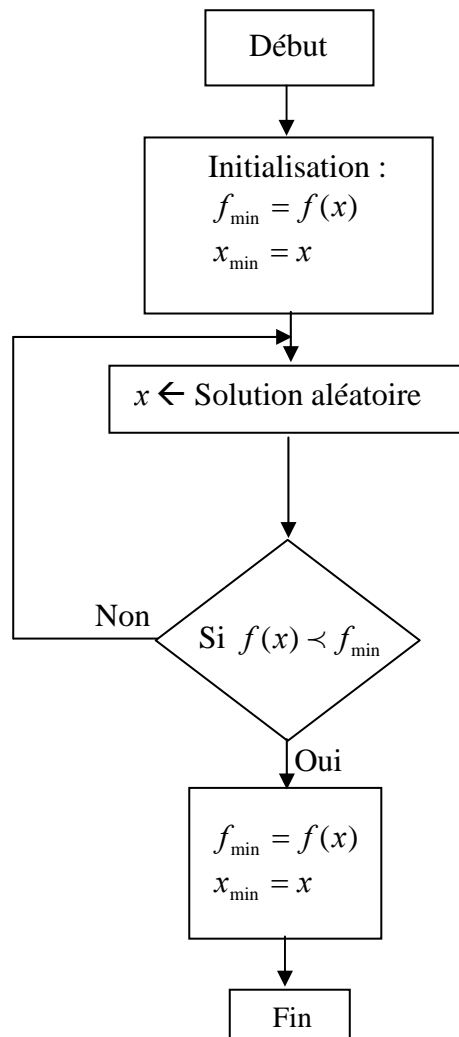


Figure I.7 L'algorithme de la recherche aléatoire

I.2.2.3 Méthode de gradient conjugué

La méthode du gradient conjugué est plus souvent utilisée comme méthode itérative pour résoudre de manière approchée les grands systèmes non linéaires. Sa force vient du fait que le nombre d'itérations nécessaires pour obtenir une bonne approximation de x est petit devant la taille N du système [22]. Elle repose sur la recherche de directions successives permettant d'atteindre la solution exacte du système étudié.

I.2.2.4 Algorithme génétiques

Les AG sont basés sur une simulation d'évolution de populations de solutions. L'objectif des algorithmes génétiques (AG) est de faire évoluer une population P dans le but de trouver l'optimum. Pour ce faire, à chaque génération t , les individus de la population sont mutés et croisés avec une probabilité et c'est les plus aptes qui survivent pour la génération suivante. Ce processus est répété pendant un certain nombre de générations, dans l'espoir que les solutions de la fonction fitness apparaissent dans la population.

Le chapitre suivant est consacré à la présentation en détail de cette méthode (AG).

I.5 Conclusion

Dans ce chapitre, nous avons exposé l'un des puissants outils de l'IA, à savoir les réseaux de neurones, que ce soit les réseaux statique ou dynamique. Trois architectures ont été présentées ; le réseau MLP (statique), MNN et DTRNN (dynamique). Ainsi, on a introduit le principe phare des RN qu'est l'apprentissage.

CHAPITRE II : LES ALGORITHMES GENETIQUES

II Introduction

Dans ce second chapitre, nous introduirons les algorithmes génétiques, métaphores biologiques inspirées des mécanismes de l'évolution Darwinienne et de la génétique, et utilisés comme outils d'optimisation ou de recherche combinatoire. Nous examinerons leurs terminologies de base, leurs principes, les opérateurs participants à l'exploration de l'espace de recherche, leur conception et leur mode de fonctionnement. En plus, nous exposerons d'une manière détaillée les méthodes les plus utilisées pour chaque opérateur génétique à savoir sélection, mutation et croisement. Nous terminerons le chapitre par une application de l'algorithme qu'on a adopté au réseau de neurones MLP appliqué à la classification non linéaire.

II.1 Algorithmes génétiques

Les algorithmes génétiques (AG) sont des techniques d'optimisation basées sur le principe Darwinien et d'évolution naturelle des espèces et de la génétique. Ils agissent sur une *population* d'*individus* qui évolue durant une succession d'itérations appelées *générations* jusqu'à ce qu'un critère, qui prend en compte à priori la qualité des solutions obtenues, soit vérifié. Seuls les individus bien adaptés à leur environnement peuvent survivre et se reproduire. Ils ont été introduits en 1975 par John Holland et ces collaborateurs comme algorithme de recherche. Ensuite, ils ont été utilisés comme outils d'optimisation [6].

Ces algorithmes font partie de la classe des algorithmes dits stochastiques. En effet, une grande partie de leur fonctionnement est basée sur le hasard. Bien qu'utilisant le hasard, les AG ne sont pas purement aléatoires. Ils exploitent efficacement l'information obtenue précédemment pour spéculer sur la position de nouveaux points à explorer, avec l'espoir d'améliorer la performance [23].

II.2 Terminologie et éléments de base

Un algorithme génétique recherche les extremums d'une fonction définie sur un espace de données appelé *population*. Par analogie avec la génétique, chaque *individu* de cette population est un *chromosome* et chaque *caractéristique* d'un individu est un *gène*. Dans un cas simple, un gène sera représenté par un bit (0 ou 1), un chromosome par une chaîne de bits. Chaque gène représente une partie élémentaire du problème, il peut être assimilé à une

variable et peut prendre des valeurs différentes appelées *allèles*. La position du gène dans le chromosome se nomme *locus* [23].

On parle également de *génotype* et de *phénotype*. Le génotype représente l'ensemble des valeurs des gènes du chromosome alors que le phénotype représente la solution réelle après transformation du chromosome. Lors de la génération d'une nouvelle population, des opérateurs génétiques telles que la sélection, le croisement et la mutation sont nécessaires pour la manipulation des chromosomes [23].

Le tableau (II.1) présente une récapitulation de la terminologie naturelle et celle utilisée par les AG [23].

Nature	Algorithme génétique
Chromosome	Chaîne
Gène	Trait, caractéristique
Allèle	Valeur de la caractéristique
Locus	Position dans la chaîne
Génotype	Structure Ensemble des valeurs des gènes
Phénotype	Ensemble de paramètres, structure décodée Evaluation d'un génotype

Tableau II.1 Comparaison de la terminologie naturelle et celle des AG

II.3 A quoi sert l'algorithme génétique ?

L'algorithme génétique résout des problèmes n'ayant pas de méthode de résolution décrite précisément ou dont la solution exacte, si elle est connue, est trop compliquée pour être calculée en un temps raisonnable. Ceci dit, face à un problème pour lequel il existe pour ainsi dire une infinité de solutions, plutôt que d'essayer naïvement toutes les solutions une à une pour trouver la meilleure, on va explorer l'espace des solutions en se laissant guider par les principes des AG.

II.4 Conception d'un algorithme génétique

La simplicité de mise en oeuvre et l'efficacité constituent deux des caractéristiques les plus attrayantes de l'approche proposée par les AG. La mise en oeuvre d'un algorithme génétique sollicite la disponibilité [23] :

- D'une représentation génétique du problème, c'est-à-dire un codage approprié des solutions sous la forme de chromosomes. Cette étape associe à chacun des points de l'espace de recherche une structure de données. La qualité du codage des données conditionne le succès des AG.
- D'un mécanisme de génération de la population initiale. Ce mécanisme doit être capable de produire une population non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut engendrer plus ou moins rapidement la convergence vers l'optimum globale. Dans le cas où l'on ne connaît rien sur tout le problème à résoudre, il est essentiel que la population initiale soit répartie sur tout le domaine de recherche.
- D'une fonction d'évaluation pour mesurer la force de chaque chromosome.
- D'un mode de sélection des chromosomes à reproduire.
- Des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace de recherche. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace de recherche.
- Des valeurs pour les paramètres qu'utilise l'algorithme : taille de la population, nombre total de générations ou critère d'arrêt, probabilité de croisement et de mutation.

La figure suivante résume ces différentes étapes

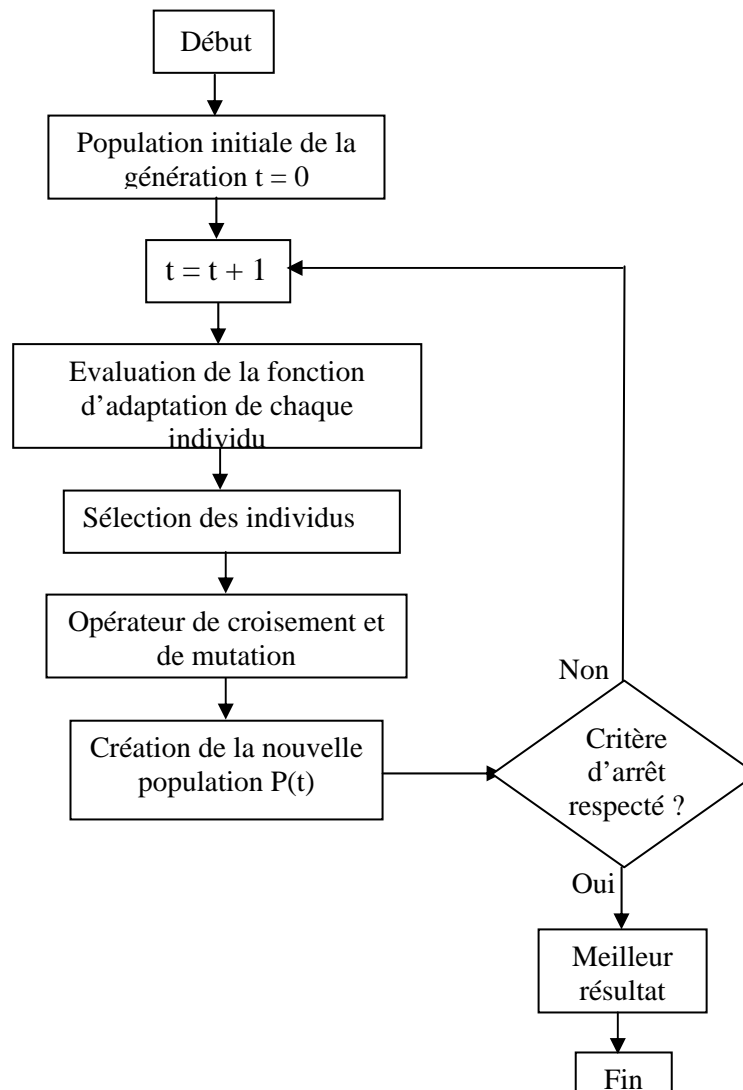


Figure II.1 Organigramme d'un AG

II.4.1 Codage des chromosomes

Le codage est une modélisation d'une solution d'un problème donné sous forme d'une séquence de caractères appelée chromosome où chaque caractère, dit aussi gène, représente une variable ou une partie du problème. La tâche principale consiste à choisir le contenu des gènes qui facilitent la description du problème et respecte ses contraintes [23]. La littérature définit deux types de codage : binaire et réel.

a. Codage binaire

Goldberg [6] et Holland [5] ont démontré qu'il est idéal de représenter le chromosome en une chaîne binaire. C'est pourquoi les AG utilisent généralement cette représentation. Dans ce cas, le chromosome représente simplement une suite de 0 et de 1. Le codage binaire est

également indépendant des opérateurs génétiques (croisement et mutation) du moment où ces derniers ne nécessitent aucune spécification. En effet, toute manipulation d'un chromosome donne naissance à un nouveau chromosome valide. Dans la pratique, le codage binaire peut présenter des difficultés. En effet, il est parfois très difficile ou très lourd de coder des solutions de cette manière. En outre, dans certains cas, la taille mémoire requise peut devenir prohibitive.

b. Codage réel

Pour certains problèmes d'optimisation, il est plus pratique d'utiliser un codage réel des chromosomes. Les auteurs dans [5], ont effectué une comparaison entre la représentation binaire et la représentation réelle. Ces auteurs ont trouvé que la représentation réelle donne de meilleurs résultats d'après leurs problèmes à résoudre.

Le gène est ainsi représenté par un nombre réel au lieu d'avoir à coder les réels en binaire puis de les décoder pour les transformer en solutions effectives.

En résumé, il n'existe pas de consensus sur la représentation idéale des individus. Soit la notation de la représentation du chromosome C suivante :

$$C = (c_1, c_2, \dots, c_i, \dots, c_L)$$

Où L représente le nombre de variables dans le chromosome et c_i représente le i^{eme} gène du chromosome C avec une représentation de type binaire ou réelle.

II.4.2 Initialisation de la population

La population initiale est constituée d'un ensemble d'individus (chromosome). Plusieurs mécanismes de génération de la population initiale sont utilisés dans la littérature [21]. Le choix de l'initialisation se fera en fonction des connaissances que l'utilisateur a sur le problème. S'il n'a pas d'informations particulières, alors une initialisation aléatoire, la plus uniforme possible afin de favoriser une exploration de l'espace de recherche maximum, sera la plus adaptée. Mais dans d'autres cas, il est possible d'utiliser d'autres mécanismes.

Le nombre d'individus d'une population, ou la taille de la population, constitue un paramètre important pour l'AG qu'il faudra déterminer. La représentation de la population P est :

$$P = (C_1, C_2, \dots, C_i, \dots, C_{taille_pop})$$

Où C_i représente la i^{eme} chromosome dans la population et $taille_pop$ représente le nombre de chromosomes dans la population.

II.4.3 Fonction d'évaluation

La fonction d'évaluation ou d'adaptation (fitness) associe un coût à chaque chromosome. Il faut distinguer entre la fonction objective et la fonction d'adaptation. Dans certains cas, elles peuvent être identiques, mais en général la fonction d'adaptation dépend de la fonction objective, laquelle dépend de la nature du problème à résoudre. La solution optimale du problème est obtenue à partir de la fonction d'évaluation du chromosome.

La fonction d'adaptation peut être soit mono critère ou multicritère. Une fonction d'adaptation mono critère signifie que la fonction dépend d'une seule fonction objective, par contre la fonction d'adaptation multicritère dépend d'une combinaison de plusieurs fonctions objectives d'où la notion *objectif unique* et *objectifs multiples*.

- **Objectif unique :** la définition de la fonction d'adaptation (fitness) ne pose pas généralement de problème, car c'est une optimisation simple à résoudre.
- **Objectifs multiples :** les problèmes d'optimisation doivent souvent satisfaire des objectifs multiples, dont certains sont concurrents. Une méthode classique consiste à définir des fonctions objectifs o_i , traduisant chaque objectif à atteindre et de les combiner au sein de la fonction d'adaptation. On établit ainsi un compromis.

Il existe des méthodes pour la définition des fonctions à optimiser qui dépendent du problème à résoudre. Parmi ces méthodes de fonction d'adaptation, on trouve [24]:

II.4.3.1 No scaling

C'est la méthode la plus facile pour le calcul de la valeur de la fonction d'adaptation f_i du chromosome C_i

$$F_i = o_i \quad (2.1)$$

Où o_i est la fonction objectif du chromosome C_i .

II.4.3.2 Echelle linéaire (Linear scaling)

La valeur de la fonction d'adaptation f_i du chromosome C_i est en relation linéaire avec la valeur de la fonction objectif o_i [5], [24].

$$F_i = ao_i + b \quad (2.2)$$

Où a et b sont choisis en fonction des valeurs des fonctions objectives des individus [6]. Cette méthode produit une bonne solution. Cependant, cette fonction d'adaptation peut être négative. Pour éviter la valeur négative, il suffit de la remplacer par une valeur nulle.

II.4.3.3 Le sigma tronqué (sigma truncation)

Cette méthode est utilisée lorsque la valeur de la fonction objective est négative. La valeur de f_i du chromosome C_i est calculée de la manière suivante

$$f_i = o_i - (\bar{o} - c\delta) \quad (2.3)$$

Où c est une petite valeur entière et \bar{o} est la moyenne des fonctions objectives. Cette moyenne correspond à la somme pondérée de toutes les valeurs de la fonction de chaque individu d'une population.

$$\text{Autrement dit : } \bar{o} = \frac{\sum_{i=1}^{\text{taille_pop}} f_i}{\text{taille_pop}}$$

δ est la déviation standard de la population.

Afin d'éviter la valeur négative de f_i , chaque résultat négatif est mis à zéro.

II.4.3.4 Puissance d'échelle (power law scaling)

La valeur de f_i est définie par la relation suivante

$$f_i = o_i^k \quad (2.4)$$

Où k est une valeur qui varie en fonction des générations.

II.4.3.5 Rangement (ranking)

La valeur de f_i ne dépend pas directement de la valeur de la fonction objective mais s'associe au rang de la valeur de la fonction objective. Cette méthode permet d'éviter la convergence prématurée et d'accélérer la recherche quand la population converge vers une solution. D'autre part, elle exige des calculs supplémentaires dans le tri des valeurs objectives des chromosomes de la population.

II.5 La sélection

La sélection est chargée de « favoriser » les meilleurs individus [23]. Plus formellement, l'opérateur de sélection va générer à partir de la population courante une nouvelle population par copie des individus choisis de la population courante. La copie des chaînes s'effectue en fonction des valeurs de la fonction d'adaptation. Ce procédé permet de donner aux meilleures chaînes, une probabilité élevée de contribuer à la génération suivante. Cet opérateur est bien entendu une version artificielle de la sélection naturelle, la survie Darwinienne des chaînes les plus adaptées [23].

Il existe de nombreuses techniques de sélection, les plus courantes seront évoquées dans la section suivante.

II.5.1 Sélection ordonnée (Ranck selection)

Chaque chromosome C_i d'une population P est évalué par la fonction d'adaptation f_i .

Les valeurs de la fonction d'adaptation obtenues pour l'ensemble des chromosomes seront classées dans un ordre croissant ou décroissant. Les meilleurs chromosomes seront donc sélectionnés.

II.5.2 Sélection par la roulette biaisée

Man et Al [5] ont mentionné que la technique RWS (Roulette Wheel Sélection) est la plus utilisée dans la littérature. Elle consiste à attribuer à chaque chromosome C_i une probabilité p_i , proportionnelle à sa valeur f_i dans la population :

$$p_i = \frac{f_i}{\sum_{j=1}^{taille_pop} f_j} \quad (\text{II.5})$$

Où $\sum_{j=1}^{taille_pop} f_j$ représente la somme de toutes les valeurs des fonctions d'adaptation de chaque chromosome C_i de la population P .

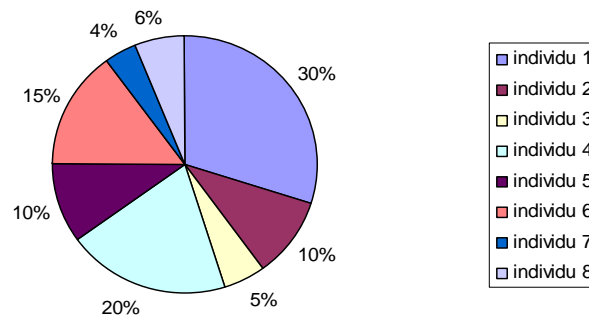


Figure II.2 La roulette proportionnelle

Lors de la phase de sélection, les individus sont sélectionnés aléatoirement en respectant les probabilités p_i associées pour former la population de la nouvelle génération. Ceci s'effectue pour le calcul d'une probabilité de sélection cumulée q_i telle que :

$$q_i = \sum_{j=1}^i p_j \quad (2.6)$$

Un individu dont la fitness est relativement faible aura un petit nombre de copies. On génère un nombre réel r aléatoirement sur l'intervalle $[0,1]$. Cette valeur est générée plusieurs fois en fonction de la taille de la population $taille_pop$. L'individu C_i est sélectionné lorsque $q_{i-1} < r < q_i$.

II.5.3 Sélection par tournoi

Elle consiste à choisir aléatoirement deux ou plusieurs individus et à sélectionner le plus fort. C'est-à-dire, elle utilise la méthode de la roulette biaisée, mais on récupère l'individu dont la valeur de la fonction d'adaptation est la plus grande. Ce processus est répété plusieurs fois jusqu'à l'obtention de N individus. L'avantage d'une telle sélection est d'éviter qu'un individu très fort soit sélectionné plusieurs fois [23].

II.5.4 Sélection uniforme [24]

C'est une technique très simple qui consiste à sélectionner un individu C_i aléatoirement de la population P . la probabilité p_i pour qu'un individu soit sélectionné est définie par :

$$p_i = \frac{1}{taille_pop} \quad (2.7)$$

II.6 Croisement

Le croisement (Hybridation) a pour but d'enrichir la diversité de la population en manipulation les composantes des chromosomes. Classiquement, les croisements sont envisagés avec deux parents et génèrent deux enfants.

A partir de deux chromosomes, on obtient deux nouveaux chromosomes (enfants) qui héritent de certaines caractéristiques de leurs parents. Le croisement sélectionne des gènes parmi deux chromosomes appelés parents. A partir de ces gènes sont générés les enfants. La probabilité de croisement représente la fréquence à laquelle les hybridations sont appliquées.

- S'il y a hybridation, les fils sont composés d'une partie de chacun de leurs parents.
- Si la probabilité est de 0%, la nouvelle génération est la copie de la précédente.
- Si la probabilité est fixée à 100%, tous les descendants sont générés par hybridation.

L'hybridation est mise en place pour que les nouveaux chromosomes gardent la meilleure partie des chromosomes anciens. Ceci dans le but d'obtenir, peut être, de meilleurs chromosomes. Néanmoins, il est quand même important qu'une partie de la population survive à la nouvelle génération [25].

Une notation des chromosomes qui est adoptée dans les prochaines sections est la suivante :

Le chromosome parent 1 : $C^1 = (c_1^1, c_2^1, \dots, c_i^1, \dots, c_L^1)$

Le chromosome parent 2 : $C^2 = (c_1^2, c_2^2, \dots, c_i^2, \dots, c_L^2)$

Où : $c_i^k \in [a_i, b_i]$ dans le cas du codage réel ou bien $c_i \in \{0,1\}$ avec $k=1,2$ dans le cas du codage binaire.

II.6.1 Croisement simple

Le croisement simple dit aussi le croisement à un site est le plus connu dans la littérature. Il consiste à choisir au hasard (aléatoirement) un site de coupure qui soit compris entre 1 et L-1, puis subdivisé le chromosome de chacun des parents en deux parties de part d'autre part de ce point, ce qui produit deux enfants. Ce type de croisement est représenté par la figure (II.3).

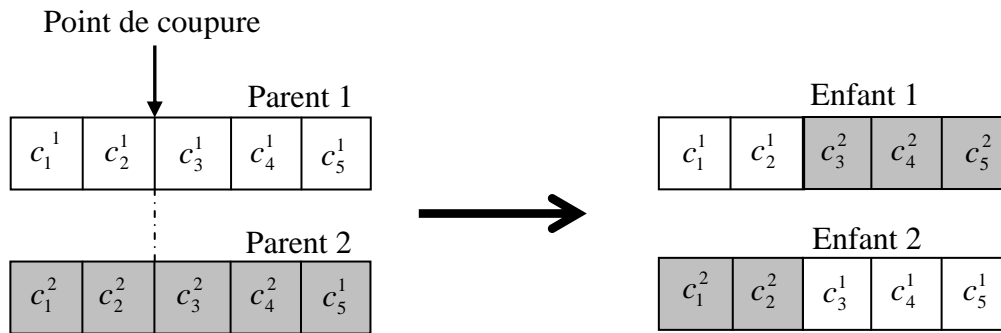


Figure II.3 Croisement simple

II.6.2 Croisement multiple

Ce type de croisement peut être vu comme une génération du croisement simple, en utilisant plusieurs points de coupure. Le nombre de points de coupure généré est en moyenne $L/2$ où L est la taille du chromosome [5]. L'individu est représenté sous la forme d'un anneau et l'échange de gènes entre les deux parents s'effectue de la façon suivante (voir figure (II.4))

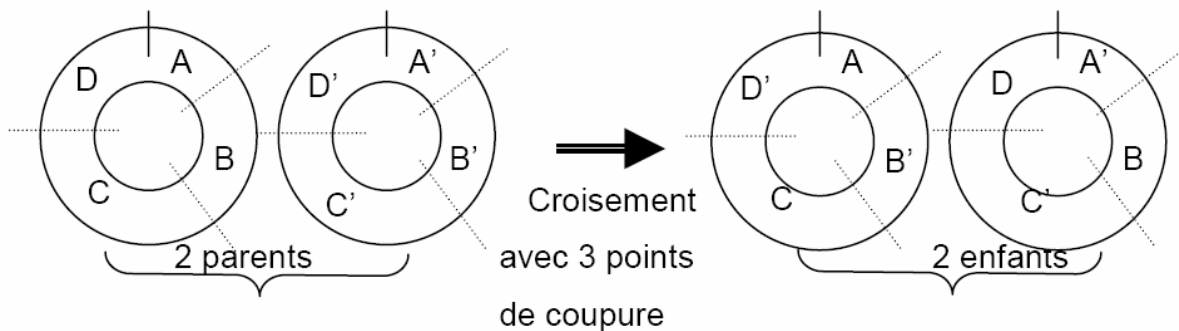


Figure II.4 Croisement avec 3 points de coupure

C'est une technique très utilisée dans différentes applications du fait que les résultats obtenus sont satisfaisants [5].

II.6.3 Croisement uniforme

Cette technique est complètement différente des deux techniques précédentes. Un masque de croisement est généré aléatoirement pour chaque couple d'individus ou pour chaque génération. Les valeurs de ce masque sont binaires. Sa taille est identique à celle du chromosome. Son fonctionnement est illustré par la figure (II.5) :

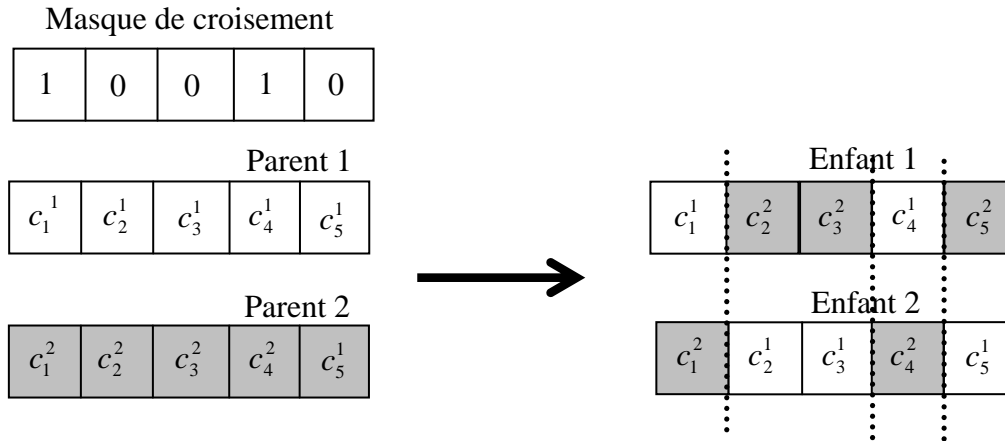


Figure II.5 Croisement uniforme

Le fonctionnement de cette technique est le suivant : si la valeur du bit du masque est égale à 1 alors la valeur du gène du parent1 est copiée chez l'enfant1 et si la valeur du bit du masque est égale à 0 alors la valeur du gène du parent2 est transmise à l'enfant1. Les valeurs des gènes de l'enfant2 sont les suivantes : les valeurs des gènes du parents1 lorsque la valeur du bit du masque est égale à 0 et les valeurs des gènes du parents2 lorsque la valeur du bit du masque est égale à 1.

II.6.4 Croisement arithmétique [5]

Lorsque cette opération est opérée sur les parents C^1 et C^2 , deux enfants (progénitures) sont générés : $H^k = (h_1^k, h_2^k, \dots, h_i^k, \dots, h_L^k)$ avec $k = 1, 2$ tels que :

$$h_i^1 = \lambda_i c_i^1 + (1 - \lambda_i) c_i^2 \quad \text{et} \quad h_i^2 = \lambda_i c_i^2 + (1 - \lambda_i) c_i^1 \tag{II.8}$$

La valeur de λ_i est une constante choisie par l'utilisateur dans le cas d'un croisement arithmétique uniforme. Par contre, dans le cas de croisement arithmétique non uniforme la valeur de λ_i est généré aléatoirement. Cette technique est montrée dans la figure (II.5).

II.6.5 Croisement BLX- α

Cet opérateur est développé par Eshelman et al [23]. Soit l'enfant générer $H = (h_1, h_2, \dots, h_i, \dots, h_L)$ où h_i est choisi aléatoirement dans l'intervalle $[c_{min} - l\alpha, c_{max} + l\alpha]$ tels que :

$$c_{\max} = \max(c_i^1, c_i^2), c_{\min} = \min(c_i^1, c_i^2) \text{ et } l = c_{\max} - c_{\min} \quad (2.9)$$

$\alpha = 0.5$.

II.6.6 Croisement linéaire

Le croisement linéaire génère trois progénitures $H^k = (h_1^k, h_2^k, \dots, h_i^k, \dots, h_L^k)$ avec $k=1, 2, 3$ tels que :

$$h_i^1 = 1/2(c_i^1 + c_i^2), h_i^2 = 3/2 c_i^1 - 1/2 c_i^2, h_i^3 = -1/2 c_i^1 + 3/2 c_i^2 \quad (2.10)$$

Lors de la sélection, la nouvelle génération sera formée des deux meilleurs de ces enfants. Cet opérateur a été développé par Wright [5].

II.6.7 Croisement discret

Les valeurs des gènes h_i des enfants générés H sont égales aux valeurs des gènes c_i^1 du parent C^1 ou aux valeurs des gènes c_i^2 du parent C^2 . Le choix du parent est effectué d'une façon aléatoire.

II.6.8 Croisement étendu

Cet opérateur génère le gène h_i du chromosome H comme suit :

$$h_i = c_i^1 + \alpha(c_i^2 - c_i^1) \quad (2.11)$$

α est choisie aléatoirement de l'intervalle $[-d, 1+d]$.

Muhelenbein [5] a déterminé la valeur de d à 0. Dans le cas contraire, ce type de croisement est appelé « *croisement étendu intermédiaire* ». Le bon choix de d est 0.25 et la valeur de α varie pour chaque gène. Autrement dit la formule (II.11) s'écrit :

$$h_i = c_i^1 + \alpha_i(c_i^2 - c_i^1) \quad (2.12)$$

La figure suivante résume les cinq derniers opérateurs.

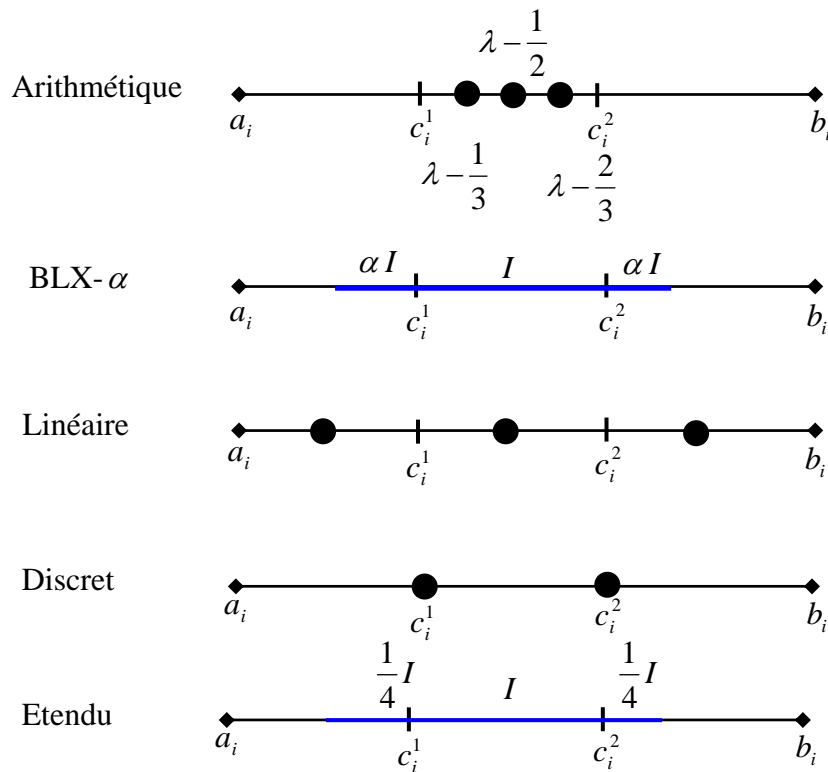


Figure II.6 Représentation des opérateurs de croisement [5]

Certaines remarques sur le croisement sont illustrées dans les points suivants :

- Le croisement est la clef de la puissance des AG. Il est lié à l'aptitude qu'à une population d'individus d'explorer son espace de recherche et de combiner entre eux les meilleurs résultats. Grâce au croisement, les AG se concentrent sur les parties les plus prometteuses de l'espace des solutions du fait que cet opérateur de croisement combine des chaînes contenant des solutions partielles.
- Le croisement n'est habituellement pas appliqué à toutes les paires d'individus choisis aléatoirement lors de la reproduction. La probabilité du croisement appliquée est comprise entre 0.6 et 1.0 [5]. Dans le cas où le croisement ne s'appliquerait pas, alors les enfants sont semblables aux parents.

II.7 Mutation

La mutation est définie comme étant la modification aléatoire d'une partie d'un chromosome. Elle consiste en une exploration aléatoire de l'espace des chaînes [23]. C'est un phénomène qui a un rôle théoriquement plus marginal : il est là pour éviter une perte irréparable de la diversité [23]. Différentes manières de mutation d'un chromosome sont aussi définies dans la littérature. Certaines d'entre elles s'appliquent sur des gènes dont la

représentation est binaire et d'autres sur des gènes de types réels. Pour déterminer le nombre de positions dont les gènes doivent subir un changement, il suffit de connaître la taille du chromosome L et la probabilité de mutation p_{mut} . Ce nombre est défini par le produit de $L * p_{mut}$.

Posons la notation suivante du chromosome qui doit subir la mutation :

$C = (c_1, c_2, \dots, c_i, \dots, c_L)$ tels que $c_i \in [a_i, b_i]$ ou $c_i \in \{0,1\}$. Le résultat obtenu par l'opérateur de mutation est un chromosome C' tels que $C' = (c'_1, c'_2, \dots, c'_i, \dots, c'_L)$.

II.7.1 Mutation aléatoire

Dans le cas réel, le gène c_i est modifié par le gène c'_i . Cette valeur du gène (c'_i) est choisie aléatoirement de l'intervalle $[a_i, b_i]$.

Dans le cas binaire, si la valeur du gène à muter est égale à 1, alors elle est inversée à 0 et si la valeur du gène est égale à 0, alors elle est inversée à 1. La figure (II.7) illustre l'effet de la mutation sur une chaîne binaire. La position du gène qui doit subir la mutation est déterminée aléatoirement.

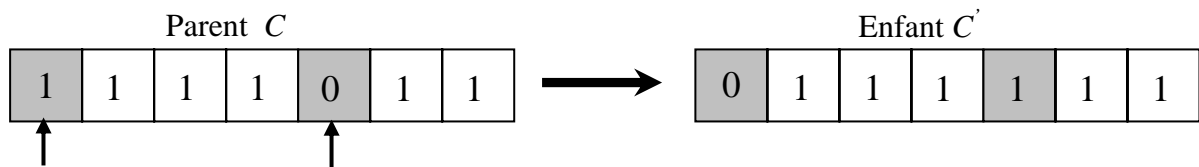


Figure II.7 Mutation aléatoire binaire

II.7.2 Mutation non uniforme

Cette technique est appliquée en fonction de la génération t courant et le nombre maximum de génération gen_max . Le gène c_i est défini comme suit

$$c'_i = \begin{cases} c_i + \Delta(t, b_i - c_i) & \text{si } \tau = 0 \\ c_i - \Delta(t, c_i - a_i) & \text{si } \tau = 1 \end{cases} \quad (2.13)$$

Où τ est généré aléatoirement tels que $\tau \in \{0,1\}$ et

$$\Delta(t, y) = y \left(1 - r \left(1 - \frac{t}{gen_max} \right)^b \right) \quad (2.14)$$

Où r est un nombre aléatoire de l'intervalle $[0,1]$, et b est un paramètre choisi par l'utilisateur.

II.8 Réinsertion

La réinsertion ou l'*élitisme* [5] : elle consiste à copier le meilleur ou un sous-ensemble des meilleurs chromosomes de la génération courante à la prochaine génération. Cette stratégie d'élitisme permet d'augmenter la dominance des meilleurs individus dans une population, ce qui permet une amélioration des solutions à obtenir.

II.9 Les paramètres de base d'un AG

Pour lancer l'AG, il faut définir certains paramètres tels que : la taille de la population, les probabilités de mutation et de croisement et le nombre de génération. Il est difficile de les fixer ou de trouver les meilleurs avant l'exécution de l'algorithme [5], [6].

II.9.1 Population

La rapidité de l'algorithme est fortement dépendante du choix de la population initiale d'individus. Si la position de l'optimum dans l'espace d'état est totalement inconnue, il est naturel et plus simple de procréer aléatoirement des individus en faisant des tirages uniformes dans l'espace d'état en veillant à ce que les individus produits respectent les contraintes. Si par contre, des informations a priori sur le problème sont disponibles, les individus sont générés dans un sous domaine particulier afin d'accélérer la convergence. Les AG doivent travailler sur des populations importantes.

II.9.2 Taille de la population

Certaines approches utilisent une population à taille fixe, alors que d'autres proposent des tailles variables, qui évoluent au cours du temps. Au lieu d'utiliser une population de taille fixe ou la fonction d'adaptation sert à la sélection, elles proposent plutôt d'indexer l'espérance de vie d'un individu à sa fonction d'adaptation. Les meilleurs individus reçoivent une espérance de vie supérieure à celle des moins bons. Au fur et à mesure que la population évolue, les individus vieillissent et finissent par mourir lorsqu'ils arrivent au terme de leur espérance de vie.

II.9.3 Taux de croisement

Il détermine la proportion des individus qui sont croisés qui remplaceront l'ancienne génération. Si le taux de croisement est trop élevé, les structures performantes sont trop fortement détruites. Par contre, s'il est petit, la population stagne.

II.9.4 Taux de mutation

A chaque reproduction, le nouveau-né est soumis à un processus mutation. En cas de succès, un gène est modifié. Pour les populations faibles, nous constatons que le taux de mutation est très élevé [6], [5].

II.9.5 Critère d'arrêt

Comme tous les algorithmes évolutionnaires, les AG sont dans l'absolu, sans fin, donc pour décider quand arrêter l'algorithme, la technique la plus courante et la plus simple est d'effectuer un nombre prédéfini d'itérations. On peut distinguer trois grandes familles de critères d'arrêt :

- Le temps ou le nombre d'itérations voulu est atteint.
- La fonction de coût est constante depuis quelque temps.
- La population est dominée par quelques individus.

Avant de se lancer dans l'application des AG dans l'identification, nous avons voulu connaître la convergence de l'algorithme implémenté. Pour cela, on va appliquer notre algorithme pour optimiser un réseau de neurone MLP appliqué à la classification des données.

II.10 Classification des données

Diverses méthodes de classification existent. Elles peuvent être classées en deux catégories : méthodes probabilistes qui permettent le calcul de l'appartenance d'un objet à une classe particulière et les méthodes séparatistes qui cherche les frontières qui séparent les classes. Les réseaux de neurones sont utilisés dans les deux catégories. Dans notre cas, on s'intéresse à la deuxième catégorie. On utilise le réseau de neurones MLP optimisé par les AG pour séparer deux classes.

La classification automatique est la dernière phase d'un processus de reconnaissance de formes. Il consiste à développer la règle de décision qui attribut à chaque objet sa classe.

Ces objets nommés exemples, sont décrits par un vecteur forme de n paramètres. Ce vecteur est représenté dans l'espace de n dimensions. Dans la littérature, nous trouvons quatre techniques de classifications différentes :

- Classification bayésienne, fondée sur la connaissance des différentes règles de probabilité. Suivant la formule de Bayes suivante :

$$proba(c = c_{i/X}) = \frac{p_i \cdot f_{X/C_i}(X)}{\sum_{i=1}^n p_i \cdot f_{X/C_i}(X)} \quad \text{qui permet le calcul de la probabilité à}$$

posteriori d'un objet appartenant à la classe C_i . La représentation de cet objet est X , qui a pour fonction de la densité de probabilité f_{X/C_i} et la probabilité à priori p_i . L'objet sera attribué à la classe susceptible [26].

- Classification paramétrique en utilisant la fonction de discrimination optimisée par l'entraînement. Cette technique consiste à faire une hypothèse sur la forme analytique de la probabilité de distribution recherchée [26]. Puis l'estimation des paramètres (moyenne, covariance ...) de cette distribution avec apprentissage. La méthode la plus utilisée est la méthode Gaussienne.
- Classification non paramétrique en utilisant la fonction discriminative basé sur l'estimation de la densité des différentes classes voisines de l'observation. Cette méthode est nommée *k plus proche voisins (kpp)*. Elle consiste à mesurer la distance entre la position du nouvel objet et les classes voisines. Ce nouvel objet sera attribué à la meilleure classe représentative parmi les k observation délimités autour de ce nouvel objet.
- Classification par réseau de neurones qui permet d'estimer la probabilité à posteriori d'objets appartenant à une classe [26].

II.10.1 Description de la méthode de classification utilisée

Le but est de réaliser une discrimination supervisée des données en deux classes. Cela signifie que les données sont étiquetées par un superviseur (expert qui connaît le classement attendu [26], chaque objet est affecté a une classe spécifique. Pour cela, on a choisi d'utiliser un réseau de neurone *MLP* (utilisé dans plusieurs applications industrielles [26]). Ce réseau est composé d'une couche d'entrée, une ou plusieurs couches cachées (dans notre application une seule couche cachée) et une couche de sortie. Chaque neurone est connecté avec les neurones de la couche suivante (voir la figure (II.8)). En plus, tous les neurones reçoivent une

entrée *biais* pris dans notre cas à 1. Le nombre de neurones dans chaque couche est choisi par l'utilisateur.

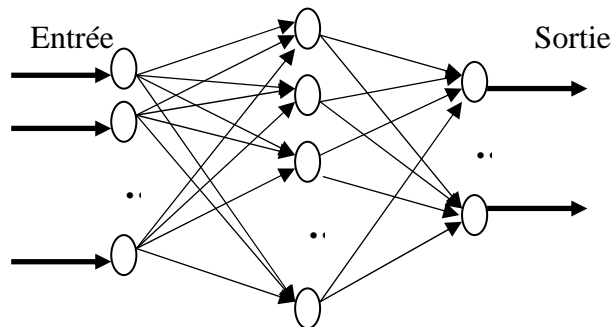


Figure II.8 Réseau *MLP* avec une seule couche cachée

Le neurone primaire reçoit un nombre variable d'entrées x_i pondérées par w_i , b est le biais. La somme des entrées pondérée représente le potentiel d'activation. La fonction d'activation permet de calculer l'état des neurones en fonction de la valeur du potentiel d'activation. $y = f(s)$, $s = \sum w_i x_i + b$ avec y est la sortie du réseau.

II.10.2 Apprentissage

L'apprentissage des réseaux de neurones artificiels est la phase permettant de modifier les paramètres du réseau. Afin d'avoir le comportement désiré, on utilise les AG. La modification des paramètres du réseau est articulée sur la minimisation d'une fonction coût quadratique entre la sortie désirée et la sortie réelle du réseau.

a) Mise en oeuvre de l'algorithme

Pour un neurone dans la couche cachée, la sortie est :

$$S_j = F\left(\sum_{i=1}^n w_{ji} \cdot x_i + b_j\right) = F\left(\sum_{i=1}^n w_{ji} \cdot x_i\right), x_0 = 1 \text{ et } w_{j0} = b_j.$$

Avec : w_{ji} indique la pondération de la connexion entre le neurone de la couche cachée et la i^{eme} entrée, et b_j est le biais associé à la j^{eme} neurone, F est la fonction d'activation du neurone de cette couche.

Pour le neurone dans la couche de sortie on a :

$$S = G\left(\sum_{i=0}^{ncc} w_{kj} \cdot S_j\right) \quad \text{avec : } w_{kj} \text{ est la pondération entre le neurone de la sortie et la } j^{eme}$$

neurone de la couche cachée, ncc est le nombre de neurones de la couche cachée, G la fonction d'activation du neurone de la couche cachée du réseau.

Pour l'adaptation des poids synaptiques w on utilise les AG, l'algorithme doit fournir au réseau *MLP* les meilleurs poids de connexion.

b) Fonction objective

Nous désirons que le réseau classifie exactement notre base de données en deux classes. Pour cela, nous avons à minimiser le critère suivant : $e(i) = \left(\sum_{i=1}^N (ydd(i) - s(i))\right)^2$.

$e(i)$: Erreur quadratique instantanée et N la longueur de la séquence d'entraînement.

c) Les paramètres utilisés dans l'AG

Le tableau (II.2) représente les paramètres utilisés dans l'AG pour l'exemple d'application

La taille de la population	1000
Nombre de générations	150
Probabilité de croisement	0.9
Probabilité de mutation	0.08
Type de codage	Réel
Type de croisement	Plusieurs sites
Type de mutation	A un seul site
Méthode de sélection	Roulette proportionnelle

Tableau II.2 Les paramètres utilisés dans l'AG

II.10.3 Application au réseau MLP

Dans ce cas, le chromosome est constitué de :

$$[w_{11} \ w_{12} \ \dots \ w_{1ncc} \ w_{21} \ w_{22} \ \dots \ w_{2ncc} \ Z_1 \ Z_2 \ \dots \ Z_{ncc}]$$

Où:

ncc : est le nombre de neurones de la couche cachée.

$w_{11} \ w_{12} \ \dots \ w_{1ncc}$: poids de connexion entre l'entrée 1 et les neurones de la couche cachée.

$w_{21} \ w_{22} \ \dots \ w_{2ncc}$: poids de connexion entre l'entrée 2 et les neurones de la couche cachée.

$Z_1 \ Z_2 \ \dots \ Z_{ncc}$: poids de connexion entre les neurones de la couche cachée et les neurones de la couche de sortie.

L'entrée 1 : est l'entrée des données à classer.

L'entrée 2 : est l'entrée désirée.

La longueur du chromosome dépend du nombre de neurones de la couche cachée (la valeur de ncc) ; ainsi si par exemple , $ncc = ncc_{min}$ ($ncc = 2$) alors la longueur du chromosome est égale à 6 (les poids de connexion entre l'entrée 1 et les neurones de la couche cachée (2 gènes), les poids de connexion entre l'entrée 2 et les neurones de la couche cachée (2 gènes), les poids de connexion entre les neurones de la couche cachée et le neurone de la couche de sortie (2 allèle)).

Et si $ncc = 10$, la longueur est égale à 30 ; et en général $Lch=3*ncc$, où Lch est la longueur de chromosome.

La taille de la population (matrice population) est égale à $N*3*ncc$.

N : est le nombre d'individus de la population.

a) Instruction de l'AG implémenté

Pour chercher la meilleure architecture du réseau MLP, on exécute les différentes instructions de l'AG :

Début

- Initialiser le nombre de génération de l'AG, choisir le nombre de neurone dans la couche cachée.
- Initialiser la première population avec des valeurs aléatoires selon le nombre de génération et le nombre d'individus (N).

Pour chaque génération les calculs suivants sont effectués :

- Pour chaque individus :
 - Calculer l'évaluation de chaque individu.
- Calcul de l'évaluation totale de la population constituée de N individus.
- Calcul de la probabilité de sélection, p_s , de chaque individu :

$$p_{sj} = \text{evaluation individu } j / \text{l'évaluation totale de la population}$$

- Pour sélectionner à l'aide de la roue de loterie biaisé, on fait tourner la roulette N fois (taille de la population) de la façon suivante : à chaque fois, on génère aléatoirement un nombre r dans l'intervalle $[0,1]$. Ensuite, on compare ce nombre aux p_s . Si $r < p_s$ l'individu est sélectionné.
- Pour chaque individu de la nouvelle génération, on génère, au hasard, N nombre r dans $[0,1]$ et on les compare à la probabilité de croisement p_c . Si $r < p_c$ alors l'individu est sélectionné pour le croisement. Sinon il ne l'est pas.
- Les individus ainsi sélectionnés seront croisés deux à deux.
- On mute les allèles des gènes des différents individus ; constituant la nouvelle population obtenue après le croisement. Si le nombre généré arbitrairement r est inférieur à la probabilité de mutation p_m .
- Si le critère d'arrêt n'est pas encore atteint, la population obtenue après la mutation sera considérée comme étant la population initiale et le processus sera réitéré.

Fin.

b) La génération des données

Les données d'apprentissage sont générés avec le logiciel MATLAB, en utilisant la fonction $\text{randn}(n)$ qui nous permet de générer une base *Gaussienne* complexe de n éléments de centre $(0,0)$ et de dispersion 1 .

La figure suivante montre un exemple de génération de deux classes (° , +) de 100 éléments.

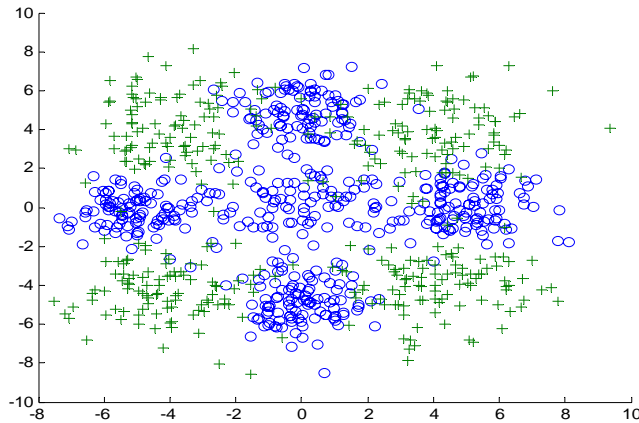


Figure II.9 Génération de deux classes

Pour séparer les données représentées dans la figure précédente en deux classes, on utilise le réseau de neurone *MLP* qui est performant dans la séparation des surfaces complexes.

D'abord, on entraîne le réseau c'est-à-dire on calcule les poids optimaux avec les AG, puis on génère une grille (ces coordonnées varient entre la valeur maximale et minimale des axes des données représentées précédemment), et on calcule la sortie du système pour chaque point, ensuite, on calcule tous les points correspondant à la sortie 0 ± 0.1 . Ces points constituent la frontière des deux classes.

La configuration optimale du réseau est obtenue après plusieurs tests. Elle correspond à : 10 neurones dans la couche cachée, 2 neurones dans la couche d'entrée et un seul neurone dans la couche de sortie. Comme fonction d'activation on a utilisé la fonction tangente hyperbolique comme fonction d'activation, pour les neurones de la couche cachée et celle de la couche de sortie.

On présente dans la figure (II.10) le résultat obtenu :

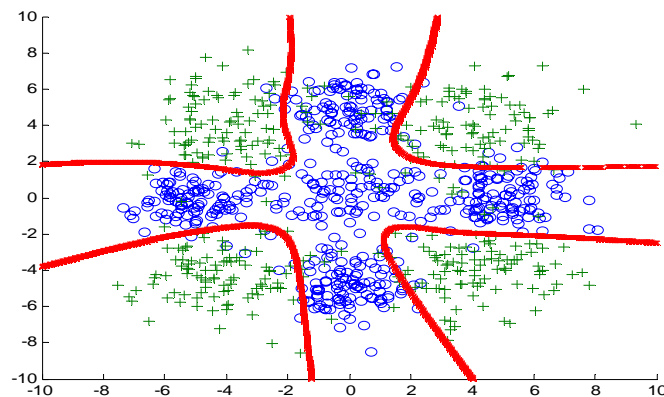


Figure II.10 La séparation en deux classes avec réseau *MLP* entraîné par l'AG

La figure (II.11) représente la sortie du réseau selon les coordonnées de la grille.

On voit bien sur cette figure que la sortie est entre -1 et +1, qui correspond à la sortie d'une fonction tangente hyperbolique.

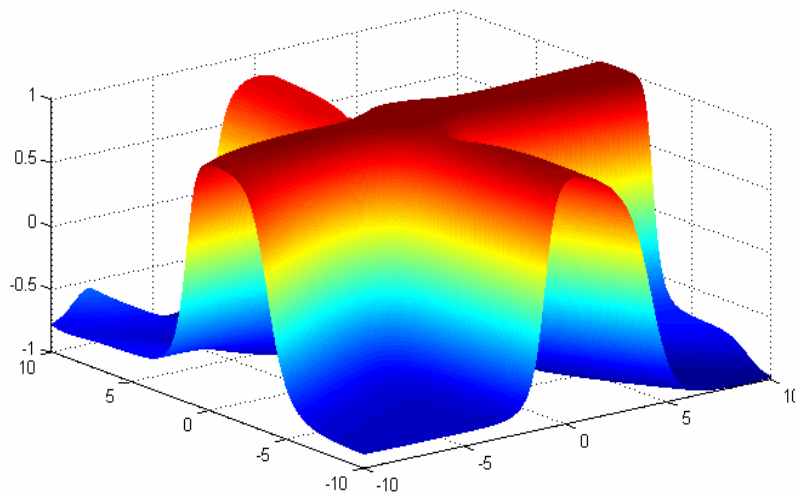


Figure II.11 Sortie du MLP selon les coordonnées de la grille.

L'évolution de l'erreur cumulée est illustrée sur la figure suivante.

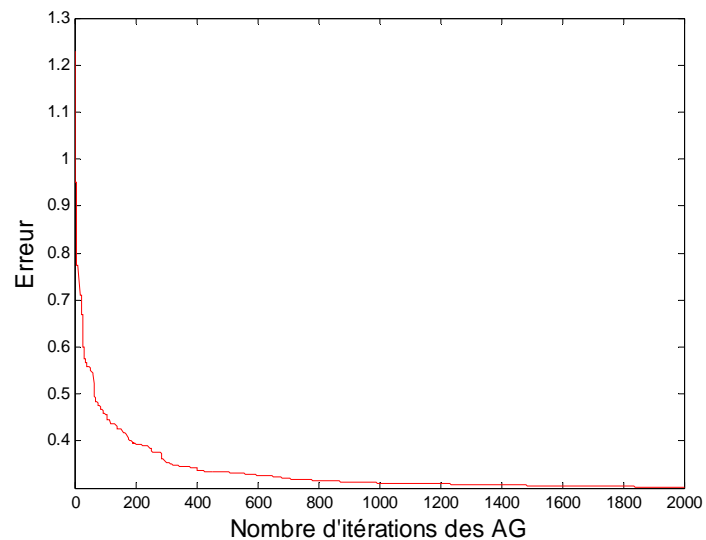


Figure II.12 L'erreur en fonction de nombre d'itération

L'erreur du réseau diminue en fonction du nombre d'itérations des AG, elle se stabilise après un seuil. L'algorithme converge après 800 itérations.

II.11 Conclusion

Ce chapitre a établi les fonctions nécessaires à la compréhension des algorithmes génétiques, de leurs mécanismes et leurs puissances. Ces algorithmes classés parmi les méthodes stochastiques, s'inspirent de l'évolution génétique des espèces, plus précisément du principe de la sélection naturelle.

Le domaine d'application des algorithmes génétiques est assez large. En effet, depuis leurs adaptations, ces méthodes connaissent une expansion considérable. Les algorithmes génétiques dans l'application qu'on leur a soumis ont montré leur grande souplesse et leur facilité d'utilisation. Cette science à part entière a trouvé des applications pratiques que ce soit dans l'industrie ou dans d'autres domaines. Dans notre travail, nous les appliquerons à l'optimisation des réseaux de neurones pour l'identification des systèmes non linéaires. Ceci fera l'objet du chapitre suivant.

**CHAPITRE III : IDENTIFICATION
PAR RESEAUX DE NEURONES
OPTIMISES PAR LES AG**

III Introduction

Dans le chapitre précédent, nous avons exposé une application des réseaux de neurones optimisés par les AG à la classification des données. Dans ce présent chapitre, nous nous servirons des différents algorithmes d'apprentissage de ces réseaux afin qu'ils puissent être utilisés par la suite pour l'identification des systèmes. Dans un premier temps, nous avons utilisé les réseaux de neurones simples, ensuite les réseaux de neurones optimisés par les AG.

III.1 Identification par réseaux de neurones

L'identification d'un système non linéaire, dont la structure peut être inconnue, consiste à trouver un ensemble d'équations non linéaires qui décrivent le mieux possible son comportement. Plusieurs approches ont été proposées pour la modélisation de ces systèmes. Cependant, des méthodes générales et efficaces ne sont pas encore disponibles. L'identification neuronale se présente alors comme une alternative.

III.1.1 Identification

L'idée de base de l'approche neuronale est de substituer aux modèles paramétriques classiques des modèles neuronaux dont les poids des connexions (qui constituent dans ce cas, les paramètres à estimer) sont ajustés en utilisant une loi d'apprentissage appropriée en utilisant les données entrées-sorties du système [3].

III.1.2 Type d'identification

En s'inspirant des techniques d'identification des systèmes linéaires, deux structures d'identification, en utilisant l'approche neuronale, ont été proposées [3].

III.1.2.1 Identification série- parallèle

La structure générale d'identification série-parallèle d'un système non linéaire est décrite par l'équation aux différences non linéaire :

$$y(t+1) = f[y(t), y(t-1), \dots, y(t-n+1); u(t-1), \dots, u(t-m+1)] \quad (3.1)$$

Avec :

$$u(t) = [u_1(t), \dots, u_r(t)] \quad (3.2)$$

$$y(t) = [y_1(t), \dots, y_m(t)] \quad (3.3)$$

Où $u(t)$ est l'entrée du système à identifier, $y(t)$ sa sortie et f une fonction non linéaire. A l'instant $(t+1)$, la sortie du système dépend des n valeurs passées de la sortie ainsi que des m valeurs passées de l'entrée, simultanément.

Le modèle neuronal placé en parallèle avec le système est de comportement entrée-sortie décrit par l'expression suivante :

$$y^M(t+1) = \hat{f}[x(t)] \quad (3.4)$$

Où $x(t)$ est le vecteur d'entrée du réseau et $y^M(t+1)$ sa sortie.

L'entrée du réseau à un instant donné est :

$$x(t) = [y(t), y(t-1), \dots, y(t-n+1); u(t), u(t-1), \dots, u(t-m+1)] \quad (3.5)$$

Le but de la procédure d'apprentissage est d'entraîner le réseau pour que la transformation non linéaire \hat{f} soit une approximation de la fonction f .

Dans cette structure, la prise en compte de l'ordre du système s'effectue par l'intermédiaire de lignes de retard mémorisant les valeurs passées de la sortie et de la commande dont dépendra la sortie courante du système.

La différence entre la sortie y et la sortie y^M que le modèle neuronal prédit est utilisée par un algorithme d'apprentissage pour ajuster les poids des connexions du modèle.

III.1.2.2 Identification parallèle

La relation entrées-sorties est donnée par l'expression suivante :

$$y^M(t+1) = \hat{f}[x(t)] \quad (3.6)$$

Où :

$$x(t) = [y^M(t), y^M(t-1), \dots, y^M(t-n+1); u(t), u(t-1), \dots, u(t-m+1)] \quad (3.7)$$

$x(t)$ est le vecteur d'entrée du modèle neuronal.

Dans ce cas, la sortie courante $y^M(t+1)$ du modèle neuronal dépend des n valeurs passées de sa sortie, au lieu des valeurs de la sortie du système dans le cas de la structure précédente.

La différence entre la sortie du système y et celle du modèle est utilisée à travers un algorithme d'apprentissage pour ajuster les paramètres du modèle (réseau de neurones).

L'utilisation de cette structure d'identification souffre des problèmes de convergence et de stabilité. Par conséquent, l'utilisation de la structure série-parallèle est préférable [3].

L'identification peut être définie comme l'obtention d'un modèle mathématique pour représenter le comportement dynamique d'un processus.

On peut distinguer deux types de modèle :

- Modèle de connaissances (équations différentielles) mais :
 - ✓ Souvent trop idéalisé par rapport au système réel.
 - ✓ Difficulté à connaître la valeur des paramètres physiques.
 - ✓ Lourdeur des calculs pour la résolution des équations différentielles.
- Modèle mathématique simple (en général linéaire) avec des paramètres sans signification physique.

III.2 Perceptron multicouche

Le perceptron multicouche est organisé en couches successives. Il comprend une couche d'entrée, une couche de sortie et une ou plusieurs couches intermédiaires appelées couches cachées.

III.2.1 Mise en œuvre du MLP

La mise en œuvre des réseaux de neurones comporte à la fois une partie conception, dont l'objectif est de permettre de choisir la meilleure architecture possible, et une partie de calcul numérique, pour réaliser l'apprentissage d'un réseau de neurones. Dans le cas général, un MLP peut posséder un nombre de couches quelconque, mais en vue de perfectionner le fonctionnement du MLP d'un côté et de minimiser au maximum le temps de calcul d'autre part, on doit chercher une architecture optimale au point de vue nombre de couche, et nombre de neurones par couche.

A partir d'une architecture de réseau de neurones donnée et des exemples disponibles (la base d'apprentissage), on détermine les poids optimaux, par l'algorithme de la rétropropagation des erreurs, pour que la sortie du modèle s'approche le plus possible du fonctionnement désiré.

III.2.2 Propriétés fondamentales du MLP

La famille des réseaux de neurones à une couche de neurones cachés possède la propriété d'apprentissage parcimonieuse. Cela signifie qu'elle est capable d'approcher n'importe quelle fonction bornée et suffisamment régulière, en utilisant moins de paramètres ajustables que les familles de fonctions usuelles telles que les polynômes [3].

Dans l'optique d'une modélisation statique, l'intérêt de la parcimonie est alors de limiter le nombre d'exemples nécessaires pour obtenir une bonne estimation de la fonction désirée.

III.2.3 Equations du réseau

On utilisera les notations suivantes pour décrire le fonctionnement du perceptron multicouche :

l : numéro de couches du réseau avec la couche 1 qui représente la couche d'entrée et la couche L qui est la couche de sortie.

(L : nombre total de couches, l : indice de couche).

N_l : Nombre de neurones présents dans la couche l .

$a_j^l(k)$: Entrée au $j^{\text{ème}}$ neurone de la couche l à l'instant k .

$s_j^l(k)$: Sortie du $j^{\text{ème}}$ neurone de la couche l à l'instant k .

$w_{ij}^l(k)$: Poids de la connexion allant du $i^{\text{ème}}$ neurone de la couche l vers le $j^{\text{ème}}$ neurone de la couche $l+1$, à l'instant k .

Le perceptron multicouche est régi par les équations suivantes :

L'entrée du $j^{\text{ème}}$ neurone dans la couche l à l'instant k est donnée par :

$$a_j^l(k) = \sum_{i=1}^{N_{l-1}} w_{ji}^{l-1}(k) x_i^{l-1}(k) \quad (3.8)$$

Sa sortie est donnée par :

$$s_j^l(k) = g(a_j^l(k)) \quad 1 \leq l \leq L \quad (3.9)$$

g étant la fonction d'activation.

L'entrée du $j^{\text{ème}}$ neurone dans la couche de sortie est :

$$x_j^L(k) = \sum_{i=1}^{N_{L-1}} w_{ji}^{L-1}(k) s_i^{L-1}(k) \quad (3.10)$$

III.2.4 Apprentissage du réseau MLP

L'apprentissage est le processus par lequel un réseau de neurones s'auto adapte en vue d'approcher une sortie désirée.

L'algorithme de rétropropagation (Back Propagation) est l'algorithme le plus utilisé dans la tâche d'apprentissage des réseaux de neurones. Il est issu de l'algorithme de descente du gradient stochastique qui a pour objectif la minimisation d'une fonction coût de la forme :

$$E_{rr} = \frac{1}{2} \sum_{k=1}^N \sum_{i=1}^m (y_i(k) - y_{i,d}(k))^2 \quad (3.11)$$

Avec :

$y_i(k)$: $i^{\text{ème}}$ sortie du réseau neuronal à l'instant k .

$y_{i,d}(k)$: Sortie désirée correspondante à $y_i(k)$.

N : représente la longueur de la séquence d'entraînement.

m : le nombre total des neurones de la couche de sortie.

Les poids du réseau de neurones seront modifiés suivant la relation suivante :

$$\Delta w(k) = -\eta \frac{\partial E_{rr}}{\partial w(k)} \quad (3.12)$$

η étant le taux ou le pas d'apprentissage ($0 < \eta < 1$).

Les poids sont alors adaptés avec l'équation suivante appelée aussi règle Delta [3] :

$$w(k+1) = w(k) + \Delta w(k) \quad (3.13)$$

Pour le cas du réseau de neurones multicouches. Le critère quadratique à minimiser est donnée par :

$$E_{rr} = \frac{1}{2} \sum_{k=1}^N (s_j^L(k) - s_j^d(k))^2 \quad (3.14)$$

$s_j^L(k)$: Sortie du réseau de neurones.

$s_j^d(k)$: Sortie désirée correspondante au $j^{\text{ème}}$ neurone de sortie.

Nous avons donc besoin de calculer les dérivées de ce critère d'erreur par rapport aux poids de connexions. En utilisant l'algorithme de la rétro-propagation et la règle de chaînage, on obtient les dérivées suivantes [3] :

L'erreur à la couche de sortie :

$$e_j^L(k) = (s_j^L(k) - s_j^d(k)) g'(x_j^L(k)) \quad (3.15)$$

Les erreurs aux couches cachées :

$$e_j^l(k) = g'(x_j^l(k)) \sum_{p=1}^{N_{l+1}} e_p^{l+1}(k) w_{jp}^l(k) \quad (3.16)$$

L'adaptation des poids se fait selon les équations suivantes :

$$w_{ij}^l(k+1) = w_{ij}^l(k) - \eta e_j^{l+1}(k) s_i^l(k) \quad (3.17)$$

III.2.5 Application du réseau MLP à l'identification

Dans ce paragraphe, nous montrerons comment entraîner un réseau MLP à travers l'identification d'un système dynamique [3].

La fonction coût utilisée est de la forme suivante :

$$E = \frac{1}{2} \sum_{k=1}^N e_1^2(k) = \frac{1}{2} \sum_{k=1}^N [y(k) - \hat{y}(k)]^T [y(k) - \hat{y}(k)] \quad (3.18)$$

N : Longueur de la séquence d'apprentissage.

$y(k)$: Sortie du système.

$\hat{y}(k)$: Sortie du modèle neuronal.

On distingue deux types d'identification :

- Identification série-parallèle.
- Identification parallèle.

Ces deux types d'identification sont schématisés dans les figures (III.2) et (III.3) suivantes :

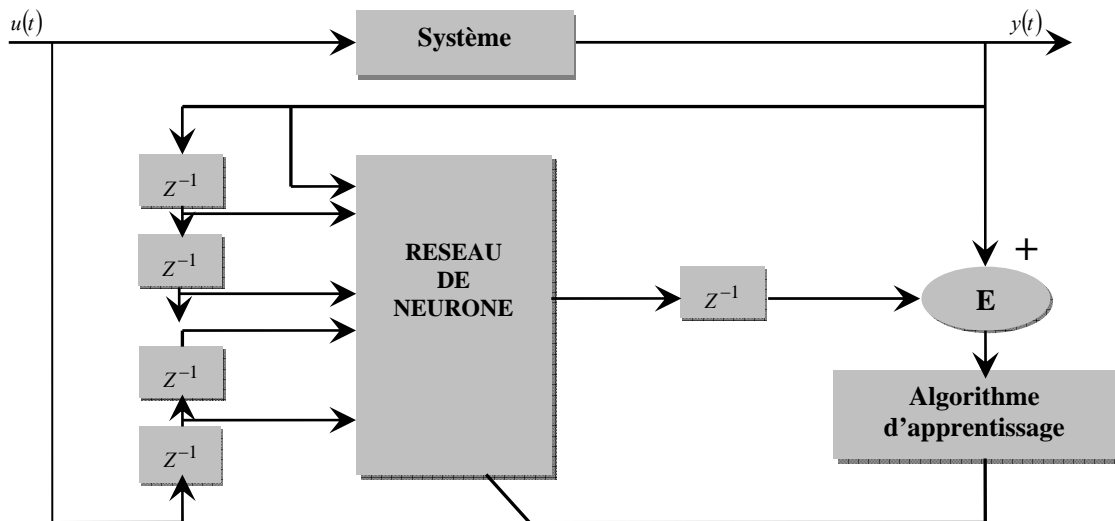


Figure III.2 Structure d'identification série-parallèle

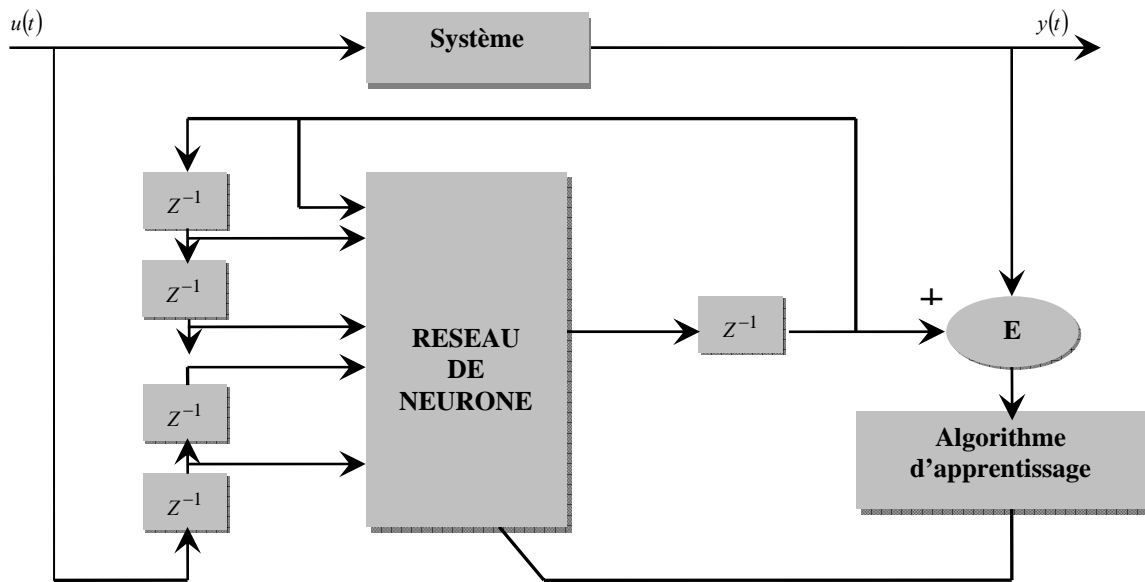


Figure III.3 Structure d'identification parallèle

Le réseau neuronal MLP utilisé lors des tests des différents programmes, est constitué de deux entrées, une couche cachée possédant huit neurones et un seul neurone dans la couche de sortie. Le taux d'apprentissage est choisi empiriquement : $\eta = 0.2$.

La fonction d'activation utilisée pour la couche d'entrée, ainsi que la couche cachée est une

fonction unipolaire :

$$f(x) = c_1 \frac{1}{1 + \exp(-k_1 x)} \quad (3.19)$$

Cependant pour la couche de sortie, on a utilisé la fonction bipolaire :

$$f(x) = c_2 \frac{1 - \exp(-k_2 x)}{1 + \exp(-k_1 x)} \quad (3.20)$$

c_1, c_2 Représentent l'amplitude de la fonction, et k_1, k_2 contrôlent la pente.

Les coefficients précédents sont tous fixés à 1.

Afin de valider et de prouver l'efficacité des réseaux de neurones dans le domaine de l'identification, nous avons pris deux exemples de systèmes non linéaires différents dont voici les équations qui les régissent :

Exemple 1 :

Le système à identifier est un système non linéaire de second ordre :

$$y(k) = 0.35 \left[\frac{y(k-1)y(k-2)[y(k-1) + 2.5]}{1 + y^2(k-1) + y^2(k-2)} + u(k) \right]$$

Exemple 2 :

Le système à identifier est un système non linéaire de premier ordre :

$$y(k) = \frac{y(k-1)}{1 + y^2(k-1)} + u^3(k-1)$$

Ces deux systèmes seront utilisés pour les trois types de réseaux de neurones et pour chaque type d'identification et sont entraînés par leurs algorithmes d'apprentissage en suivant les étapes pour chaque algorithme.

A la fin de la phase d'apprentissage, on présente au système ainsi qu'au réseau une séquence d'entrée test qui validera et montrera est ce qu'on a trouvé les bons poids pour approcher la sortie du système. Cette entrée test est de la forme suivante [2], [3]:

$$u(k) = \begin{cases} \sin(\pi k / 10) & 0 < k \leq 250 \\ +1 & 250 < k \leq 500 \\ -1 & 500 < k \leq 750 \\ 0.3 \sin(\pi k / 25) + 0.1 \sin(\pi k / 32) + 0.6 \sin(\pi k / 10) & 750 < k \leq 1000 \end{cases}$$

Les résultats obtenus sont illustrés sur les figures suivantes :

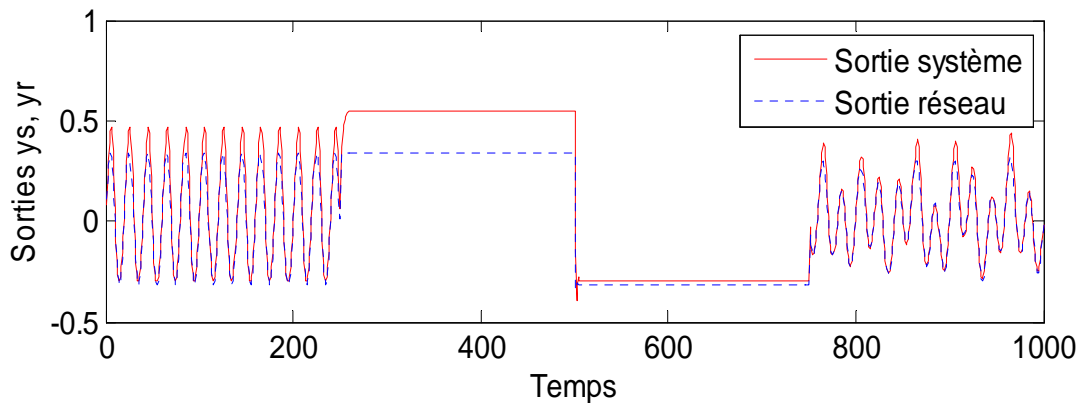
a) Exemple 1

Figure III.4 Identification série-parallèle

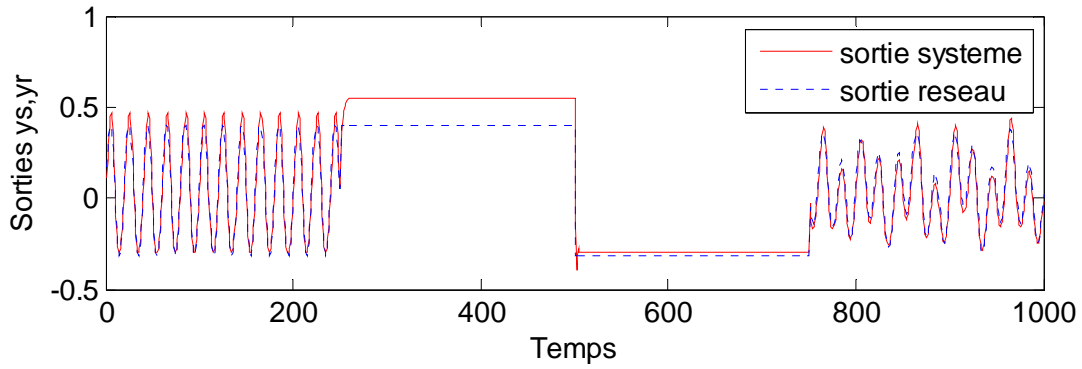


Figure III.5 Identification parallèle

On remarque que l'écart entre les deux sorties est presque négligeable, l'erreur quadratique moyenne est égale à $1.3 \cdot 10^{-2}$, pour l'identification série-parallèle, et de $6.8 \cdot 10^{-3}$, pour l'identification parallèle.

b) Exemple2

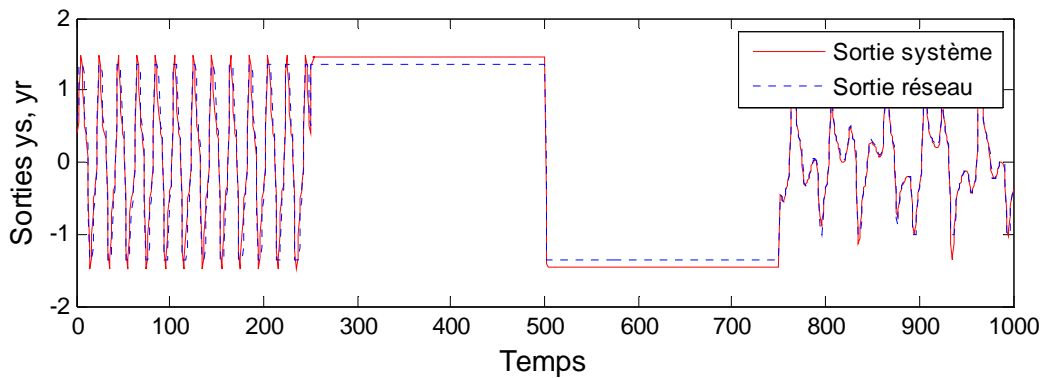


Figure III.6 Identification série-parallèle

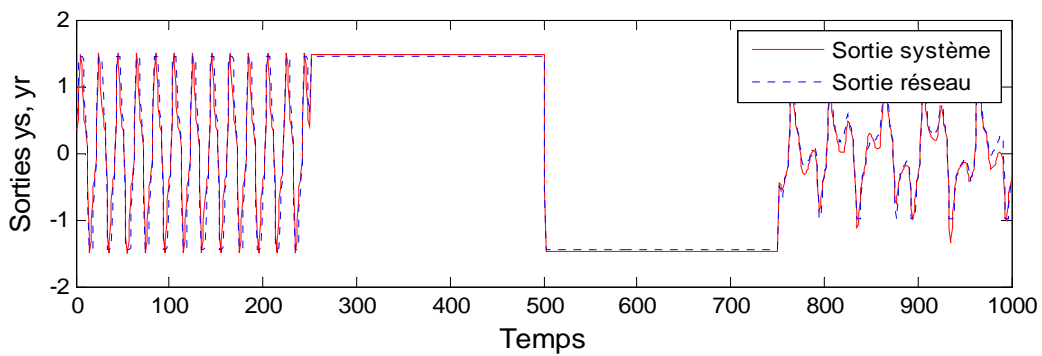


Figure III.7 Identification parallèle

L'écart entre les deux sorties est très faible, mais un peu plus petit par rapport au premier exemple illustré dans les figures (III.4), (III.5). L'erreur quadratique moyenne est égale à $6.0587e-003$, pour l'identification série-parallèle, et de $4.8728e-003$, pour l'identification parallèle.

III.3 Réseau de neurone à mémoire (MNN)

Ce réseau a été introduit par Sastry et G.Santharam dans [2], [3].

Ce type de réseau ressemble beaucoup à un réseau multicouche statique excepté que :

- La couche d'entrée est une couche de deux neurones.
- Chaque neurone situé dans la couche d'entrée, ou dans les couches cachées, est relié à une unité mémoire appelée neurone mémoire, qui a comme entrée sa propre sortie retardée. Ainsi, que la sortie retardée du neurone auquel il est connecté (ces deux entrées sont pondérées).
- Chaque neurone dans la couche de sortie possède plus d'un neurone mémoire.
- Chaque neurone reçoit comme entrées les sorties des neurones ordinaires ainsi que les sorties des neurones mémoires.

La figure suivante indique la configuration d'un réseau de neurones à mémoires :

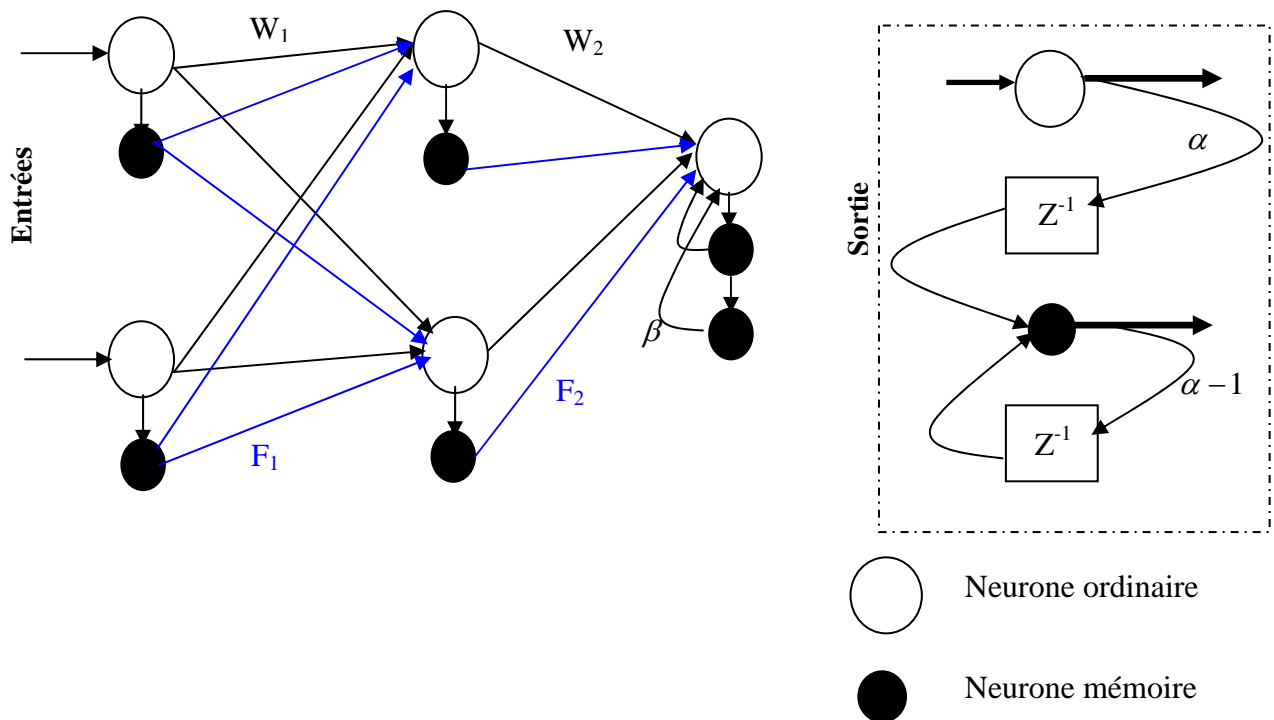


Figure III.6 Architecture de réseau a mémoire

L'intérêt de ce type de réseau réside dans sa topologie où chaque neurone du réseau possède une case mémoire dans laquelle est stockée l'information passée. Dans ce cas, l'entrée des neurones reçoit les données présentes ainsi que celle passées. De plus, les neurones dans la couche de sortie peuvent avoir plusieurs unités mémoires.

Ceci procure à la sortie une large gamme de connaissances et permet ainsi au réseau neuronal de faire face aux différentes situations.

III.3.1 Equations du réseau

On utilisera les notations suivantes pour décrire la dynamique du réseau de neurones à mémoire :

- l : numéro de couches du réseau avec la couche 1 qui représente la couche d'entrée, et la couche L est la couche de sortie.
- (L : nombre de couches, l : indice de couche)
- N_l : nombre de neurones présents dans la couche l .
- $x_j^l(k)$: entrée du $j^{\text{ème}}$ neurone de la couche l à l'instant k .
- $s_j^l(k)$: sortie du $j^{\text{ème}}$ neurone de la couche l à l'instant k .
- $v_j^l(k)$: sortie du neurone mémoire associé au $j^{\text{ème}}$ neurone de la couche l à l'instant k .
- $w_{ij}^l(k)$: poids de la connexion allant du $i^{\text{ème}}$ neurone de la couche l vers $j^{\text{ème}}$ neurone de la couche $l+1$, à l'instant k .
- $F_{ij}^l(k)$: poids de la connexion allant du neurone mémoire associé au $i^{\text{ème}}$ neurone de la couche l , vers le $j^{\text{ème}}$ neurone de la couche $l+1$ à l'instant k .
- $\alpha_j^l(k)$: poids de la connexion allant du $j^{\text{ème}}$ neurone vers son neurone mémoire dans la couche l à l'instant k (voir figure (III.6)).
- M_i : nombre de neurones mémoires associés au $i^{\text{ème}}$ neurone dans la couche de sortie.
- $\alpha_{ij}^L(k)$: poids de la connexion allant du $(j-1)^{\text{ème}}$ neurone mémoire, vers le $j^{\text{ème}}$ neurone mémoire associés au $i^{\text{ème}}$ neurone de la couche de sortie à l'instant $k, j=1, \dots, M_j$.
- $\beta_{ij}^L(k)$: poids de la connexion allant du $j^{\text{ème}}$ neurone mémoire associé au $i^{\text{ème}}$ neurone vers le $i^{\text{ème}}$ neurone dans la couche de sortie à l'instant $k. \quad j=1, \dots, M_j$

Le réseau de neurones à mémoire est régi par les équations suivantes :

L'entrée du $j^{\text{ème}}$ neurone dans la couche l à l'instant k est donnée par :

$$x_j^l(k) = \sum_{i=1}^{N_{l-1}} w_{ji}^{l-1}(k) s_i^{l-1}(k) + \sum_{i=1}^{N_{l-1}} F_{ji}^{l-1}(k) v_i^{l-1}(k) \quad (3.21)$$

Sa sortie est donnée par :

$$s'_j(k) = g(x'_j(k)) \quad 1 \leq l \leq L \quad (3.22)$$

g étant la fonction d'activation.

La sortie de tous les neurones mémoires, excepté pour ceux de la couche de sortie est donnée par :

$$v'_j(k) = \alpha'_j(k) s'_j(k-1) + (1 - \alpha'_j(k)) v'_j(k-1) \quad (3.23)$$

Le paramètre $\alpha'_j(k)$ permet de contrôler la dynamique du neurone. En effet, suivant cette dernière équation, on peut voir que :

- Lorsque $\alpha'_j(k)$ est proche de 1, l'état du neurone mémoire dépend beaucoup plus de la sortie du neurone ordinaire. Ce qui mène à considérer les neurones mémoires comme ligne de retard.
- Quand $\alpha'_j(k)$ tend vers 0, le neurone mémoire devient insensible à la sortie du neurone ordinaire (i.e. nouvelles données) et son état demeure presque inchangé (dépendance complète du passé).

Pour les unités mémoire de la couche de sortie :

$$v_{ij}^L(k) = \alpha_{ij}^L(k) v_{ij-1}^L(k-1) + (1 - \alpha_{ij}^L(k)) v_{ij}^L(k-1) \quad (3.24)$$

$$v_{i0}^L(k) = s_i^L(k) \quad (3.25)$$

L'entrée du $j^{\text{ème}}$ neurone dans la couche de sortie est :

$$x_j^L(k) = \sum_{i=1}^{N_{L-1}} w_{ji}^{L-1}(k) s_i^{L-1}(k) + \sum_{i=1}^{N_{L-1}} F_{ji}^{L-1}(k) v_i^{L-1}(k) + \sum_{i=1}^{M_j} \beta_{ji}^L(k) v_{ij}^L(k) \quad (3.26)$$

III.3.2 Apprentissage du réseau MNN

Vu la ressemblance du réseau MNN avec le réseau multicouche, le plus simple serait d'utiliser le même algorithme d'apprentissage. Cependant, l'aspect récurrent du réseau MNN nécessite certaine modification sur cet algorithme et un autre algorithme s'impose dont l'idée est issu de l'algorithme BP (Back Propagation) et qu'est nommé algorithme de rétropropagation dynamique DBP (Dynamic Back Propagation) [3].

Notre objectif est toujours la minimisation d'une fonction coût de la forme :

$$E = \frac{1}{2} \sum_{k=1}^T \sum_{i=1}^m (y_i(k) - y_{i,d}(k))^2 \quad (3.27)$$

Avec :

$y_i(k)$: $i^{\text{ème}}$ sortie du réseau neuronal à l'instant k .

$y_{i,d}(k)$: Sortie désirée correspondante à $y_i(k)$.

T : représente la longueur de la séquence d'entraînement.

m : le nombre total des neurones de la couche de sortie.

Les poids du réseau de neurones seront modifiés suivant la relation suivante :

$$\Delta w(k) = -\eta \frac{\partial E}{\partial w(k)} \quad (3.28)$$

η étant le taux ou le pas d'apprentissage ($0 < \eta < 1$).

Les poids sont alors adaptés avec l'équation suivante appelée aussi règle Delta [3] :

$$w(k+1) = w(k) + \Delta w(k) \quad (3.29)$$

Pour le cas du réseau de neurones à mémoire, l'algorithme de rétropropagation peut être modifié afin qu'il tienne compte des récurrences internes du réseau. Le critère quadratique à minimiser est donnée par :

$$E = \frac{1}{2} \sum_{k=1}^T \sum_{i=1}^m (s_j^L(k) - s_j^d(k))^2 \quad (3.30)$$

$s_j^L(k)$: Sortie du réseau de neurones.

$s_j^d(k)$: Sortie désirée correspondante au $j^{\text{ème}}$ neurone de sortie.

Nous avons besoin de calculer les dérivées de ce critère d'erreur par rapport aux poids de connexions. En utilisant l'algorithme BP et la règle de chaînage, on obtient les dérivées suivantes [2], [3] :

L'erreur à la couche de sortie :

$$e_j^l(k) = g'(x_j^l(k)) \sum_{p=1}^{N_{l+1}} e_p^{l+1}(k) w_{jp}^l(k) \quad (3.31)$$

L'adaptation des poids se fait selon les équations suivantes :

$$w_{ij}^l(k+1) = w_{ij}^l(k) - \eta e_j^{l+1}(k) s_i^l(k) \quad (3.32)$$

$$F_{ij}^l(k+1) = F_{ij}^l(k) - \eta e_j^{l+1}(k) v_i^l(k) \quad (3.33)$$

Les différents coefficients mémoires sont adaptés comme suit :

$$\alpha_j^l(k+1) = \alpha_j^l(k) - \eta' \frac{\partial e}{\partial v_j^l}(k) \frac{\partial v_j^l}{\partial \alpha_j^l}(k) \quad (3.34)$$

$$\alpha_{ij}^L(k+1) = \alpha_{ij}^L(k) - \eta' \frac{\partial e}{\partial v_{ij}^L}(k) \frac{\partial v_{ij}^L}{\partial \alpha_{ij}^L}(k) \quad (3.35)$$

$$\beta_{ij}^L(k+1) = \beta_{ij}^L(k) - \eta' e_i^L(k) v_{ij}^L(k) \quad (3.36)$$

Avec :

$$\frac{\partial e}{\partial v_j^L}(k) = \sum_{s=1}^{N_{l+1}} F_{js}^l(k) e_s^{l+1}(k) \quad (3.37)$$

$$\frac{\partial v_{ij}^L}{\partial \alpha_{ij}^L}(k) = s_j^l(k-1) - v_j^l(k-1) \quad (3.38)$$

$$\frac{\partial e}{\partial v_{ij}^L}(k) = \beta_{ij}^L(k) e_i^L(k) \quad (3.39)$$

$$\frac{\partial v_{ij}^L}{\partial \alpha_{ij}^L}(k) = v_{ij-1}^L(k-1) - v_{ij}^L(k-1) \quad (3.40)$$

III.3.3 Algorithme MNN

Les étapes suivies lors de l'identification, peuvent se résumer comme suit :

Étape 1 : choix du nombre de couches cachées ainsi que le nombre de neurones pour chaque couche.

Étape 2 : initialisation des différentes matrices de poids intervenant dans le réseau. Elles sont prises aléatoirement dans l'intervalle [-1 1]. Pour les poids des connexions associées aux neurones mémoire, ils sont pris dans l'intervalle [0 1], le choix de cet intervalle assure la convergence [27], [3].

Étape 3 : choix de la séquence d'apprentissage. La séquence choisie s'étend sur 15000 échantillons tous choisis aléatoirement dans l'intervalle [-1 1]. Le réseau est ainsi confronté à plusieurs changements intervenant dans cet intervalle.

Étape 4 : on soumet le système et le réseau neuronal à la même entrée. On calcule la sortie du système, puis à l'aide des équations suivantes, on calcule la sortie de la couche d'entrée et de la couche cachée ainsi que la sortie du réseau neuronal.

Étape 5 : le critère à minimiser est la moyenne de l'écart quadratique entre les deux sorties calculées à l'étape 4, qui est donnée par l'équation générale (III.27).

Étape 6 : l'adaptation des poids des neurones simples se fait selon les équations (III.32) et (III.33) et les coefficients mémoires sont adaptés comme suit :

$$\alpha_j^l(k+1) = \alpha_j^l(k) - \eta' \sum_{s=1}^{N_{l+1}} F_{js}^l(k) e_s^{l+1}(k) [s_j^l(k-1) - v_j^l(k-1)]$$

$$\beta_{ij}^L(k+1) = \beta_{ij}^L(k) - \eta' e_i^L(k) v_{ij}^L(k)$$

L'algorithme de la rétro-propagation est utilisé pour calculer les erreurs dans chaque couche du réseau et adapter les poids des connexions.

Les étapes 4 à 6 se répètent durant toute la période de la séquence d'entraînement. A la fin de l'entraînement, les nouveaux poids du réseau neuronal sont sauvegardés. Puis on procède à un test pour confirmer la validité des résultats obtenus.

III.3.4 Application du réseau MNN à l'identification

Dans ce paragraphe, nous montrerons comment entraîner un réseau MNN à travers l'identification d'un système dynamique [28].

La fonction coût utilisée est de la forme suivante :

$$E = \frac{1}{2} \sum_{k=1}^N e_1^2(k) = \frac{1}{2} \sum_{k=1}^N [y(k) - \hat{y}(k)]^T [y(k) - \hat{y}(k)] \quad (3.41)$$

N : longueur de la séquence d'apprentissage.

$y(k)$: Sortie du système.

$\hat{y}(k)$: Sortie du modèle neuronal.

Ces deux types d'identification sont schématisés dans les figures (III.2) et (III.3) précédentes. Le réseau neuronal récurrent MNN utilisé lors des simulations, est constitué d'une seule couche cachée possédant huit neurones et un seul neurone présent dans la couche de sortie auquel on associe trois neurones mémoires. Les taux d'apprentissage sont choisis empiriquement : $\eta = 0.2$ et $\eta' = 0.1$.

La fonction d'activation utilisée pour la couche d'entrée, ainsi que la couche cachée est une

fonction unipolaire : $f(x) = c_1 \frac{1}{1 + \exp(-k_1 x)}$

Cependant, pour la couche de sortie, on a utilisé la fonction bipolaire :

$$f(x) = c_2 \frac{1 - \exp(-k_2 x)}{1 + \exp(-k_1 x)}$$

c_1, c_2 représentent l'amplitude de la fonction, et k_1, k_2 contrôlent la pente.

Les coefficients précédents sont tous fixés à 1.

Les résultats obtenus sont illustrés sur les figures suivantes :

a) Exemple 1

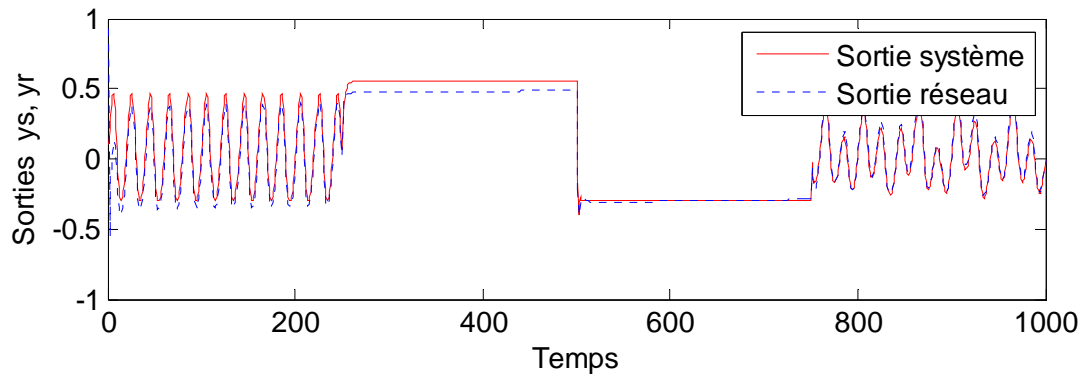


Figure III.8 Identification série-parallèle

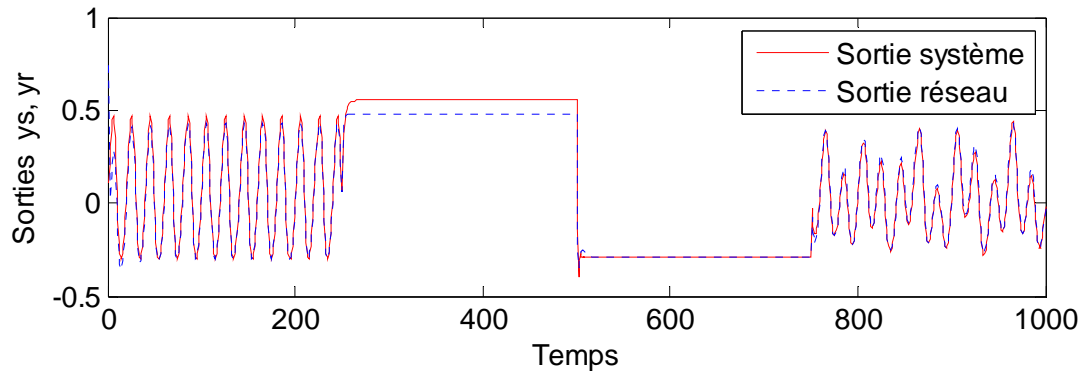


Figure III.9 Identification parallèle

L'erreur quadratique moyenne dans ce cas est de $4.6 \cdot 10^{-3}$ pour l'identification série- parallèle et de $2.1 \cdot 10^{-3}$ pour l'identification parallèle.

b) Exemple 2

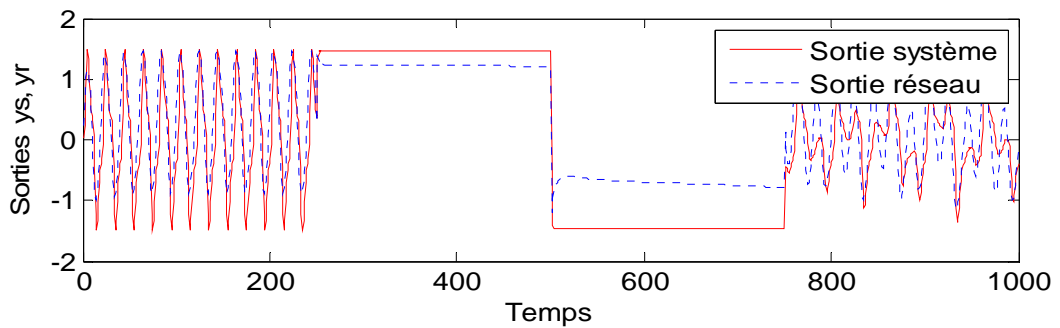


Figure III.10 Identification série-parallèle

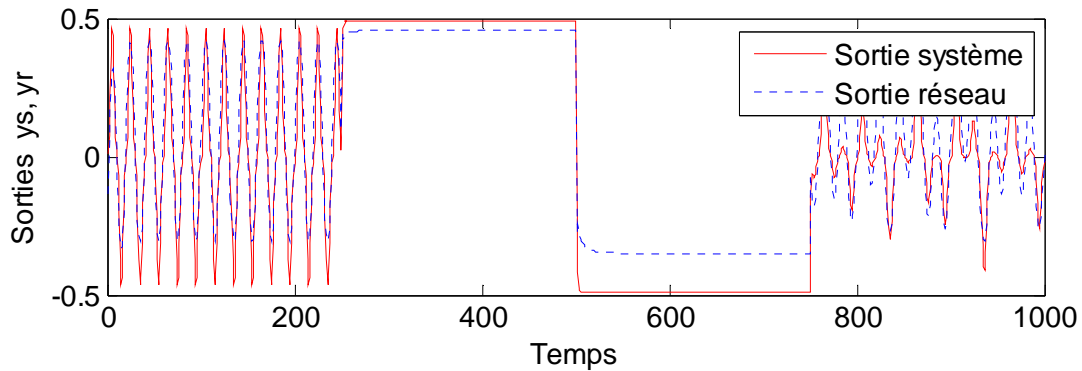


Figure III.11 Identification parallèle

Après plusieurs tests sur ce réseau avec cet exemple on a obtenu ce bon résultat. L'erreur quadratique moyenne est égale à $5 \cdot 10^{-3}$ pour l'identification serie-parallèle, et de $1,08 \cdot 10^{-2}$ pour l'identification parallèle.

Les résultats obtenus montrent clairement l'efficacité des réseaux récurrents dans le domaine de l'identification. Le réseau MNN, bien qu'il ait une architecture presque identique à celle du réseau statique MLP, se caractérise cependant de ce dernier par les avantages qu'il possède en raison de son aspect dynamique. En effet, l'emploi du réseau MLP dans l'identification par exemple, nécessite la connaissance de l'ordre du système ainsi que l'ajout de neurones supplémentaires afin d'introduire le comportement dynamique du système.

Par contre, le réseau MNN est plus souple à manipuler en raison de sa dynamique interne, ce qui lui permet de ne pas prendre en compte l'ordre du système ou l'addition de neurones complémentaires dans sa couche d'entrée.

III.4 Réseau neuronal récurrent à temps discret DTRNN

Une autre topologie de réseau récurrent étudiée dans notre travail, est un réseau analogue au réseau de Hopfield. Il s'agit de l'architecture neuronale récurrente à temps discret (Discrete Time Recurrent Neuronal Network).

La figure (III.7) montre l'architecture d'un réseau DTRNN :

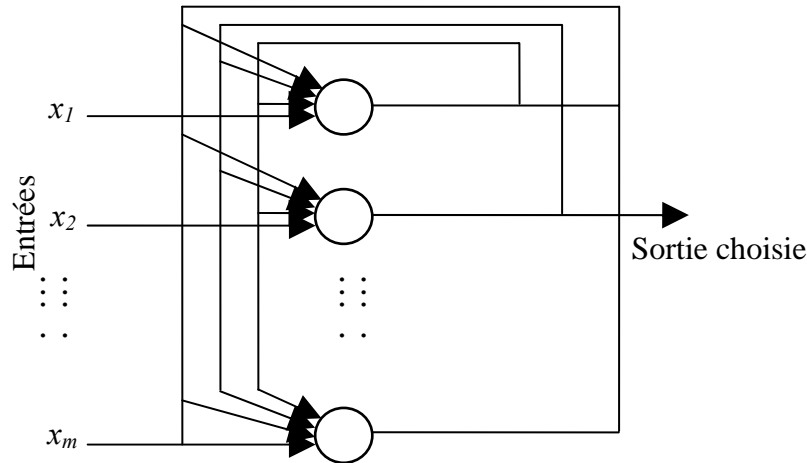


Figure III.7 Architecture du réseau DTRNN à une seule sortie

Parmi tous les états du réseau, on choisit une ou plusieurs sortie(s) selon le besoin. Dans notre cas, on a choisi la deuxième variable d'état comme étant la sortie.

III.4.1 Equations du réseau

Les équations régissant ce type de réseau sont de la forme :

$$u_i(k+1) = f \left[\sum_{j=1}^m w_{ij} u_j(k) + x_i(k) \right] \quad (3.42)$$

Si on considère que les entrées x_i sont aussi pondérées, l'équation (III.42) peut être réécrite sous la forme suivante :

$$u_i(k+1) = f \left[\sum_{j=1}^{n+m} w_{ij} u_j(k) \right] \quad (3.43)$$

Avec :

$$u_j(k) = \begin{cases} 1 & j = 0 \\ u_j(k) & j = 1, \dots, n \\ x_{j-n}(k) & j = n+1, \dots, n+m \end{cases} \quad (3.44)$$

Où :

$u_i(k+1)$: $i^{\text{ème}}$ variable d'état du réseau ($j = 1, \dots, n$).

$x_j(k)$: $j^{\text{ème}}$ entrée externe ($j = 1, \dots, m$).

n : nombre de neurones dans le réseau.

m : dimension du vecteur d'entrée x .

$u_0(k) = 1$: Entrée seuil.

Les sorties sont choisies à partir des variables d'état.

III.4.2 Apprentissage du réseau DTRNN

Un autre algorithme utilisé tout aussi populaire que l'algorithme de rétropropagation est l'algorithme de l'apprentissage en temps réel (Real Time Recurrent Learning). Cet algorithme est très efficace puisqu'il permet un calcul récursif exact du gradient. Le taux d'apprentissage pris suffisamment petit, l'adaptation se fait à chaque itération (en temps réel) d'où l'appellation de l'algorithme [29].

L'algorithme RTRL peut être formulé de la manière suivante :

Soit le critère suivant à minimiser :

$$J_p(w) = \frac{1}{2} \sum_{k=1}^{K_p} \sum_{j \in \Omega} [u_j(k) - u_j^d(k)]^2 \quad (3.45)$$

$J_p(w)$: Le critère quadratique de l'erreur à travers la $P^{\text{ème}}$ séquence.

K_p : Longueur de la $P^{\text{ème}}$ séquence d'entraînement.

Ω : Ensemble des neurones de sortie du réseau.

$u_j(k)$: Sortie du $j^{\text{ème}}$ neurone.

$u_j^d(k)$: Valeur désirée correspondante à $u_j(k)$.

La loi d'adaptation est la suivante :

$$w_{ji}(k+1) = w_{ji}(k) - \mu \frac{\partial J_p}{\partial w_{ji}(k)} \quad (3.46)$$

μ est le taux d'apprentissage.

Le gradient est exprimé comme suit :

$$\frac{\partial J_p}{\partial w_{ji}(k)} = \frac{\partial J_p}{\partial u_h(k)} \frac{\partial u_h(k)}{\partial w_{ji}(k)} \quad (3.47)$$

$$= \sum_{k=1}^{K_p} \sum_{j \in \Omega} [u_j(k) - u_j^d(k)] \frac{\partial u_h(k)}{\partial w_{ji}(k)} \quad (3.48)$$

Calcul du terme $\frac{\partial u_h(k)}{\partial w_{ji}(k)}$

Posons $P_{ji}^h(k) = \frac{\partial u_h(k)}{\partial w_{ji}(k)}$

$$P_{ji}^h(k) = \frac{\partial}{\partial w_{ji}(k)} f \left[\sum_{\alpha=0}^{n+m} w_{h\alpha} u_{\alpha}(k-1) \right] = \frac{\partial}{\partial w_{ji}(k)} f[S_h(k)] \quad (3.49)$$

$$= f'[S_h(k)] \left[\sum_{\alpha=0}^{n+m} \frac{\partial}{\partial w_{ji}(k)} [w_{h\alpha} u_{\alpha}(k-1)] \right] \quad (3.50)$$

$$= f'[S_h(k)] \left[\sum_{\alpha=0}^{n+m} \left(\frac{\partial w_{h\alpha}}{\partial w_{ji}(k)} u_{\alpha}(k-1) + w_{h\alpha} \frac{\partial u_{\alpha}(k-1)}{\partial w_{ji}(k)} \right) \right] \quad (3.51)$$

$$= f'[S_h(k)] \left[\sum_{\alpha=0}^{n+m} \delta_{hj} \delta_{\alpha i} u_{\alpha}(k-1) + \sum_{\alpha=0}^{n+m} w_{h\alpha} \frac{\partial u_{\alpha}(k-1)}{\partial w_{ji}(k)} \right] \quad (3.52)$$

$$= f'[S_h(k)] \left[\delta_{hj} u_i(k-1) + \sum_{\alpha=0}^{n+m} w_{h\alpha} \frac{\partial u_{\alpha}(k-1)}{\partial w_{ji}(k)} \right] \quad (3.53)$$

Rappelons que le terme $u_{\alpha}(k-1)$ représente toutes les entrées du réseau neuronal.

Ainsi, le terme $\frac{\partial u_{\alpha}(k-1)}{\partial w_{ji}(k)}$ sera nul pour les entrées seuil ainsi que pour les entrées externes

puisque ces entrées ne dépendent pas des poids w_{ji} . L'équation (3.53) deviendra donc :

$$P_{ji}^h(k) = f'[S_h(k)] \left[\delta_{hj} u_i(k-1) + \sum_{\beta=1}^n w_{h\beta} \frac{\partial u_{\beta}(k-1)}{\partial w_{ji}(k)} \right] \quad (3.54)$$

$$P_{ji}^h(k) = f' \left[\sum_{\alpha=0}^{n+m} w_{h\alpha} u_{\alpha}(k-1) \right] \left[\delta_{hj} u_i(k-1) + \sum_{\beta=1}^n w_{h\beta} P_{ji}^{\beta}(k-1) \right] \quad (3.55)$$

δ_{hj} : est la fonction de Kronecker ($\delta_{hj}=1$ si $h=j$ et $\delta_{hj}=0$ si $h \neq j$).

Le terme $P_{ji}^h(k)$ représente la sensibilité de la sortie $u_i(k)$ par rapport aux poids à l'instant k .

III.4.3 Algorithme RTRL

La procédure complète de l'algorithme RTRL est décrite comme suit :

Etape 1 : initialiser les poids à de petites valeurs aléatoires.

Etape 2 : choisir la séquence d'entraînement.

Etape 3 : initialiser toutes les variables d'état à zéros.

Etape 4 : initialiser tous les éléments gradients à zéro : $g_{ji}(m) = 0$.

Etape 5 : calcul des sorties du réseau :

$$u_i(k) = f \left[\sum_{j=0}^{n+m} w_{ij}(m) u_j(k-1) \right] \quad i = 1, \dots, n \quad (3.56)$$

Etape 6 : calcul du terme $P_{ji}^h(k)$:

$$P_{ji}^h(k) = f' \left[\sum_{\alpha=0}^{n+m} w_{h\alpha} u_\alpha(k-1) \right] \left[\delta_{hj} u_i(k-1) + \sum_{\beta=1}^n w_{h\beta} P_{ji}^\beta(k-1) \right] \quad (3.57)$$

Etape 7 : calcul des différents gradients g_{ji} :

$$g_{ji}(m) = g_{ji}(m) + \sum_{h \in \Omega} [u_h(k) - u_h^d(k)] P_{ji}^h(k) \quad (3.58)$$

Etape 8 : adaptation des poids du réseau :

$$w_{ji}(m+1) = w_{ji}(k) - \mu g_{ji}(m) \quad (3.59)$$

On répète les étapes 5, 6, 7 et 8 jusqu'à la fin de la séquence d'entraînement.

On remarque d'après l'équation (3.57) que la stabilité $P_{ji}^h(k)$ dépend aussi de la sensibilité passée de $P_{ji}^h(k-1)$, ce qui entraîne la dépendance du gradient $\frac{\partial J_p}{\partial w_{ji}(k)}$ de son gradient passé.

Cette particularité rend l'algorithme RTRL plus précis et permet ainsi le calcul exact du gradient. Seulement cette précision engendre des inconvénients tels que : la lenteur d'exécution et l'espace mémoire considérable requise pour ces opérations [3].

De plus, prenons par exemple un réseau DTRNN possédant ' n ' neurones et ' m ' entrées externes. On aura alors pour une seule itération lors de la phase d'entraînement :

1. le calcul de ' n^n ' sensibilités liées aux neurones du réseau.
2. le calcul de ' $n^2 * m$ ' sensibilités liées aux entrées externes.
3. le calcul de ' n^2 ' sensibilités liées aux entrées seuil.

Imaginons dans ce cas, ce que cela prendra comme temps si on avait un réseau avec une dizaine de neurones entraînés sur une longueur de 10000 échantillons.

III.4.4 Application du réseau DTRNN à l'identification

De même que pour le réseau MNN, on utilisera ici le réseau récurrent DTRNN pour l'identification du système décrit auparavant.

Le réseau récurrent DTRNN utilisé possède six neurones dans son unique couche. L'une des variables d'état du réseau est choisie comme sortie. Le taux d'apprentissage est choisi empiriquement $\eta = 0.02$. Une fonction d'activation bipolaire est choisie pour les

neurones. On a opté pour une séquence d'apprentissage d'une centaine d'échantillons. Ce choix est guidé par l'accumulation du calcul du gradient qui engendre un temps de calcul très élevé dans le cas où on aurait utilisé une séquence beaucoup plus importante.

Après achèvement de l'apprentissage, on a testé le réseau neuronal avec la même séquence de test de la page 47.

Les résultats obtenus sont illustrés dans les figures suivantes :

a) Exemple 1

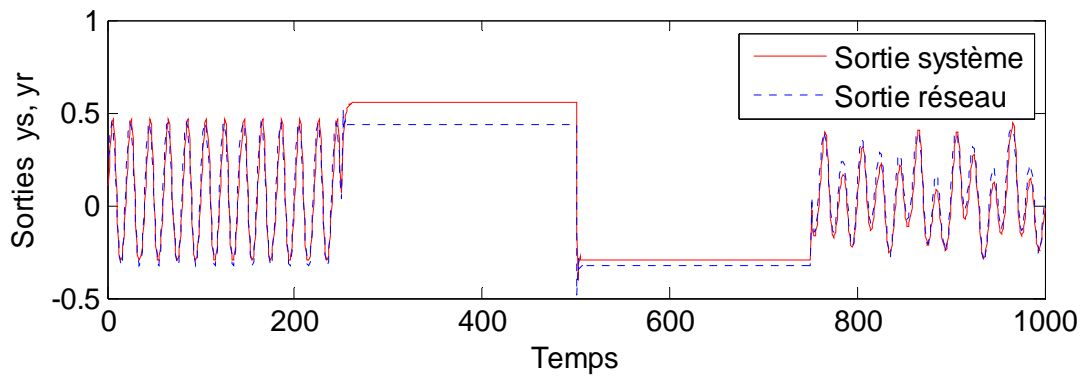


Figure III.12 Identification série-parallèle

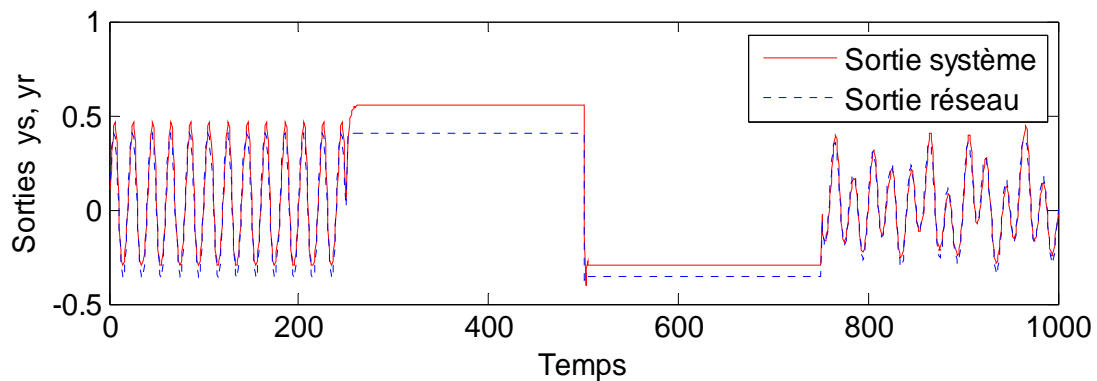


Figure III.13 Identification parallèle

Dans ce type de réseau, on remarque que l'erreur est un peu grande, elle est égale à $5.4 \cdot 10^{-3}$ pour l'identification série- parallèle, et égale à $7.7 \cdot 10^{-3}$ Pour l'identification parallèle.

b) Exemple 2

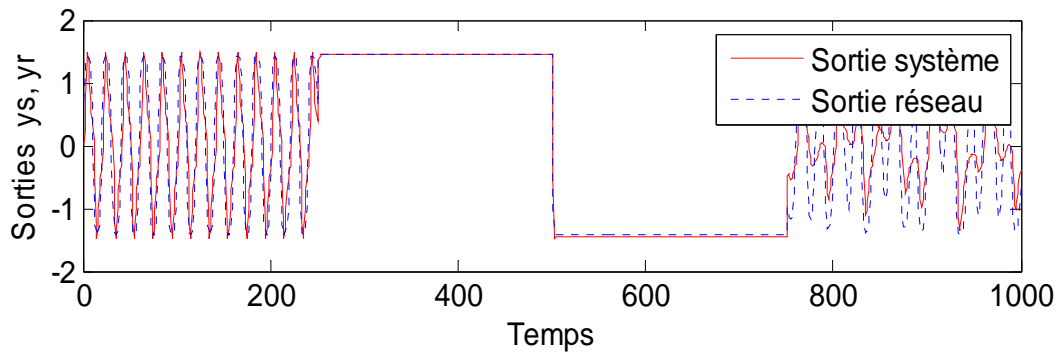


Figure III.14 Identification série-parallèle

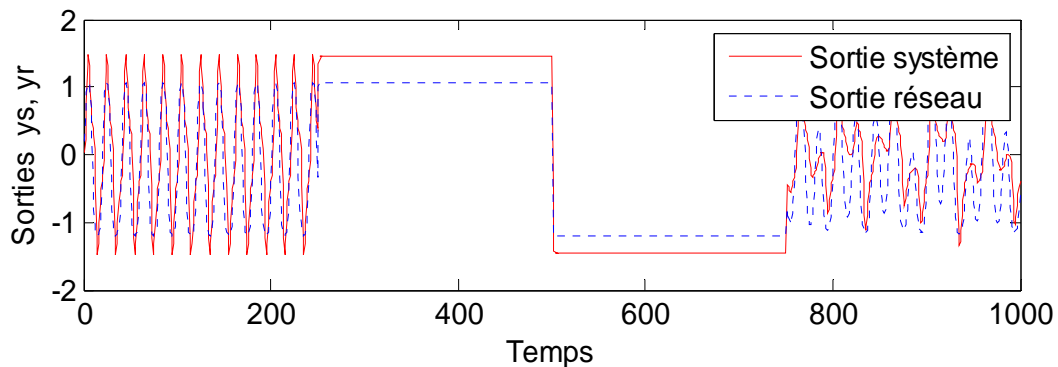


Figure III.15 Identification parallèle

Le résultat est assez remarquable, l'erreur quadratique moyenne est de $6.9 \cdot 10^{-2}$ pour l'identification série- parallèle, et de 0.1592 pour l'identification parallèle.

De même que pour les réseaux MLP et MNN, le réseau DTRNN a donné de bons résultats en ce qui concerne les deux types d'identification. Cependant, le réseau DTRNN souffre du problème des « boucles temporelles » lors de son apprentissage. En effet, l'erreur entre la sortie du réseau et la sortie désirée n'est pas directement injectée dans le réseau, mais elle est « acheminée » à travers les lignes de retard du réseau qui est ensuite traitée grâce à l'algorithme RTRL.

Malgré l'efficacité des algorithmes utilisés dans les simulations précédentes, il y a certains problèmes qui affectent le bon déroulement de l'apprentissage. Parmi ces problèmes, on peut citer le problème du choix de l'architecture, le nombre de couches cachées (pour le réseau MLP), la mise à jour ou l'adaptation des poids de connexion au sein du réseau.

Les réseaux de neurones ont besoin de données pour leurs entraînements, de façon à minimiser les erreurs entre leurs solutions et la réponse correcte attendue. Les résolutions de problèmes d'apprentissage de réseaux de neurones par descente de gradient sont caractérisées par leur incapacité remarquée de s'échapper aux optima locaux [23].

Appliquer les algorithmes génétiques aux réseaux de neurones peut permettre de résoudre ce problème. Aucune donnée n'est plus requise pour l'entraînement étant donné qu'il est effectué à travers l'évolution d'une population de réseaux. De plus, l'utilisation du croisement et de mutation sur ces réseaux assure qu'ils ne resteront pas bloqués dans un optimum local.

Dans la section qui suit nous proposons l'utilisation des AG à codage réel pour optimiser la structure du réseau de neurone et assurer ainsi son apprentissage.

L'apprentissage des réseaux de neurones par AG peut être considéré comme la minimisation d'une fonction d'erreur mesurée entre la sortie du réseau et un ensemble d'apprentissage. Cet algorithme destiné à trouver un bon individu (correspondant à un réseau de neurones). On maintient donc la diversité de la population tout en assurant une convergence globale.

III.5 Identification avec les réseaux de neurones entraînés par les AG

III.5.1 Réseau MLP

a) Exemple 1

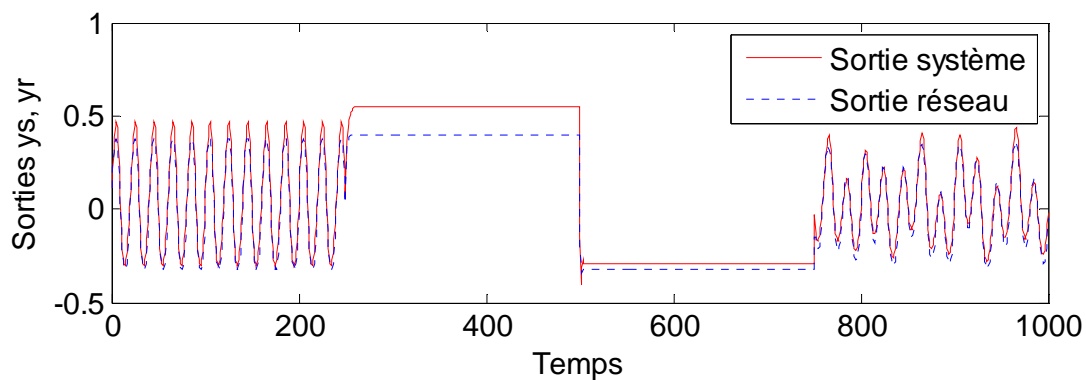


Figure III.16 Identification série- parallèle

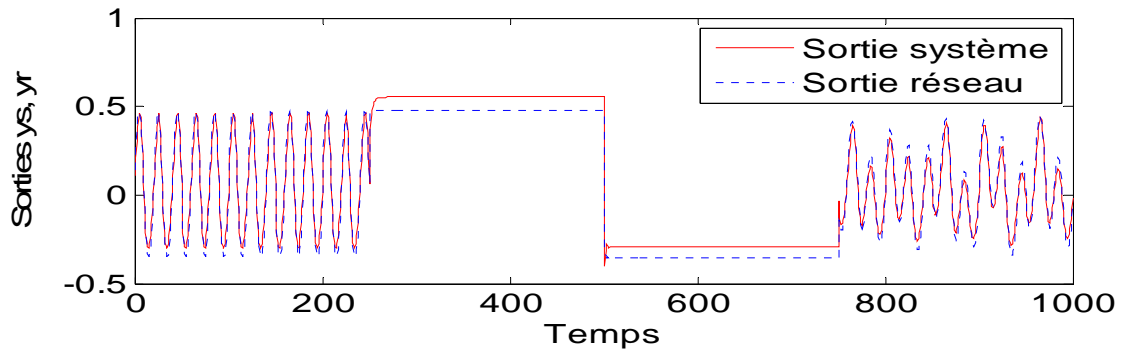


Figure III.17 Identification parallèle

D'après les résultats obtenus pour l'exemple 1, nous remarquons que la meilleure structure est celle où le nombre de neurones de la couche cachée est égale à 20 et le nombre de génération est égale à 200, ce qui correspond à une erreur quadratique de $1.32 \cdot 10^{-4}$ pour l'identification série parallèle et de $1.43 \cdot 10^{-5}$ pour l'identification parallèle.

b) Exemple 2

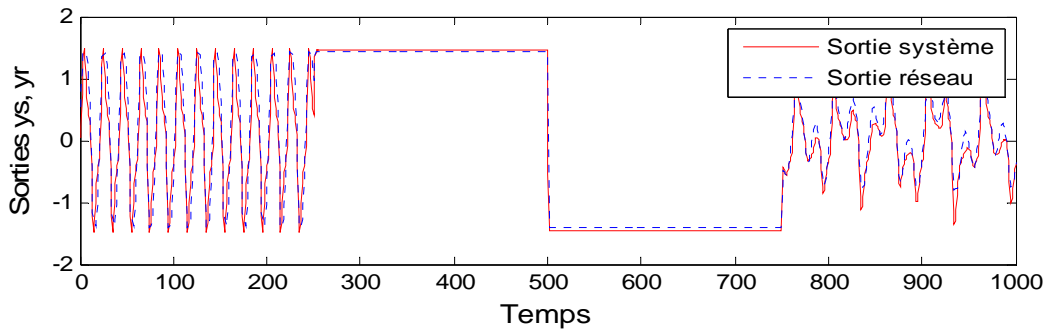


Figure III.18 Identification série-parallèle

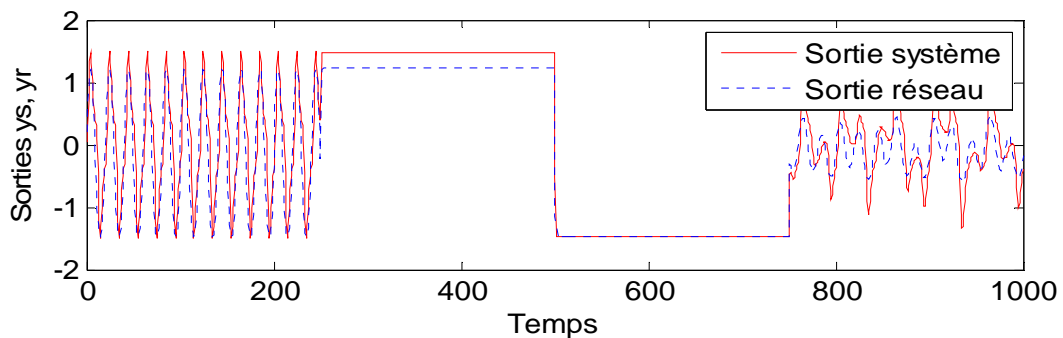


Figure III.19 Identification parallèle

De même pour l'exemple 2, d'après les résultats des figures (III.18) et (III.19), on remarque que le modèle des réseaux de neurones suit parfaitement la sortie de système, cela est obtenu pour 14 neurones dans la couche cachée et 200 générations pour les AG. L'erreur quadratique est égale 0.0123 pour l'identification série-parallèle, et égale 0.0443 pour l'identification parallèle.

III.5.2 Réseau MNN

a) Exemple 1

Les sorties du système et du réseau pour 1000 échantillons sont présentées dans les figures ci-dessous :

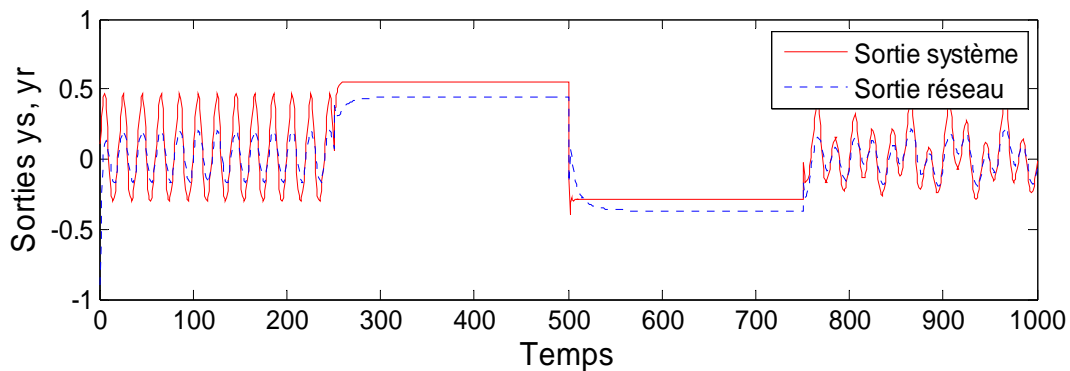


Figure III.20 Identification série- parallèle

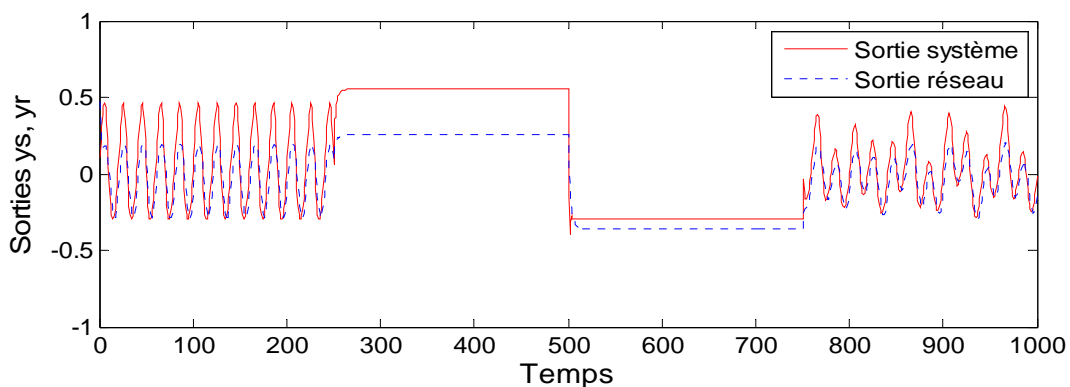


Figure III.21 Identification parallèle

L'erreur quadratique moyenne dans ces cas est égale à 0.0020 pour l'identification série-parallèle, et est égale à 0.0023 pour l'identification parallèle. On remarque qu'il y a une légère

différence entre les deux valeurs, alors que la différence du nombre de génération est beaucoup plus importante. Elle est égale à 1000 générations pour l'identification série-parallèle, et 200 générations pour l'identification parallèle, et cela pour le même nombre de neurones dans la couche cachée.

b) Exemple 2

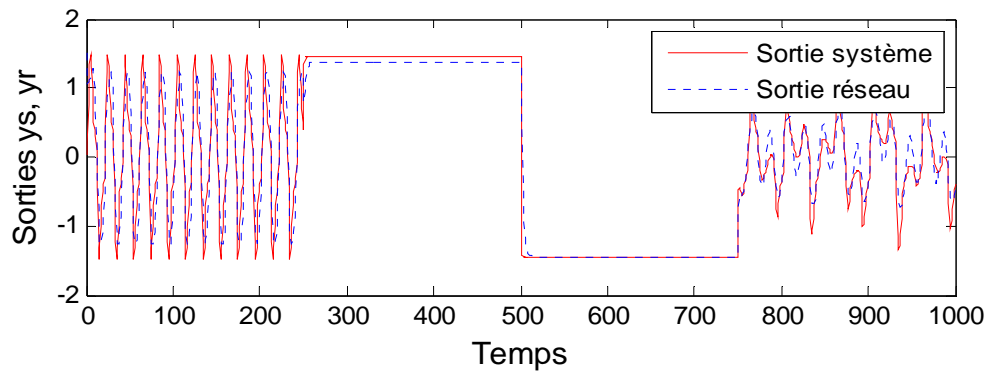


Figure III.22. Identification série-parallèle

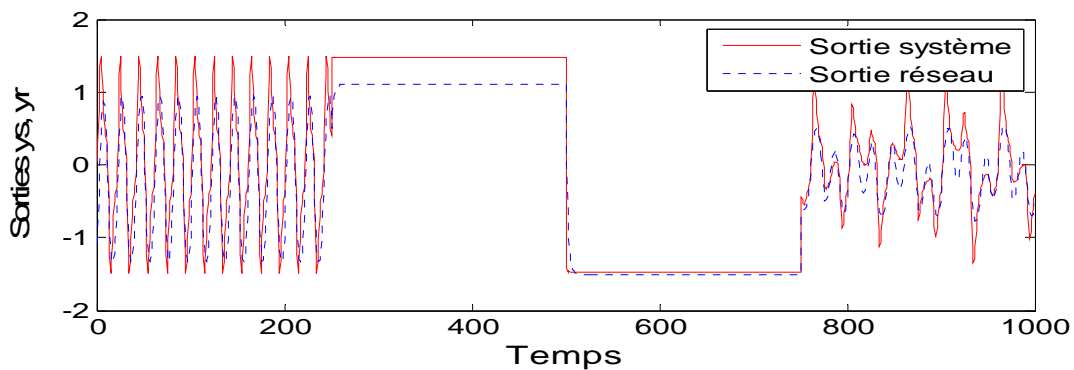


Figure III.23. Identification parallèle

Pour cet exemple, l'erreur quadratique moyenne est un peu faible (0.011 que ce soit pour l'identification série-parallèle ou l'identification parallèle) par rapport à l'exemple précédent.

III.5.3 Réseau DTRNN

a) Exemple 1

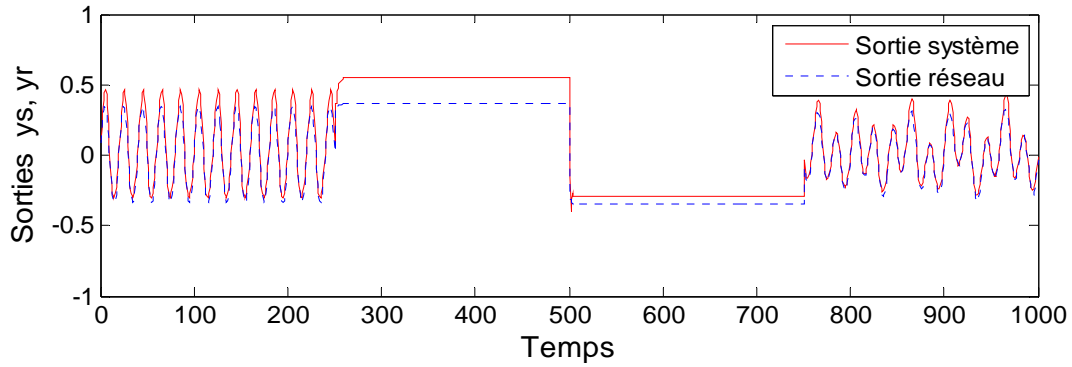


Figure III.24 Identification série-parallèle

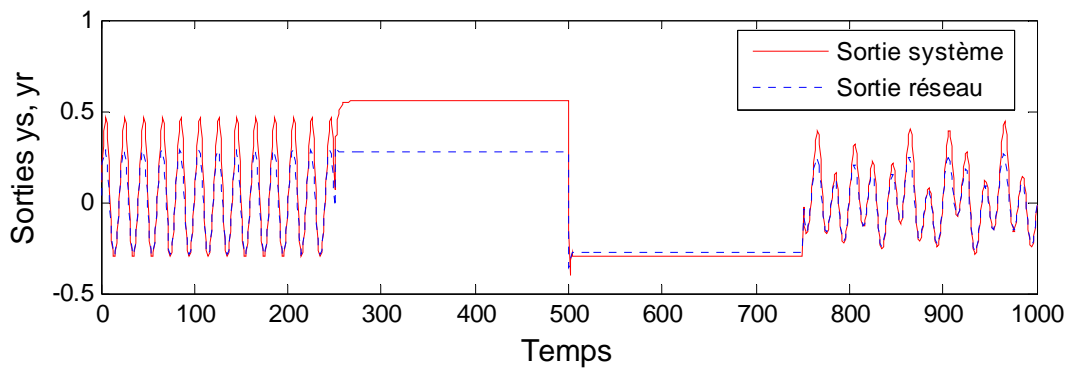
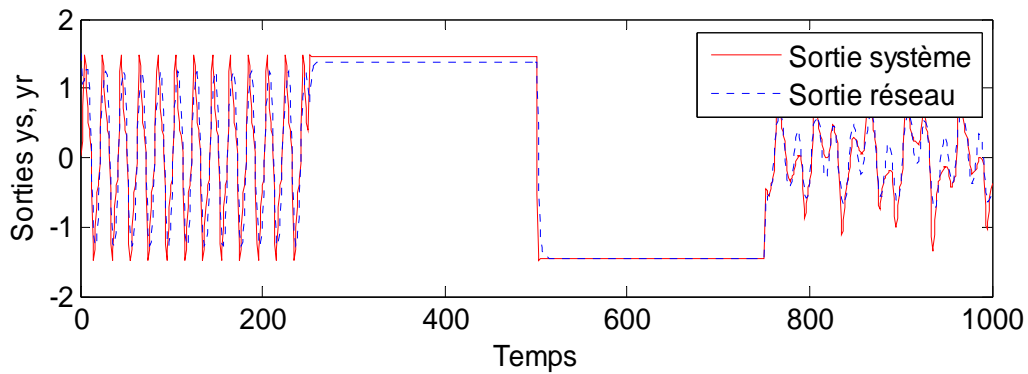
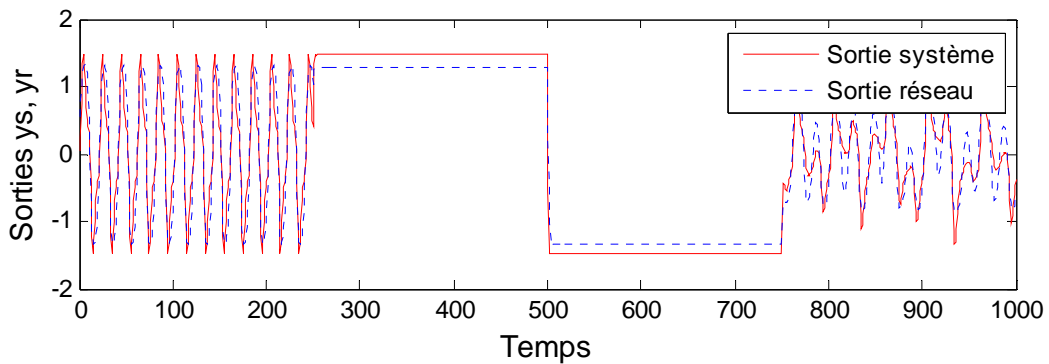


Figure III.25 Identification parallèle

Dans ce type de réseau, on remarque que la sortie du réseau suit parfaitement la sortie du système et l'erreur quadratique moyenne est égale à $3.35 \cdot 10^{-4}$ pour l'identification série parallèle, et est égale à $1.5 \cdot 10^{-3}$ pour l'identification parallèle. Les résultats sont obtenus pour un nombre de neurones égal à 14 pour l'unique couche de ce type de réseau et pour le nombre de générations égale à 200 pour les deux types d'identification.

b) Exemple 2**Figure III.26** Identification série-parallèle**Figure III.27** Identification parallèle

Pour l'exemple 2, on remarque que l'erreur quadratique est un peu plus grande par rapport à l'exemple 1, elle est égale à 0.0432 pour l'identification série- parallèle et est égale à 0.0585 pour l'identification parallèle, le nombre de générations est égal à 200 pour les deux types d'identification. On remarque aussi qu'il y a une divergence entre les deux sorties au point du changement de signe et cela dans la troisième partie de la séquence de test (c'est-à-dire entre 750 et 1000).

III.5.4 Commentaires

Les résultats obtenus par l'application des AG ont confirmé la puissance de ces algorithmes pour résoudre les problèmes d'optimisation d'ordre paramétrique. Les dernières générations fournissent une erreur quadratique plus petite que celle obtenue à partir des premières générations.

L'optimisation des réseaux de neurones par les AG a donné de résultats meilleurs. Ils se démarquent des algorithmes utilisant le gradient par l'initialisation des poids du réseau. En effet, les algorithmes d'apprentissages connexionnistes sont en général très dépendants de l'état initial du réseau (initialisation aléatoire des poids) et de la configuration de la base d'apprentissage. Un mauvais choix des poids employés pour initialiser le réseau, de la méthode de codage des données, ou même de l'ordre des données, peut bloquer l'apprentissage ou poser des problèmes pour la convergence du réseau vers une bonne solution.

Par contre pour les AG, il suffit juste d'augmenter la taille du choix des poids de départ (initialisation du réseau), ensuite par le bien des opérateurs génétiques (croisement, mutation, sélection) on assure la convergence du réseau.

III.6 Conclusion

Nous avons exposé dans ce chapitre les résultats des différents exemples avec leurs types d'identification.

Malgré l'efficacité des algorithmes utilisés pour l'adaptation des poids du réseau de neurones, il y a certains problèmes qui affectent le bon déroulement de l'apprentissage. L'utilisation des AG nous a permis d'avoir de meilleurs résultats. Ils feront l'objet d'une application dans le prochain chapitre sur une machine asynchrone.

CHAPITRE IV : APPLICATION A LA MACHINE ASYNCHRONE

IV Introduction

L'objet du présent chapitre est de tester la validité des programmes déjà développés et d'évaluer leurs performances à travers l'identification du courant de phase d'une machine asynchrone (MAS).

Ce chapitre est organisé en trois parties.

La première partie est consacrée à la modélisation de la machine asynchrone. Une mise en équation de la machine dans la base de Park est donnée. A partir des équations de phases, du couple et de l'équation mécanique, nous donnons le modèle d'état obtenu. Enfin une simulation en boucle ouverte du modèle termine cette première partie.

En deuxième partie, nous exposerons le principe de l'onduleur triphasé à modulation de large impulsion (M.L.I) afin de commander la MAS. Des résultats de simulation seront présentés.

La troisième partie concerne les résultats de simulation de l'association onduleur-MAS. Nous verrons comment les AG peuvent atteindre les objectifs désirés en optimisant les réseaux de neurones optimisés.

IV.1 Description de la machine asynchrone

IV.1.1 Constitution

La machine asynchrone comporte une partie fixe dite *STATOR*, et une partie mobile appelée *ROTOR* :

STATOR : il est constitué d'un enroulement bobiné réparti à l'intérieur d'une carcasse cylindrique faisant office de bâti et logé à l'intérieur d'un circuit magnétique supporté par cette carcasse. Ce circuit magnétique est formé d'un empilage de tôles, en forme de couronnes circulaires, dans lesquelles sont découpées les encoches parallèles à l'axe de la machine.

ROTOR : le rotor présente deux configurations dont le choix dépend de l'utilisation faite de la machine.

➤ *Rotor bobiné* :

Le rotor comporte un enroulement bobiné à l'intérieur d'un circuit magnétique constitué de disques empilés sur l'arbre de la machine. Cet enroulement est obligatoirement polyphasé même si le moteur est monophasé, et en pratique, toujours triphasé. Les encoches découpées dans les tôles sont théoriquement parallèles à l'axe du moteur.

Les extrémités de l'enroulement *rotorique* sont sorties et reliées à des bagues montées sur l'arbre, sur laquelle frottent des balais en carbone. On peut ainsi mettre en série avec le circuit *rotorique* des éléments complémentaires qui permettent des réglages par exemple de couple ou de vitesse.

➤ *Rotor à cage :*

Le circuit du rotor est constitué de barres conductrices régulièrement réparties entre deux couronnes métalliques formant les extrémités. Le tout rappelant la forme d'une cage d'écureuil bien étendu. Cette cage est insérée à l'intérieur d'un circuit magnétique analogue à celui du moteur à rotor bobiné.

Ce type de moteur, beaucoup plus aisé à construire que le moteur à rotor bobiné, et d'un prix de revient inférieur et à une robustesse intrinsèquement plus grande. Il n'est donc pas étonnant qu'il constitue, et de loin, la plus grande partie du parc des moteurs asynchrones en service.

IV.1.2 Principe de fonctionnement [30]

Le principe de fonctionnement repose entièrement sur les lois de l'induction :

« La machine asynchrone est un transformateur à champ magnétique tournant dont le secondaire (rotor) est en court-circuit ».

La vitesse de rotation N_s du champ tournant d'origine *statorique* qualifiée de synchronisme, est comme dans le cas des machines synchrones, rigidement liée à la fréquence f_s des tensions triphasées d'alimentation :

$$N_s = (\text{tr/min}) = 60 f_s / P \quad (4.1)$$

P : nombre de paires de pôles de chacun des enroulements des phases statoriques.

Lorsque le rotor tourne à une vitesse N différente de N_s (asynchronisme), l'application de la loi de *Faraday* aux enroulements rotoriques montre que ceux-ci deviennent le siège d'un système de forces électromotrices triphasées engendrant elles-mêmes trois courants rotoriques; d'après la loi de *Lenz* ces derniers s'opposent à la cause qui leur a donné naissance, c'est-à-dire la vitesse relative de l'induction tournante statorique par rapport au rotor.

Ainsi, les effets de l'induction statorique sur les courants induits rotoriques se manifestent par l'élaboration d'un couple de force électromagnétique sur le rotor tel que l'écart des vitesses, soit réduit.

De ce fait, selon que N est inférieur ou supérieur à N_s , la machine développe respectivement un couple moteur tendant à accroître N ou un couple résistant tendant à réduire N . De toute évidence, le couple électromagnétique s'annule à l'égalité des vitesses. L'échange énergétique avec le réseau dépend donc du signe de l'écart ($N_s - N$), c'est pourquoi on caractérise le fonctionnement asynchrone par le glissement g ainsi défini :

$$g = (N_s - N) / N_s \quad (4.1)$$

IV.2 Modélisation de la machine

Soit une machine asynchrone triphasée au stator et au rotor représentée schématiquement par la figure (VI.1) et dont les phases sont repérées respectivement par S_a , S_b , S_c et R_a , R_b , R_c , l'angle électrique θ variable en fonction du temps définit la position relative instantanée entre les axes magnétiques des phases S_a et R_a , choisis comme axes de référence.

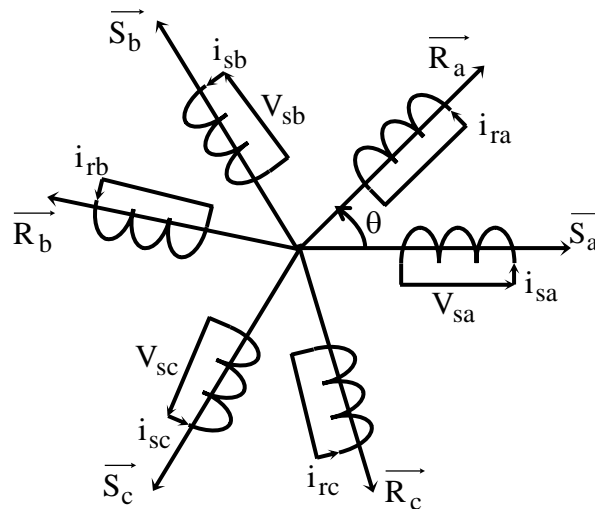


Figure VI.1 Représentation des enroulements de la MAS triphasée dans l'espace électrique

IV.2.1 Hypothèse simplificatrice [30]

L'étude de cette machine traduit les lois de l'électromagnétisme dans le contexte habituel d'hypothèses simplificatrices :

- Entrefer constant.
- Effet des encoches négligé.
- Distribution spatiale sinusoïdale des forces magnétomotrices d'entrefer.
- Circuit magnétique non saturé et à perméabilité constante.
- Pertes ferromagnétiques négligeables.
- L'influence de l'effet de peau et de l'échauffement sur les caractéristiques n'est pas prise en compte.

Parmi les conséquences importantes des hypothèses, on peut citer :

- L'additivité des flux.
- La constance des inductances propres.
- La loi de variation sinusoïdale des inductances mutuelles entre les enroulements *statoriques* et *rotoriques* en fonction de l'angle électrique de leurs axes magnétiques.

IV.2.2 Equations électriques

En se basant sur les hypothèses simplificatrices citées précédemment, les équations de tension peuvent s'écrire sous la forme matricielle suivante [30], [31] :

$$\begin{aligned} [V_s] &= [R_s][i_s] + \frac{d}{dt}[\phi_s] \\ [V_r] &= [R_r][i_r] + \frac{d}{dt}[\phi_r] \end{aligned} \quad (4.2)$$

$$[V_s] = \begin{bmatrix} V_{sa} \\ V_{sb} \\ V_{sc} \end{bmatrix} ; \text{ tension de stator, } [V_r] = \begin{bmatrix} V_{ra} \\ V_{rb} \\ V_{rc} \end{bmatrix} ; \text{ tension de rotor, } [i_s] = \begin{bmatrix} i_{sa} \\ i_{sb} \\ i_{sc} \end{bmatrix} ; \text{ courant statorique,}$$

$$[i_r] = \begin{bmatrix} i_{ra} \\ i_{rb} \\ i_{rc} \end{bmatrix} ; \text{ courant rotorique, } [\phi_s] = \begin{bmatrix} \phi_{sa} \\ \phi_{sb} \\ \phi_{sc} \end{bmatrix} ; \text{ flux statorique, } [\phi_r] = \begin{bmatrix} \phi_{ra} \\ \phi_{rb} \\ \phi_{rc} \end{bmatrix} ; \text{ flux rotorique,}$$

$$[R_r] = \begin{bmatrix} R_r & 0 & 0 \\ 0 & R_r & 0 \\ 0 & 0 & R_r \end{bmatrix} ; \text{ résistance rotorique, } [R_s] = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix} ; \text{ résistance statorique.}$$

IV.2.3 Equations magnétiques

Une matrice des inductances $[L(\theta)]$ établit la relation entre les flux et les courants. Elle comporte trente-six coefficients non nuls dont la moitié dépend du temps, par l'intermédiaire de θ (position du rotor).

$$\text{Soit: } \begin{bmatrix} \phi_{sa} \\ \phi_{sb} \\ \phi_{sc} \\ \phi_{ra} \\ \phi_{rb} \\ \phi_{rc} \end{bmatrix} = \begin{bmatrix} l_s & M_s & M_s & M_1 & M_3 & M_2 \\ M_s & l_s & M_s & M_2 & M_1 & M_3 \\ M_s & M_s & l_s & M_3 & M_2 & M_1 \\ M_1 & M_2 & M_3 & l_r & M_r & M_r \\ M_3 & M_1 & M_2 & M_r & l_r & M_r \\ M_2 & M_3 & M_1 & M_r & M_r & l_r \end{bmatrix} \begin{bmatrix} i_{sa} \\ i_{sb} \\ i_{sc} \\ i_{ra} \\ i_{rb} \\ i_{rc} \end{bmatrix} \quad (4.3)$$

La matrice des flux réels fait apparaître quatre sous matrices d'inductances :

$$\begin{aligned} [\phi_s] &= [L_{ss}][i_s] + [M_{sr}][i_r] \\ [\phi_r] &= [L_{rr}][i_r] + [M_{rs}][i_s] \end{aligned} \quad (4.4)$$

$$\text{Avec : } [L_{ss}] = \begin{bmatrix} l_s & M_s & M_s \\ M_s & l_s & M_s \\ M_s & M_s & l_s \end{bmatrix}; [L_{rr}] = \begin{bmatrix} l_r & M_r & M_r \\ M_r & l_r & M_r \\ M_r & M_r & l_r \end{bmatrix}$$

Les coefficients instantanés de mutuelle inductance entre le rotor et le stator s'expriment en fonction de M_{sr} (mutuelle entre le stator et le rotor) et θ

$$\text{On pose : } M_1 = M_{sr} \cos(\theta)$$

$$M_2 = M_{sr} \cos\left(\theta - \frac{2\pi}{3}\right)$$

$$M_3 = M_{sr} \cos\left(\theta + \frac{2\pi}{3}\right)$$

Alors, il vient :

$$[M_{sr}] = [M_{rs}]^T = M_{sr} \cdot \begin{bmatrix} \cos(\theta) & \cos\left(\theta + \frac{2\pi}{3}\right) & \cos\left(\theta - \frac{2\pi}{3}\right) \\ \cos\left(\theta - \frac{2\pi}{3}\right) & \cos(\theta) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ \cos\left(\theta + \frac{2\pi}{3}\right) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos(\theta) \end{bmatrix} \quad (4.5)$$

Finalement, l'équation (4.2) devient :

$$\begin{aligned} [V_s] &= [R_s][i_s] + \frac{d}{dt} \{ [L_{ss}][i_s] + [M_{sr}][i_r] \} \\ [V_r] &= [R_r][i_r] + \frac{d}{dt} \{ [L_{rr}][i_r] + [M_{sr}]^T [i_s] \} \end{aligned} \quad (4.6)$$

IV.2.4 Equation mécanique [31]

Le couple électromagnétique est donné par l'expression générale :

$$C_{em} = P [i_s]^T \frac{d}{d\theta} [M_{sr}][i_r] \quad (4.7)$$

L'équation mécanique s'écrit :

$$J \frac{d}{dt} \Omega = C_{em} - C_r - K_f \Omega \quad (4.8)$$

Avec :

- J : Moment d'inertie du rotor
- C_{em} : Couple électromagnétique
- C_r : Couple résistant
- Ω : Vitesse angulaire du rotor
- K_f : Coefficient de frottement

IV.3 Modèle biphasé de la machine

IV.3.1 Transformation de Park [32]

On désire transformer l'enroulement triphasé a, b, c , en trois enroulements équivalents d, q, o , dénommés :

- Direct selon x (indice d)
- Transversal selon y (indice q)
- Homopolaire selon z (indice o)

Les deux systèmes d'enroulement doivent être équivalents du point de vue des énergies instantanées électriques et magnétiques.

Il y a plusieurs transformations développées dans ce sens. Parmi ces transformations, on choisit, « *La transformation de Park* » dans laquelle on définit une matrice unique de transformation pour les courants, les tensions et les flux, à savoir :

$$[P(\theta)] = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos \theta & \cos(\theta - 2\pi/3) & \cos(\theta + 2\pi/3) \\ -\sin \theta & -\sin(\theta - 2\pi/3) & -\sin(\theta + 2\pi/3) \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \quad (4.9)$$

IV.3.2 Transformation de Park appliquée à la MAS [31]

Pour obtenir un système d'équations à coefficients constants, on transforme les enroulements statoriques et rotoriques en enroulements orthogonaux équivalents selon la méthode exposée au paragraphe précédent. Ainsi, les enroulements statoriques S_a, S_b, S_c , sont remplacés par trois enroulements équivalents S_d, S_q, S_o , et les enroulements rotoriques R_a, R_b, R_c , par R_d, R_q, R_c (voir figure (VI.2)).

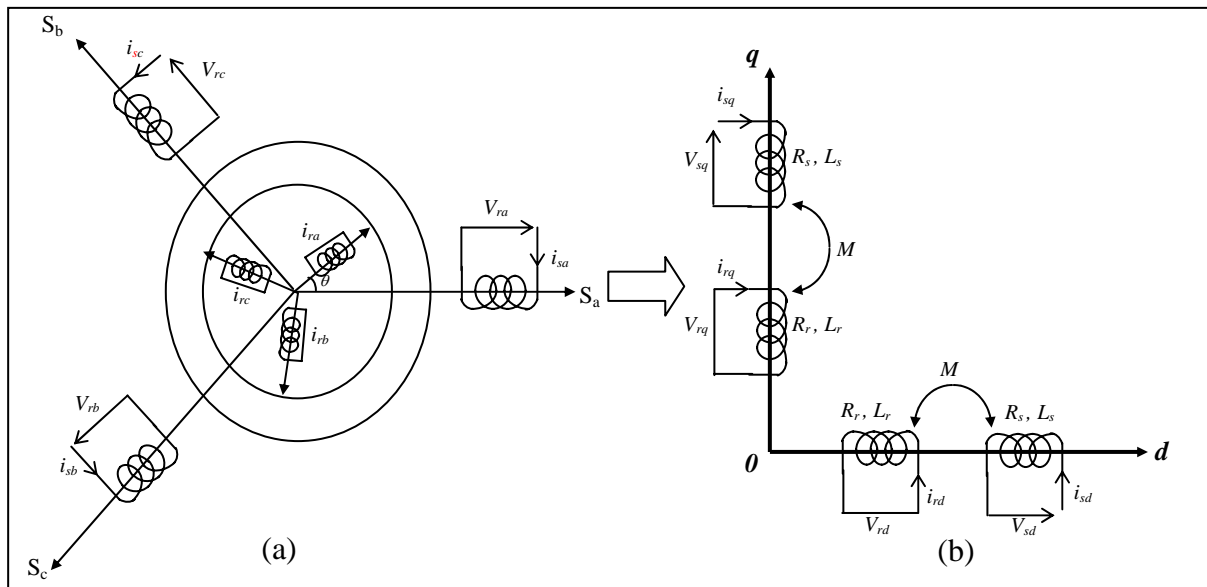


Figure VI.2 Transformée de Park appliquée aux enroulements de la MAS.

Deux transformations de Park sont définies à partir de la matrice dans laquelle l'angle θ est remplacé par θ_s pour le stator, et par θ_r pour le rotor ; on les note respectivement $[p(\theta_s)]$ et $[p(\theta_r)]$, ou:

θ_s : L'angle électrique (\vec{S}_a, \vec{d})

θ_r : L'angle électrique (\vec{R}_a, \vec{d})

On remarque sur la figure (VI.3) que θ_s et θ_r sont naturellement liés à θ par la relation :

$$\theta_s - \theta_r = \theta \quad (4.10)$$

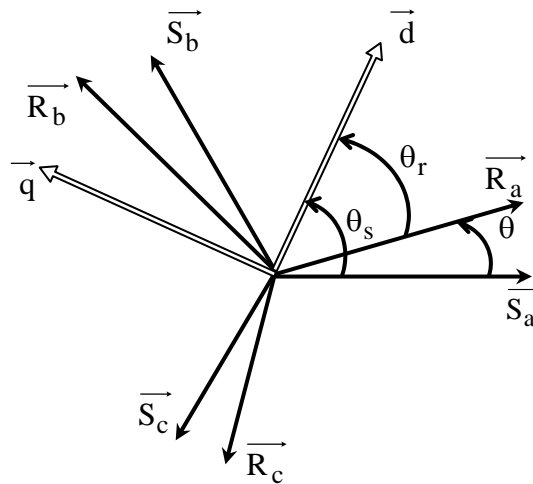


Figure VI.3 Repérage angulaire des systèmes d'axes dans l'espace électrique

Avec:

$$w_s = \frac{d\theta_s}{dt} : \text{Vitesse angulaire des axes (d, q) dans le repère statorique.}$$

$$w_r = \frac{d\theta_r}{dt} : \text{Vitesse angulaire des axes (d, q) dans le repère rotorique.}$$

De l'équation (4.10), on obtient :

$$w_s - w_r = \frac{d\theta}{dt} = w = P\Omega$$

La transformation de Park permet de réduire le nombre des paramètres électromagnétiques à :

$$L_s = l_s - M_s : \text{Inductance cyclique statorique}$$

$$L_r = l_r - M_r : \text{Inductance cyclique rotorique}$$

$$M = (3/2)M_{sr} : \text{Inductance mutuelle cyclique entre stator et rotor.}$$

Dans le repère de Park, les équations (4.2) et (4.4) s'écrivent comme suit :

$$\begin{cases} V_{sd} = R_s i_{sd} + \frac{d\phi_{sd}}{dt} - w_s \phi_{sq} \\ V_{sq} = R_s i_{sq} + \frac{d\phi_{sq}}{dt} + w_s \phi_{sd} \end{cases} \quad (4.11)$$

$$\begin{cases} 0 = R_r i_{rd} + \frac{d\phi_{rd}}{dt} - (w_s - w) \phi_{rq} \\ 0 = R_r i_{rq} + \frac{d\phi_{rq}}{dt} + (w_s - w) \phi_{rd} \end{cases} \quad (4.12)$$

$$\begin{cases} \phi_{sd} = L_s i_{sd} + M i_{rd} \\ \phi_{sq} = L_s i_{sq} + M i_{rq} \end{cases}, \begin{cases} \phi_{rd} = L_r i_{rd} + M i_{sd} \\ \phi_{rq} = L_r i_{rq} + M i_{sq} \end{cases} \quad (4.13)$$

IV.3.3 Expression du couple électromagnétique instantané

Une expression du couple électromagnétique exprimée à partir des différentes grandeurs prises dans le repère de Park peut être donnée par l'équation suivante :

$$C_{em} = P \frac{M}{L_r} (\phi_{rd} i_{sq} - \phi_{rq} i_{sd}) \quad (4.14)$$

L'équation mécanique du moteur s'écrit :

$$J \frac{d\Omega}{dt} = C_{em} - C_r - K_f \Omega \quad (4.15)$$

IV.3.4 Choix du référentiel

Il existe différentes possibilités concernant le choix de l'orientation du repère d'axes (d, q) qui dépendent des objectifs de l'application :

- Axes tournant à la vitesse du rotor $\theta_r = 0$: étude des grandeurs statoriques
- Axes liés au stator $\theta_s = 0$: étude des grandeurs rotoriques.
- Axes solidaires du champ tournant : étude de la commande.

C'est cette dernière solution qui fait correspondre à la solution la plus générale et la plus complexe. Nous utilisons un référentiel lié au stator noté (α, β) évitant une transformation dans un référentiel dont la position est mal connue.

Le modèle de la machine asynchrone triphasée s'écrit alors dans ce repère comme suit :

$$\begin{cases} V_{s\alpha} = r_s i_{s\alpha} + \frac{d}{dt} \phi_{s\alpha} \\ V_{s\beta} = r_s i_{s\beta} + \frac{d}{dt} \phi_{s\beta} \end{cases}, \begin{cases} 0 = r_r i_{r\alpha} + \frac{d}{dt} \phi_{r\alpha} + w \phi_{r\beta} \\ 0 = r_r i_{r\beta} + \frac{d}{dt} \phi_{r\beta} - w \phi_{r\alpha} \end{cases} \quad (4.16)$$

$$\begin{cases} C_{em} = P \frac{M}{L_r} (\phi_{r\alpha} i_{s\beta} - \phi_{r\beta} i_{s\alpha}) \\ J \frac{d\Omega}{dt} = C_{em} - C_r - k_f \Omega \end{cases} \quad (4.17)$$

Avec :

$$\begin{cases} \phi_{s\alpha} = L_s i_{s\alpha} + M i_{r\alpha} \\ \phi_{s\beta} = L_s i_{s\beta} + M i_{r\beta} \end{cases}, \begin{cases} \phi_{r\alpha} = L_r i_{r\alpha} + M i_{s\alpha} \\ \phi_{r\beta} = L_r i_{r\beta} + M i_{s\beta} \end{cases} \quad (4.18)$$

IV.4 Représentation d'état de la MAS [30], [31]

Le choix des variables d'état dépend des objectifs soit pour la commande soit pour l'observation. Pour le modèle complet, la vitesse mécanique Ω est une variable d'état, et pour les quatre variables électriques, les choix les plus courants sont :

$$\begin{aligned} X_1 &= [i_{s\alpha}, i_{s\beta}, \phi_{r\alpha}, \phi_{r\beta}, \Omega] \\ X_2 &= [i_{s\alpha}, i_{s\beta}, \phi_{s\alpha}, \phi_{s\beta}, \Omega] \\ X_3 &= [\phi_{r\alpha}, \phi_{r\beta}, \phi_{s\alpha}, \phi_{s\beta}, \Omega] \\ X_4 &= [i_{s\alpha}, i_{s\beta}, i_{r\alpha}, i_{r\beta}, \Omega] \end{aligned}$$

L'équation d'état de la machine asynchrone dans le repère (α, β) est donnée par :

$$\begin{cases} \dot{i}_{s\alpha} = -\gamma i_{s\alpha} + \frac{k}{T_r} \phi_{r\alpha} + k P \Omega \phi_{r\beta} + \frac{1}{\sigma L_s} V_{s\alpha} \\ \dot{i}_{s\beta} = -\gamma i_{s\beta} - k P \Omega \phi_{r\alpha} + \frac{k}{T_r} \phi_{r\beta} + \frac{1}{\sigma L_s} V_{s\beta} \\ \dot{\phi}_{r\alpha} = \frac{M}{T_r} i_{s\alpha} - \frac{1}{T_r} \phi_{r\alpha} - P \Omega \phi_{r\beta} \\ \dot{\phi}_{r\beta} = \frac{M}{T_r} i_{s\beta} - \frac{1}{T_r} \phi_{r\beta} + P \Omega \phi_{r\alpha} \\ \dot{\Omega} = \frac{PM}{L_r J} (\phi_{r\alpha} i_{s\beta} - \phi_{r\beta} i_{s\alpha}) - \frac{C_r}{J} - \frac{k_f}{J} \Omega \end{cases}$$

L'équation d'état de la MAS dans le repère (α, β) est donnée par :

$$\begin{cases} \dot{i}_{s\alpha} = -\gamma i_{s\alpha} + \frac{k}{T_r} \phi_{r\alpha} + k P \Omega \phi_{r\beta} + \frac{1}{\sigma L_s} V_{s\alpha} \\ \dot{i}_{s\beta} = -\gamma i_{s\beta} - k P \Omega \phi_{r\alpha} + \frac{k}{T_r} \phi_{r\beta} + \frac{1}{\sigma L_s} V_{s\beta} \\ \dot{\phi}_{r\alpha} = \frac{M}{T_r} i_{s\alpha} - \frac{1}{T_r} \phi_{r\alpha} - P \Omega \phi_{r\beta} \\ \dot{\phi}_{r\beta} = \frac{M}{T_r} i_{s\beta} - \frac{1}{T_r} \phi_{r\beta} + P \Omega \phi_{r\alpha} \\ \dot{\Omega} = \frac{PM}{L_r J} (\phi_{r\alpha} i_{s\beta} - \phi_{r\beta} i_{s\alpha}) - \frac{C_r}{J} - \frac{k_f}{J} \Omega \end{cases} \quad (4.19)$$

Avec :

$$\sigma = 1 - \frac{M^2}{L_s L_r} \quad \text{Coefficient de dispersion}$$

$$T_r = \frac{L_r}{R_r} \quad \text{Constante de temps rotorique}$$

$$T_s = \frac{L_s}{R_s} \quad \text{Constante de temps statorique}$$

$$\gamma = \frac{1}{\sigma T_s} + \frac{M^2}{\sigma L_s L_r T_r}$$

$$k = \frac{M}{\sigma L_s L_r}$$

Parmi les procédés permettant de faire varier la vitesse du moteur à courant alternatif, le plus utilisé consiste à faire varier la fréquence de ses tensions d'alimentation, donc l'alimenter par un onduleur ou un cycloconvertisseur.

La section qui suit sera consacrée à la modélisation de l'onduleur de tension à M.L.I (Modulation Largeur d'Impulsion).

IV.5 Onduleur de tension à MLI

La génération du réglage de la vitesse du moteur à courant alternatif s'effectue de plus en plus par la variation de fréquence, ce qui nécessite alors l'emploi d'un onduleur triphasé [32].

IV.5.1 Principe de la MLI sinus-triangle

➤ Stratégie de commande :

La modulation de largeur d'impulsion consiste à imposer aux bornes de la machine des créneaux de tension de manière à ce que le fondamental de la tension soit le plus proche de la sinusoïde idéale avec des valeurs d'harmoniques les plus faibles possibles. Les instants des impulsions de commande des interrupteurs sont déterminés par les intersections du signal de référence appelé « modulatrice » avec un signal triangulaire appelé « porteuse » de fréquence très élevée par rapport à la fréquence de la modulatrice.

Dans le cas de la M.L.I triphasé, le principe de la modulation est le suivant :

- La porteuse est commune pour les trois phases.
- La modulatrice est propre à chaque phase.

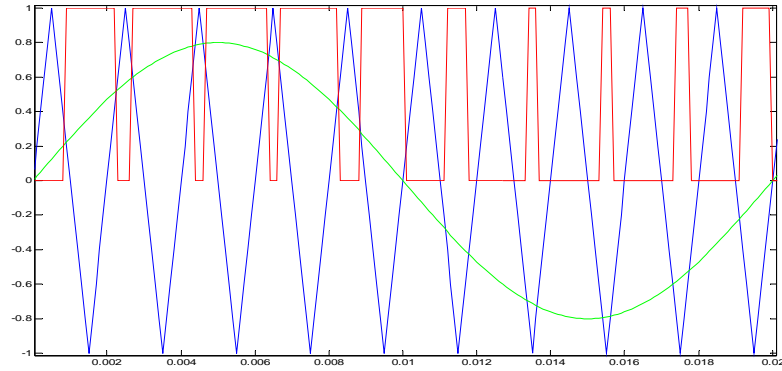


Figure VI.4 Le principe de la MLI

Sur la figure, la référence est schématisée en vert, la porteuse en bleu et le résultat de la comparaison au rouge.

IV.5.2 Modélisation de l'onduleur de tension à M.L.I

L'onduleur de tension est un convertisseur statique constitué de cellules de commutation à semi-conducteur associée aux diodes en antiparallèle (figure (VI.4)).

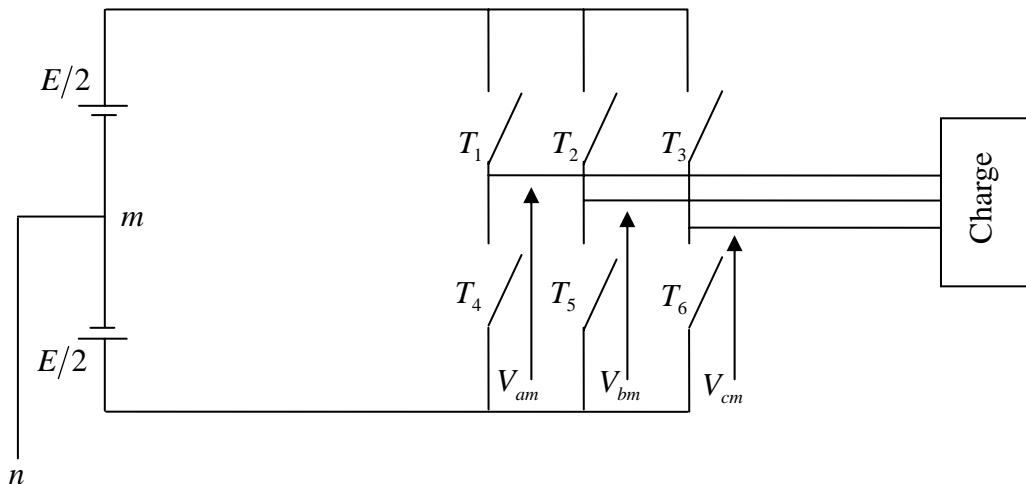


Figure VI.5 Schéma d'un onduleur de tension triphasé

Dans cette modélisation et pour ne pas alourdir ce travail, le phénomène des temps morts de l'onduleur de tension n'est pas pris en compte. Les temps morts représentent les temps de sécurité entre les impulsions de commande des semi-conducteurs de chaque bras de l'onduleur, et ce pour éviter le court-circuit de la source de tension continue [31].

Hypothèses

- Les semi-conducteurs sont des interrupteur parfaits ; une résistance nulle à l'état passant, une résistance infinie à l'état bloquée et une réaction instantanée aux signaux de commande.

On modélise le convertisseur en utilisant une matrice de connexion, l'analyse par la notation de modèle étant compliquée. Cette matrice représente les états logiques de ces semi-conducteurs.

$$[Tr] = \begin{bmatrix} Tr_1 & Tr_4 \\ Tr_2 & Tr_5 \\ Tr_3 & Tr_6 \end{bmatrix} \quad (4.8)$$

Avec :

$$Tr_i = \begin{cases} 1 & \text{Si le semi - conducteur } Tr_i \text{ est passant} \\ 0 & \text{Si le semi - conducteur } Tr_i \text{ est bloqué} \end{cases}$$

En plus de cette matrice, on définit une autre matrice de connexion $[D]$ qui caractérise les différents états des diodes.

$$[D] = \begin{bmatrix} D_1 & D_4 \\ D_2 & D_5 \\ D_3 & D_6 \end{bmatrix} \quad (4.9)$$

Avec :

$$D_i = \begin{cases} 1 & \text{Si la diode } D_i \text{ est conductrice} \\ 0 & \text{Si la diode } D_i \text{ est bloquée} \end{cases}$$

On posera :

$$T_i = Tr_i + D_i \quad i = 1,6 \quad (4.10)$$

Donc, la matrice de connexion $[T]$ de l'onduleur s'écrit :

$$[T] = \begin{bmatrix} T_1 & T_4 \\ T_2 & T_5 \\ T_3 & T_6 \end{bmatrix} \quad (4.11)$$

Les tension imposées de chaque bras de l'onduleur sont définies par :

$$\begin{bmatrix} V_{1m} \\ V_{2m} \\ V_{3m} \end{bmatrix} = \begin{bmatrix} T_1 & T_4 \\ T_2 & T_5 \\ T_3 & T_6 \end{bmatrix} \cdot \begin{bmatrix} \frac{E}{2} \\ 0 \end{bmatrix} \quad (4.12)$$

On a aussi :

$$\begin{cases} V_{1m} = V_{an} + V_{nm} \\ V_{2m} = V_{bn} + V_{nm} \\ V_{3m} = V_{cn} + V_{nm} \end{cases} \quad (4.13)$$

Avec : V_{an}, V_{bn}, V_{cn} les tensions aux bornes de la charge et le neutre.

Pour un système de tensions triphasées équilibrées on peut déduire le système d'équations des tensions simples suivant :

$$\begin{bmatrix} V_{an} \\ V_{bn} \\ V_{cn} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \end{bmatrix} \bullet \begin{bmatrix} V_{1m} \\ V_{2m} \\ V_{3m} \end{bmatrix} \quad (4.14)$$

IV.5.3 Résultat de simulation

Des essais de simulation numérique ont été faits dans le cas d'une MAS alimentée par un onduleur de tension à M.L.I.

➤ **Caractéristiques de la MAS utilisées :**

Puissance nominale	p_n	1.5 Kw
Vitesse nominale	Ω_n	1424 rad/s
Inductance mutuelle	M	0.058 H
Inductance statorique	L_s	0.16 H
Inductance rotorique	L_r	0.023 H
Résistance statorique	R_s	0.85 Ω
Résistance rotorique	R_r	0.16 Ω
Moment d'inertie	J	0.05 Kg / m ²
Nombre de paire de pôles		2

Tableau VI.1 Caractéristiques de la machine

a) à vide

b) avec couple résistant = 10N.m

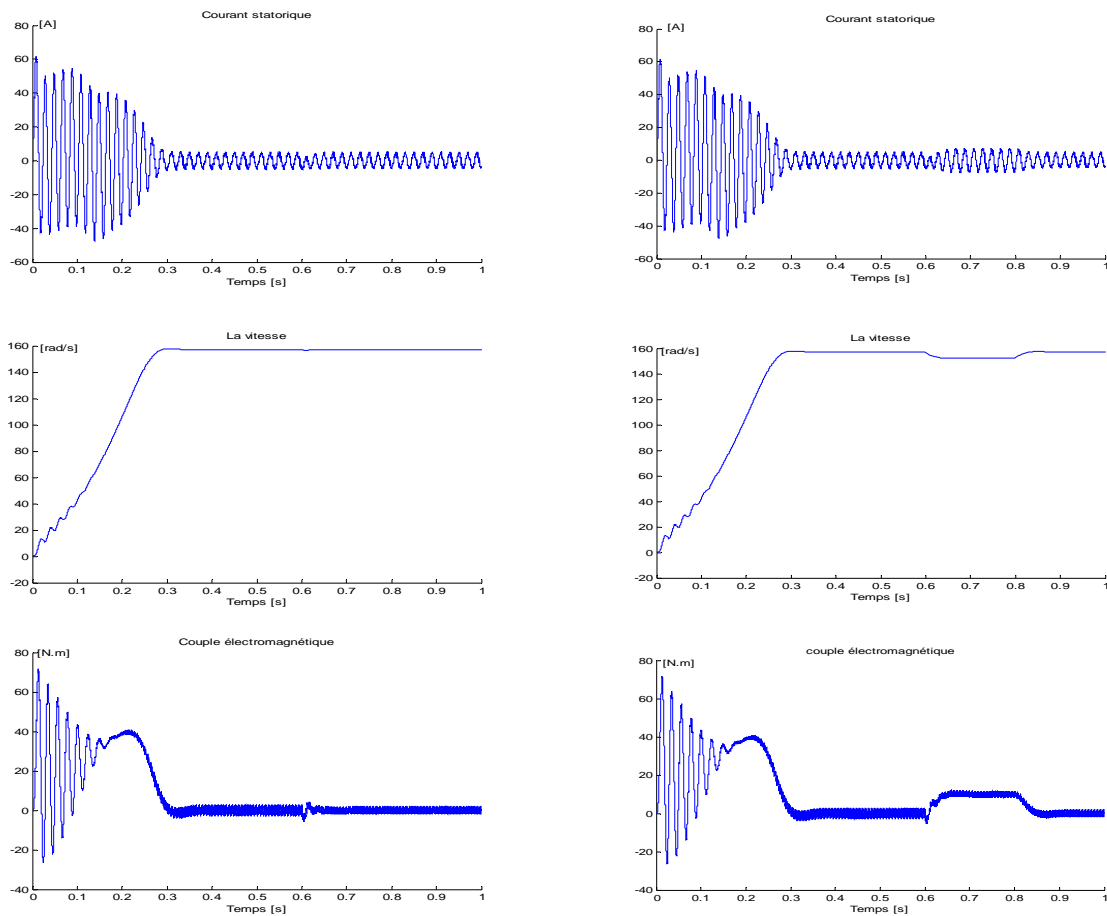


Figure VI.6 Résultat de simulation onduleur-MAS

➤ Commentaires des résultats

Lors du démarrage à vide, on constate d'emblée l'importance des courants statoriques avec un couple fortement pulsatoire [voir figure (VI.6 a)]. Il atteint une valeur maximale instantanée de 60 N.m.

Les oscillations du couple se font évidemment ressentir sur l'évolution de la vitesse qui en régime permanent se stabilise à 157 rad/s avec glissement pratiquement nul [voir figure (VI.6 a)].

Le temps de démarrage est de 0,22 secondes. En absence du couple résistant, les courants se stabilisent à des valeurs faibles correspondant au comportement inductif du moteur à vide.

Lorsque l'arbre de la machine est sollicité par un échelon de couple résistant, le couple électromagnétique compense instantanément cette sollicitation par une repense optimale. Néanmoins, nous remarquons une chute de la vitesse traduite par un glissement supplémentaire de la machine [voir figure (VI.6 b)].

IV.6 Identification de l'onduleur-MAS

Nous allons maintenant nous intéresser au problème de l'identification de l'association onduleur-MAS. Notre but ici n'est plus de présenter des algorithmes d'optimisation des réseaux de neurones, mais d'appliquer les algorithmes développés auparavant pour l'identification d'un système réel.

La figure suivante montre le schéma adopté pour l'application :

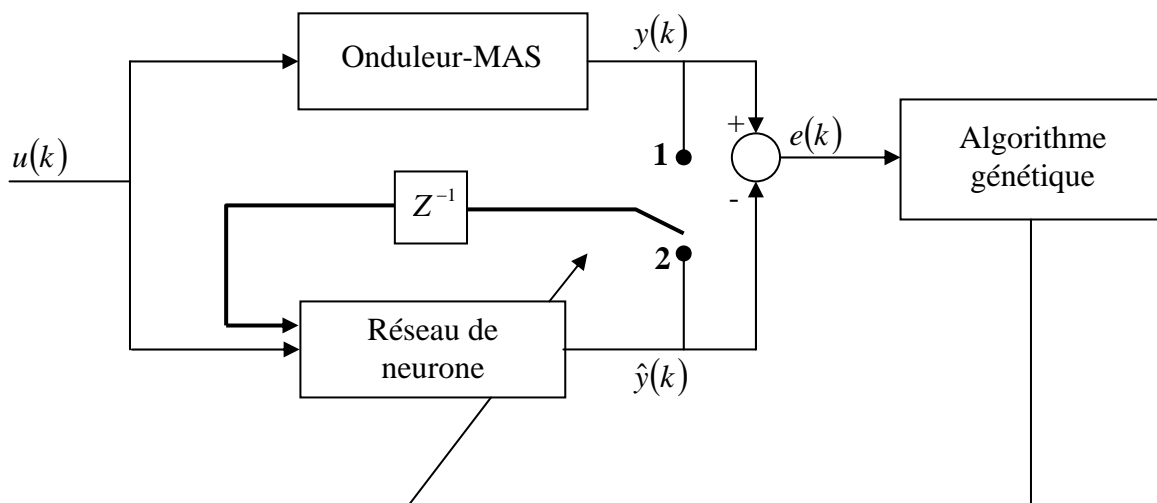


Figure VI.7 Structure d'identification
Position du commutateur :
(1) série-parallèle, (2) parallèle.

IV.7 Résultats de simulation

On s'intéresse dans cette partie seulement à l'identification du régime permanent du courant statorique du système onduleur-MAS. L'apprentissage des réseaux se fait sans charge c'est-à-dire le couple résistant est nul. L'adaptation des poids des réseaux se fait à l'aide des AG. Puis on teste les réseaux obtenus par rapport au système avec charge (couple résistant égal à 10 N.m entre $0,6 < t < 0,8$) pour voir leurs efficacités.

IV.7.1 Identification série-parallèle

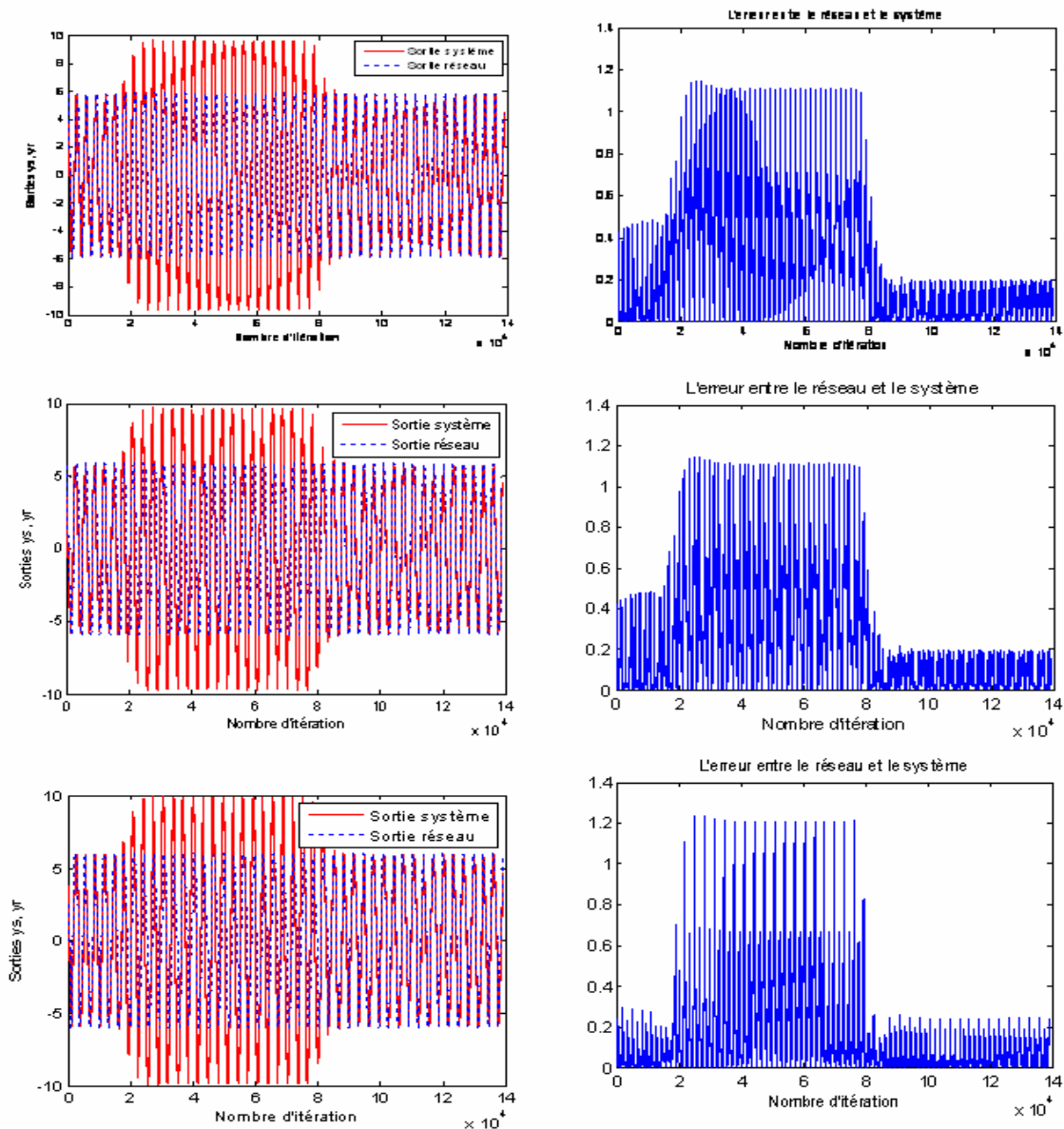


Figure VI.8 Identification série-parallèle
 a) MLP, b) MNN, c) DTRNN

➤ **Commentaires:**

Après une série de tests nous avons opté pour les résultats illustrés dans la figure (VI.8). Elle montre bien que les réseaux suivent parfaitement la sortie du système pour les trois types de réseaux. Lorsqu'on applique une charge sur la machine entre $t=0,6$ s et $t=0,8$ s, on remarque que la machine fait appel à un courant plus fort. Par contre, la sortie du réseau reste inchangée. Cela veut dire que les réseaux de neurone ont bien appris le comportement du système.

IV.7.2 Identification parallèle

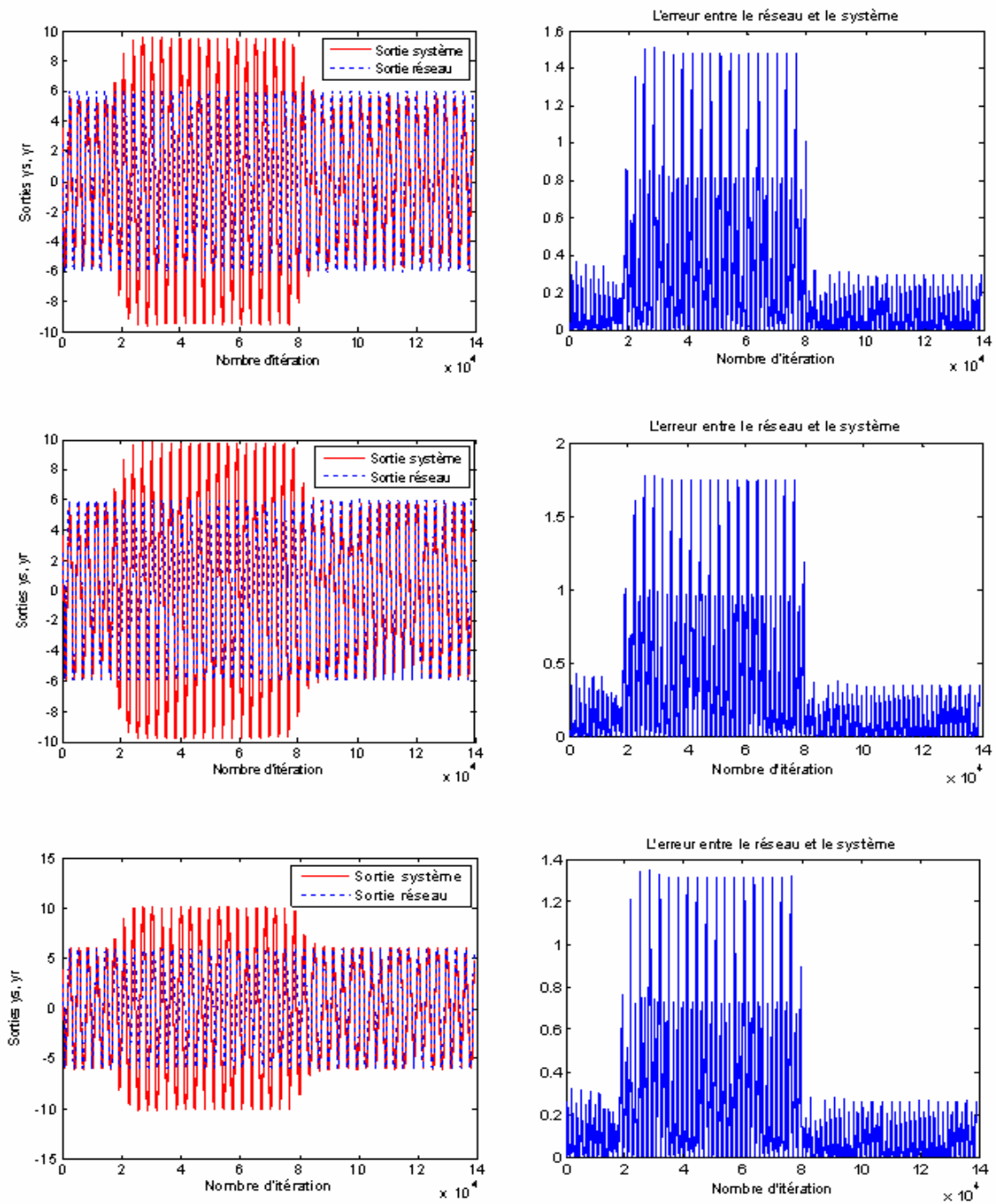


Figure VI.9 Identification parallèle
 a) MLP, b) MNN, c) DTRNN

➤ Commentaires :

De même, comme l'identification série-parallèle, l'identification parallèle a donné de bons résultats, pour tous les trois types de réseaux. On arrive bien à identifier la charge

appliqué à la machine. Et cela juste pour 200 générations des AG. Par contre, pour l'identification série-parallèle, les résultats obtenus sont pour 500 générations des AG.

IV.8 Conclusion

L'application des AG pour l'optimisation des topologies des réseaux de neurones fait état de résultats satisfaisants (suivi parfait et identification correcte des systèmes non linéaires).

Il suffit uniquement de bien définir les objectifs par une fonction d'évaluation adéquate pour que les AG interviennent avec efficacité pour sélectionner les meilleurs poids d'adaptation pour les réseaux de neurones.

CONCLUSION GENERALE

Conclusion générale

Le travail présenté dans ce mémoire a porté sur l'optimisation des réseaux de neurones par les AG en vue d'identification des systèmes non linéaires.

Nous avons tiré parti de deux caractéristiques fondamentales des réseaux de neurones :

- La propriété d'approximation universelle parcimonieuse, dont l'apport est considérable pour l'identification de processus non linéaires.
- L'existence d'algorithmes d'apprentissage également universels, au sens où leurs mises en œuvre ne dépendent pas de l'application considérée ni de la complexité du réseau soumis à un apprentissage.

Les AG sont des outils d'optimisation performants qui permettent de réaliser une exploration globale de l'espace de recherche. Contrairement aux méthodes déterministes classiques, ils ne nécessitent aucun calcul de dérivées.

L'entraînement en ligne du modèle de réseau de neurone avec les structures MLP, MNN, DTRNN, consiste à l'ajustement des poids du réseau qui est effectué dans notre travail à l'aide de AG. Cet algorithme a été utilisé pour l'entraînement des réseaux MLP, MNN, DTRNN, présent dans toutes les structures d'identification que nous avons procédées dans notre mémoire.

A partir des résultats de simulation, nous avons démontré la capacité des réseaux de neurones optimisés par les AG pour l'identification des processus non linéaires. Les résultats sont satisfaisants vu les exigences de chaque système utilisé. En particulier, nous avons montré par une application sur l'onduleur-MAS, l'intérêt des AG et leurs capacités à optimiser les structures des réseaux de neurones.

Comme perspectives de recherche nous suggérons de continuer dans la même voie en investissant dans l'utilisation d'autres méthodes d'optimisation globales telle que le « recuit simulé » pour une éventuelle étude comparative.

BIBLIOGRAPHIE

Bibliographie

- [1] : T. Takagi, M. Sugeno. « *Derivation of Fuzzy Control Rules From Human Operator's Control Actions* », Proc/ of the IFAC Symposium on Fuzzy Information, pp. 55-60, 1983.
- [2] : P. S. Sastry, G. Santharam, K. P. Unnikrishnan. « *Memory Neuron Networks for Identification and Control of Dynamical Systems* », IEEE transactions on neural networks, Vol.5 N°2, pp. 306-319, March 1994.
- [3] : S. Ladjouzi. « *Commande des Systèmes Dynamiques par les Réseaux Neuronaux (RNNs) et la Propagation Directe de l'erreur* », Mémoire de Magister, Université de Bejaia, Février 2005.
- [4] : W. Guenounou. « *Optimisation des contrôleurs flous par Algorithmes génétiqueHiérarchisés. Application sur un actionneur Asynchrone* », Mémoire de Magister, Université de Bejaia, Juin 2003.
- [5] : N. Benahmed. « *Optimisation de réseaux de neurones pour la reconnaissance de chiffres manuscrits isolés : sélection et pondération des primitives par algorithmes génétiques* », Mémoire de Maîtrise, Ecole de Technologie Supérieure Montréal, Mars 2002.
- [6] : D. E. Goldberg. « *AG, Exploration, optimisation et Apprentissage Automatique* », Addisonwesly, 1994
- [7] : A. Spalanzani. « *Algorithmes évolutionnaires pour l'étude de la robustesse des systèmes de reconnaissance automatique de la parole* » Thèse de Doctorat, Université de Grenoble 1999.
- [8] : U. Seiffert, « *Multipele Layer Perceptron training using genetic algorithms* », In. Proc. Of the 9th European Symposium on artificial Neural Networks, SANN'01, pp.159-164, 2001.
- [9] : R. Tlemsani, N. Neggaz, A. Benyettou. « *Amélioration de l'apprentissage des réseaux de neurones par les algorithmes évolutionnaires* » 3rd International Conference : Sciences of Electronic, Technologies of information and Telecommunications, SETIT' 2005.
- [10] : D. Benhaddouche « *Data mining par hybridation algorithmes génétiques-réseaux de neurones* » Proceedings of the international conference Méthodes et Outils d'aide à la Décision. Béjaia, pp. 245-250, 2007
- [11] : C. Touzet. « *Les réseaux de neurone artificiels* », Cours, Université de Provence (Aix-Marseille I), juillet 1992.
- [12] : H.Abdi. « *Les Réseaux de neurones* », édition presses universitaires de Grenoble, 1994.
- [13] : P.K. Simpson. « *Artificial Neural Systems* », Pergmon Press Elmsford, New York, 1989.
- [14] : G. A. Harrison, J. M. Tanner. « *Human biology: An Introduction to human evolution* » Oxford science publication, 1998.

- [15] : M. Parizeau. « *Réseaux de neurones* », GIF-21140 et GIF-64326, Université Laval, Automne 2004.
- [16] : G. Dreyfur. « *Réseaux de neurones : Méthodologie et application* », édition Eyrolles, 2004.
- [17] : L.Behera , S.Kumar, S.Chandradas. « *Identification of nonlinear dynamical systems using recurrent neural network* », India Institute of Technology 0-7803-7651-X /03, IEEE. 2003.
- [18] : D. Hush, B.G. Horne. « *Progress in supervised Neural Networks* », IEEE signal processing Magazine, pp. 8-39, Janvier 1993.
- [19] : A. Blum. « *Neural Networks in C ++* », Wiley & Sons Edit, New York, 1992.
- [20] : B. Kosko. « *Unsupervised Learning in Noise* », IEEE Trans.Neural Net, Vol.1, N°1, pp. 44-57, Mars 1990.
- [21] : A.A. Zhigljavsky. « *Theory of Global Random Search Mathematics and its Applications* », Kluwer Academic Publishers. 1991.
- [22] : A. Traynard. « *D'une méthode de gradient conjugué préconditionné élément par élément* » Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France. 1990.
- [23] : N. Zerari. « *Algorithmes génétiques en maintenance* » Mémoire de Magister, Université El Hadj Lakhadar, Batna. 2006.
- [24] : M. Wall, Galib. « *A C++ library of genetic algorithms components* » Massachusetts Institute of Technology MIT press, 1996, disponible : <http://lancet.mit.edu/ga/>
- [25] : P. Rebreyend. « *AG, Hybrides en optimisation combinatoire* », Thèse doctorat en informatique, l'Ecole Normale Supérieure de Lyon, 1999.
- [26] : M. Demouche, A. Oukaour, D. Aissani, B. Boudart. « *Non Linear Classification with Neural Networks* » Proceedings of the international confrence Méthodes et Outils d'aide à la Décision. Béjaia, pp. 649-653, 2007.
- [27] : I Guyon, I. Poujaud, L. Personnaz, G. Dreyfus, J. Denker, Y. Gun. « *Comparing different neural networks architectures for classifying handwritten digit* », Int. J. Conf. On Neural networks, Vol 2, Washington, USA, pp. 127-132. 1989.
- [28] : A. Forsgren, R. Kling. « *An Implementation of Recurrent Neural Networks for Prediction and Control of Nonlinear Dynamic Systems* », Thèse Doctorat, Université de Lulea, Australie, Mars 2003.
- [29] : A. Aussem. « *Théorie et applications des réseaux de Neurones Récurrents et Dynamiques à la Modélisation et au contrôle Adaptatif des Processus Dynamiques* », Thèse Doctorat, Université Paris V, Juin 1995.

- [30] : P. Barret. « *Régime transitoire des machines tournantes électriques* », édition Eyrolles, 1987.
- [31] : J. P. Caron, J.P. Hautier. « *Modélisation et commande de la machine asynchrone* » édition TECHNIP France, 2002.
- [32] : J. Chatelain, « *Machine électriques, traite d'électricité* », volume X, presse polytechnique romandes, 1989.
- [33] : Holland J.H « *Adaptation in natural and artificial systems* » MIT press, 1975.
- [34] : Man K.F, Tang K.S, and Kwong S « *Genetic algorithms concepts and designs* » édition Springer, 2000.

Résumé

Les réseaux de neurones artificiels constituent l'un des outils les plus performants de l'Intelligence Artificielle. Ces réseaux possèdent des caractéristiques intéressantes telles que la modélisation de fonctions non linéaires, le traitement parallèle, l'apprentissage,.... Ces caractéristiques font qu'ils sont des candidats idéaux pour l'identification des systèmes non linéaires. Cependant, l'utilisation des réseaux de neurones entraîne des inconvénients tels que : difficultés lors de l'apprentissage de systèmes hautement complexes et la lenteur de l'algorithme de rétro- propagation.

Face à ces inconvénients, on s'est tourné vers l'application des algorithmes génétiques pour entraîner les réseaux de neurones et optimiser leurs structures.

Dans ce travail, on a surmonté la difficulté d'entraînement en utilisant les algorithmes génétiques, appliqués à trois types de réseaux de neurones, à savoir : réseaux multi-couches (MLP), réseaux à mémoire et réseaux récurrents à temps discret (DTRNN).

Les résultats théoriques sont validés sur un exemple académique puis sur le modèle onduleur-MLI-machine asynchrone.

Mots clés : Réseaux de neurones, Algorithmes génétiques, Identification, Systèmes non linéaires.

Abstract

The artificial neural network is one of the most powerful Artificial Intelligence. These networks have features such as modeling nonlinear functions, parallel processing, learning These characteristics are that they are ideal candidates for the identification of nonlinear systems. However, the use of neural networks entails disadvantages such as difficulties in learning systems highly complex and slow algorithm of back propagation.

Faced with these disadvantages, we turned to the application of genetic algorithms to train neural networks and optimize their structure.

In this work, we overcame the difficulty of training using genetic algorithms applied to three types of neural networks, namely multi-layers perceptron (MLP), memory neural networks (MNN) and discrete time recurrent neural networks (DTRNN)

The theoretical results are validated on a sample academic model and then the asynchronous machine.

Keywords: neural networks, genetic algorithms, Identification, nonlinear systems.