République Algérienne Démocratique et Populaire

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

Faculté de Génie Electrique et d'Informatique
Département d'Automatique

MEMOIRE DE FIN D'ETUDES

En vue de l'obtention du diplôme

de MASTER ACADEMIQUE EN AUTOMATIQUE
OPTION : COMMANDE DES SYSTEMES

Thème

Commande par Modèle Inverse neuronal, application à la commande en vitesse d'un moteur à courant continu

Proposé par :
M. HAMACHE Amar

Dirigé par:
M. HAMACHE Amar

Présenté par :
- M. ANANE Hamid
- M^elle^ AHMED ZAID Tassadit

Soutenu le : 01/07 /2013

Promotion 2013

Remerciements

Nous tenons à remercier tout d'abord DIEU le tout puissant qui nous a donné, durant toutes ces années, la santé, le courage et la foi pour arriver a ce jour.

Nous ne pouvons réellement trouver, les expressions éloquentes que mérite notre encadreur : Mr HAMACHE Amar, afin de le remercier pour ça sympathie, ses encouragement, son aide, son dévouement pour le travail et sa présence au cours de cette étude

Nous adressons nos remerciements aux membres de jury qui nous ont fait l'honneur d'évaluer, d'examiner, d'enrichir ce travail.

Nos remerciements vont également à tous les enseignants et les responsables de notre faculté de génie électrique et d'informatique.

Dédicaces

Aux être qui me sont les plus chers

« MES PARENTS »

Pour leurs AMOUR leur EDUCATION et leurs SACRIFICES.

A mes frères et ma sœur, pour qui je souhaite la réussite dans leurs projets en avenir.

A mes grandes mères.

A mes tantes, mes oncles, mes cousins, mes cousines.

A mes cher amis et frères Slimane, Merizek, Malik, Mourad, Farid, Ali, Amazigh, Nassim.

A mes amies et sœurs Baby, Sylia, Tassadit.

A ma future femme.

A toute ma famille.

A tout mes amis.

A tous ceux qui m'ont aidé à faire ce travail.

ANANE Hamid

Dédicaces

Je dédie ce travail à :

Mon très cher oncle Farid,

Ma très chère mère,

Ma très chère grand- mère,

A mes tendres frère et sœurs,

A tous mes oncles, tantes, cousins et cousines,

A mon binôme Hamid,

A tout mes ami(e)s et camarades,

A ceux que j'aime et qui m'aiment,

AHMED ZAID Tassadit

Résumé

Dans le travail mené en cette thèse, il a été constaté que la commande inverse neuronale appliquée au moteur électrique à courant continu était une commande qui pouvait garantir de bonnes performances, que ce soit au niveau de la précision, du dépassement, ou de la stabilité ou aussi la robustesse vis-à-vis les perturbations extérieures et intérieurs.

Dans un premier temps : présentation du neurone biologique et le neurone formel, les différents types de réseaux de neurones formels, différents types d'apprentissage, et le signal d'excitation SBPA. En suite présentation du Modèle Inverse Neuronal ses architectures d'apprentissages ainsi que celles de la commandes et une étude détaillée de l'algorithme de rétro-propagation du gradient de l'erreur. Le travail se termine par la simulation de l'apprentissage du réseau de neurones, et l'utilisation de ce dernier pour une Commande en vitesse avec le Modèle Inverse Neuronal du moteur à courant continu.

Mots clés

Réseaux de neurones, commande inverse neuronal, commande, commande neuronal, commande en vitesse d'un moteur à courant continu,

III. CHAPITRE 3 : COMMANDE DU MCC AVEC LE RESEAU DE NEURONES

LISTE DES FIGURES ET DES TABLEAUX

Introduction générale

Introduction Générale

La technologie moderne a permis le développement des sciences tout en imposant l'exploration de domaines théoriques de plus en plus complexes. Parmi ces sciences en pleine expansion et intégrant rapidement l'apport des technologies modernes, on compte l'automatique. Le substantif « automatique » a été utilisé pour la première fois en 1914 dans un article « Essai sur l'Automatique » publié dans une revue scientifique.

De nos jours, l'automatique fait partie des sciences de l'ingénieur. Cette discipline traite de la modélisation, de l'analyse, de la commande et de la régulation des systèmes dynamiques. Elle a pour fondements théoriques les mathématiques, la théorie du signal et l'informatique théorique. L'automatique permet l'automatisation de différentes tâches de fonctionnement des machines et des chaines industriels. On parle alors de système asservi ou régulé, d'où est le plus répondu dans notre vie quotidienne, en particulier dans le domaine industriel, car il permet de réaliser plusieurs opérations sans l'intervention de l'être humain, pour de divers besoins, par exemple : convoyeurs industriels à navettes indexées, pilotage automatique de l'avion (autopiloté), asservissement d'angle pour des bras robotiques.

Dans la plupart des processus industriels, en particulier les moteurs électriques, il est indispensable de maîtriser certains paramètres physiques (vitesse, position, angle...etc.), il est donc très souvent nécessaire d'avoir recours à une commande. La commande avec le modèle inverse neuronal est une méthode qui a fait ses preuves et qui donne de bons résultats.

L'objectif de notre travail est d'implémenter une commande inverse neuronale pour un moteur à courant continu. La modélisation de la commande et du système (moteur MCC) est programmée sous l'environnement MATLAB.

Nous avons choisi de scinder notre travail en trois chapitres, suivis d'une conclusion générale.

- Le premier chapitre sera consacré à la présentation théorique détaillée des réseaux de neurones et leurs applications en automatique.

- Dans le deuxième chapitre, il y aura une présentation détaillée de l'algorithme de rétro-propagation de l'erreur qui est actuellement l'outil le plus puissant utilisé dans le domaine d'apprentissage des réseaux de neurones.

- Dans le troisième chapitre on modélisera le système à commander et on définira le réseau de neurones, puis la simulation et l'interprétation des résultats obtenus dans le cas de l'application du réseau de neurones pour la commande inverse neuronale.

Chapitre 1

Réseaux de neurones

1. Introduction

Les réseaux de neurones artificiels (RNA) sont des systèmes de traitement de l'information dont la structure s'inspire de celle du système nerveux. Ils sont destinés à effectuer des taches au quelles les approximateurs traditionnels sembles moins bien adaptés. Ainsi, les applications des réseaux de neurones artificiels à la reconnaissance de formes, à la modélisation, à la commande et à la classification ont pris une place importante au sein des réalisations industrielles.

Dans ce chapitre, nous allons introduire les réseaux de neurones en passant par l'historique de l'évolution de ces systèmes, et allant vers leurs différents types ainsi que leurs modes d'apprentissage.

2. Réseaux de neurones

2.1. Neurone biologique

Le neurone biologique est une cellule vivante spécialisée dans le traitement transmission et le traitement des influx nerveux. Les neurones sont reliés entre eux par des liaisons appelées axones. Ces axones eux même jouent un rôle important dans le comportement logique de l'ensemble. Ces axones conduisent les influx nerveux de la sortie d'un neurone vers l'entrée (synapse) d'un autre neurone. Les neurones font en quelque sorte une sommation des signaux reçus en entrée et en fonction du résultat obtenu fournissent un signal en sortie.

Le neurone biologique se compose de trois parties (Voir **Figure.1.1**) :

- **Le noyau** : ou cellule d'activation nerveuse, ou centre du neurone.
- **L'axone** : attaché au noyau qui est électriquement actif, ce dernier conduit l'impulsion générée par le neurone.
- **Dendrites** : électriquement passives, elles reçoivent les impulsions d'autres neurones.

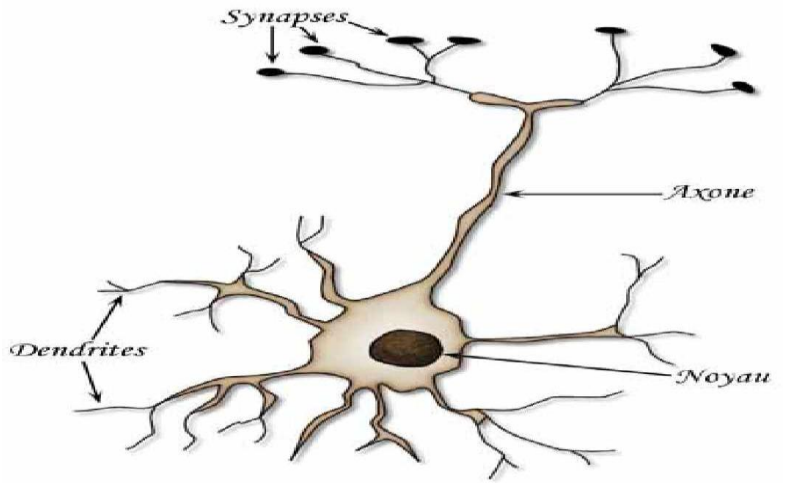

Figure.1.1. Neurone biologique

2.2. Neurones formel (artificiel)

Le neurone artificiel (ou cellule) et un processus élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones appartenant à un niveau situé en amont. A chacune des entrées est associé un poids *W* représentatif de la force de connexion. Chaque processus élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones appartenant à un niveau situé en aval. A chaque connexion est associé un poids (voir **figure.1.2**.).

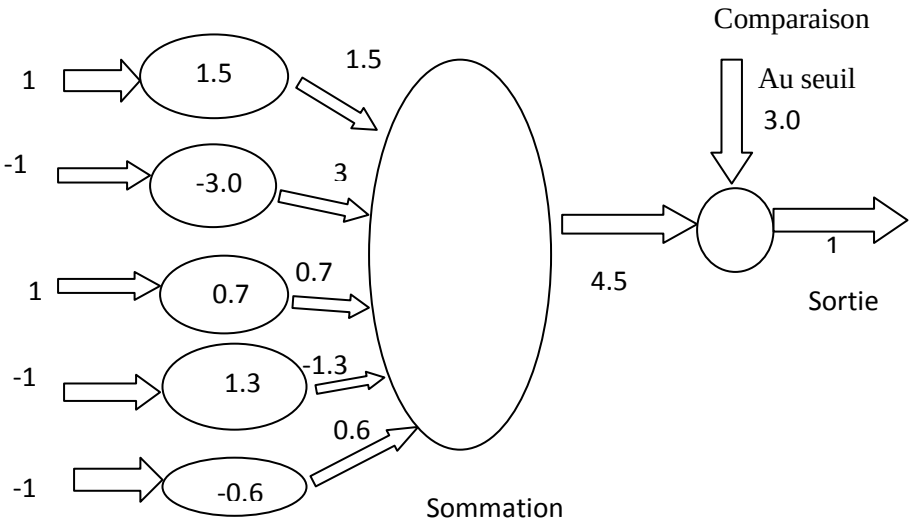

Figure.1.2. Neurones formel.

- **Modélisation d'un neurone formel**

Les réseaux de neurones formels sont à l'origine d'une tentative de modélisation mathématique du cerveau humain. Les premiers travaux datent de 1943 et sont l'œuvre de Mac Culloch et Pitts. Ils représentent un modèle assez simple pour les neurones et explorent les possibilités de ce modèle.

La modélisation consiste à mettre en œuvre un système de réseau neuronal sous un aspect non biologique mais artificiel, donc pour chaque élément composant le neurone biologique, il y a son correspondant en neurone formel.

Cette modélisation est résumée par le tableau suivant, qui permettra de voir clairement la transition du neurone biologique au neurone formel :

Neurones biologique	**Neurones formel**
Synapses	poids de connexions
Axones	signal de sortie
Dendrites	signal d'entrée
Noyau	fonction d'activation

Tableau.1.1. Analogie entre composants neuronaux biologiques et formels.

- **Entrées**

  Elles peuvent être :

  - Booléennes.
  - Binaires (0,1) ou bipolaires (-1,1).
  - Réelles.

- **Fonction d'activation :** Cette fonction permet de définir l'état interne du neurone en fonction de son entrée totale, voici ci-après quelques fonctions les plus utilisées :

  - **Fonction binaire à seuil :** fonction Heaviside définie par la fonction h illustrée par la **Figure.1.3.**

$$h(x) = \begin{cases} 1 & si\ x \geq 0 \\ 0 & sinon \end{cases} \quad (1.1)$$

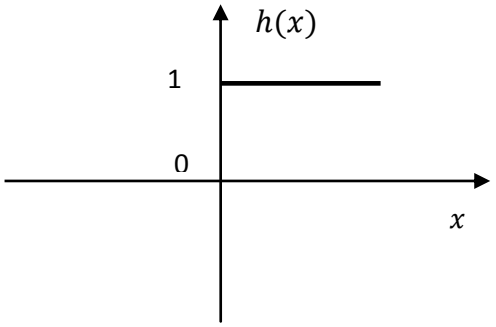

Figure.1.3. Fonction de Heaviside.

- **Fonction signe** : définie par la fonction $sgn(x)$ illustrée par la **figure.1.4**

$$sgn(x) = \begin{cases} +1 & si\ x \geq 0 \\ -1 & sinon \end{cases} \tag{1.2}$$

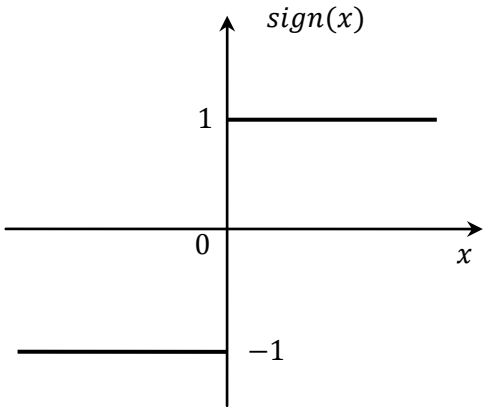

Figure.1.4. Fonction signe.

Le seuil introduit une non-linéarité dans le comportement du neurone, cependant il limite la gamme des réponses possibles à deux valeurs.

- **Fonction linéaire :** C'est l'une des fonctions d'activations les plus simples, elle est définie par :

$$f(x) = x \tag{1.3}$$

Elle est représentée à la **figure.1.5.**

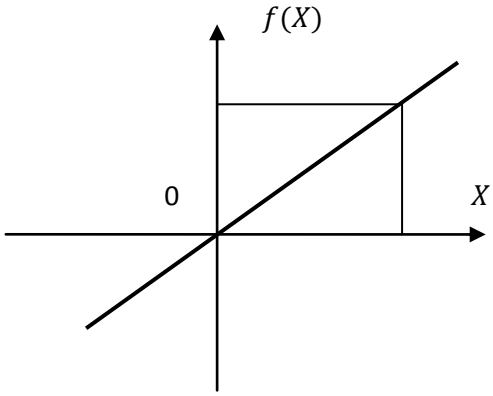

Figure.1.5. Fonction linéaire.

- **Fonction linéaire à seuil ou multi-seuils :** elle est définie comme suit :

$$F(x) = \begin{cases} x & x \in [u\ v] \\ v & si\ x \geq v \\ u & si\ x \leq u \end{cases} \tag{1.4}$$

Cette fonction représente un compromis entre la fonction linéaire et la fonction seuil ; entre ses deux barres de saturations, elle confère au neurone une gamme de réponses possibles. En modulant la pente de la linéarité, la plage de neurone est affectée. Elle est représentée à la **figure.1.6**.

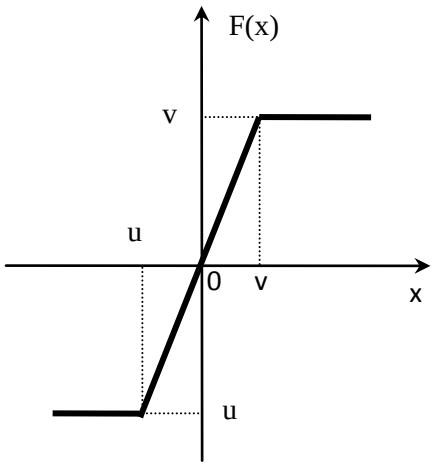

Figure.1.6. Fonction linéaire à seuil.

- **Fonction sigmoïde :** Elle est l'équivalent continu de la fonction linéaire. Etant continue, elle est dérivable, d'autant plus que sa dérivée est simple à calculer, elle est définie par la fonction suivante :

$$f(x) = \frac{1}{1+e^{-x}} \qquad (1.5)$$

Le tracé de cette fonction est donné par la **figure.1.7 :**

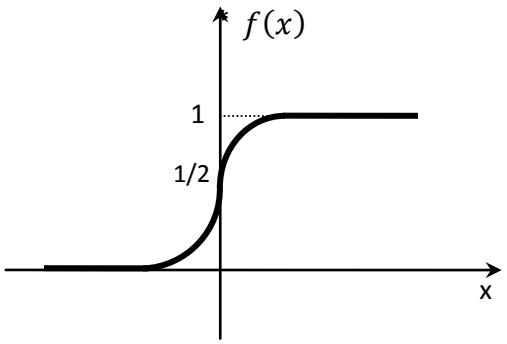

Figure.1.7. Fonction sigmoïde.

- **Fonction de sortie :**

Elle renvoie la sortie d'un neurone en fonction de son état d'activation. En général cette fonction est considérée comme la fonction identité. Elle peut être :

- Binaire (0,1) ou bipolaire (-1,1).
- Réelle.

3. Histoire des réseaux de neurones

Année	Auteur	Articles / Publications
1943	Mc Culloch & Pitts	Définition du neurone formel
1948	Von Neumann	Définition de l'automate cellulaire
1949	Hebb	Définition de la "Synaptic plasticity"
1960	Rosenblatt	Premiers travaux sur le Perceptron
1969	Minsky & Papert	Limites du Perceptron
1970	Conway	Turing et jeu de la vie
1982	Hopfield	Modélisation "Spin glasses"
1985	Hopfield	Résolution du problème du voyageur de commerce avec un réseau de neurones
1985	Rumelhart	Définition et expérimentation de l'algorithme de rétro-propagation du gradient
1986	Sejnowski	Définition et réalisation du réseau "NetTalk" d'apprentissage à la lecture

Tableau.1.2. histoire des réseaux de neurones.

Depuis, les réseaux de neurones deviennent un domaine où bouillonnent constamment de nouvelles théories, structures ainsi que de nouveaux algorithmes.

4. Objectifs des réseaux de neurones

Les réseaux de neurones servent aujourd'hui à toutes sortes d'applications dans divers domaines. Citant comme exemple, le développement d'un auto-pilote pour avion, ou encore un système de guidage pour automobile, conception des systèmes de lecture automatique de chèques bancaires et d'adresses postales, la production des systèmes de traitement du signal pour différentes applications militaires, des réseaux sont aussi utilisés pour bâtir des systèmes de vision par ordinateur, pour faire des prévisions sur les marchés monétaires, pour évaluer le risque financier ou en assurance, pour différents processus manufacturiers, pour le diagnostic médical, pour l'exploration pétrolière ou gazière, en robotique, en télécommunication...etc. Les réseaux de neurones ont aujourd'hui un impact considérable et, il y a fort à parier, que leur importance ira grandissant dans le futur.

5. Différents types de réseaux de neurones

Un neurone réalise simplement une fonction non linéaire, paramétrée, de ses entrées. L'intérêt des neurones réside dans les propriétés qui résultent de leur association en réseaux, c'est-à-dire de la composition des fonctions non linéaires réalisées par chacun des neurones.

Essentiellement, on distingue deux familles de réseau de neurones à savoir :

5.1. Réseaux de neurones non bouclés

Un réseau de neurones non bouclé peut donc être imaginé comme un ensemble de neurones connectés entre eux, l'information circule des entrées vers les sorties sans retour en arrière. Le réseau peut alors être représenté par un graphe *acyclique* dont les nœuds sont les neurones et les arêtes les connexions entre ceux-ci. En se déplaçant dans le réseau, à partir d'un neurone quelconque, en suivant les connexions et en respectant leurs sens, il n'y aura pas de retour au neurone de départ.

La représentation de la topologie d'un réseau par un graphe est très utile, notamment pour les réseaux bouclés, comme à la section «**Réseaux de neurones bouclés**». Les neurones qui effectuent le dernier calcul de la composition de fonctions sont les ***neurones de sortie*** ; ceux qui effectuent des calculs intermédiaires sont les ***neurones cachés*** (voir **figure.1.1**).

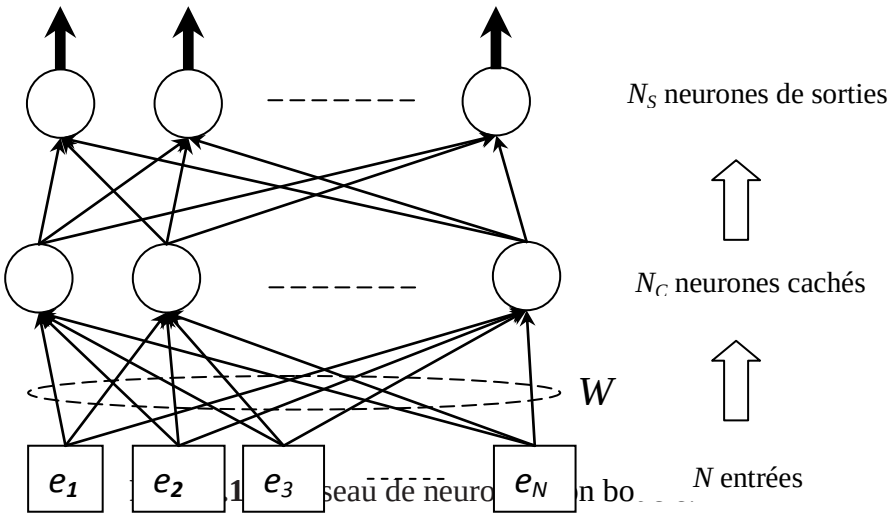

Figure 1.1 : Réseau de neurones non bouclé

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

$$Y(k) = \phi(E(k), W) \tag{1.6}$$

Où $Y(k) \in R^{N_S}$ est le vecteur de sorties à l'instant k, $E(k) \in R^N$ est le vecteur des entrées et $\phi : \mathrm{R}^N \to \mathrm{R}^{N_S}$ est la fonction non linéaire réalisée par les neurones du réseau.

Le temps ne joue aucun rôle fonctionnel dans un réseau de neurones non bouclé : si les variables sont indépendantes du temps, les sorties le sont également. Le temps nécessaire pour le calcul de la fonction réalisée par chaque neurone est négligeable et, fonctionnellement, on peut considérer ce calcul comme instantané. Pour cette raison, les réseaux non bouclés sont souvent appelés «**Réseaux statiques**», par opposition aux réseaux bouclés qui sont dit «**Dynamiques**».

Toute fonction $\rho(x)$ à valeur bornées, continue ou non, dans un intervalle fermé $[X_A, X_B]$ peut être approchée, à ε près, avec la fonction réalisée par un RNF à une couche cachée d'un nombre fini de neurones à fonction d'activation sigmoïde exponentielle. Ceci est vrai aussi pour toute fonction d'activation universelle, c'est-à-dire toute fonction non polynomiale. En particulier, les fonctions dont les limites en-∞ et +∞ sont finies et distinctes, dites sigmoïdales, sont universelles. La précision peut toujours être améliorée en augmentant le nombre de neurones cachés. C'est ce que l'on appelle la propriété d'approximation universelle des RNF qui a été démontrée par [Cybenko, 1989].

5.2. Réseaux de neurones bouclés

L'architecture la plus générale, pour un réseau de neurones, est celle des « réseaux bouclés », dont le graphe des connexions est *cyclique* : en se déplaçant 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 (un tel chemin est désigné sous le terme de « cycle »). La sortie d'un neurone du réseau peut donc être fonction d'elle même ; ceci n'est évidemment concevable que si la notion de *temps* est explicitement prise en considération.

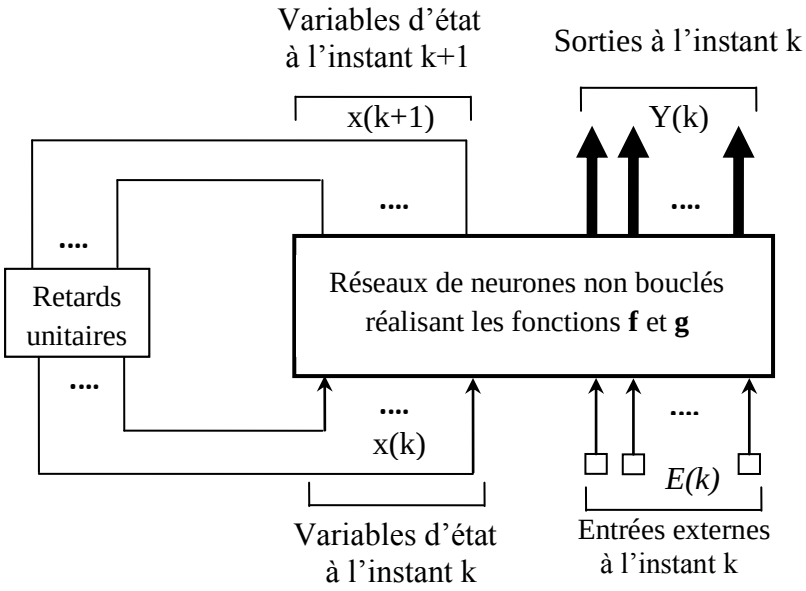

Figure.1.9 .Forme canonique d'un réseau de neurones bouclé.

Ainsi, à chaque connexion d'un réseau de neurones bouclé (ou à chaque arête de son graphe) est attaché, outre un paramètre comme pour les réseaux non bouclés, un *retard,* multiple entier (éventuellement nul) de l'unité de temps choisie. Une grandeur, à un instant donné, ne pouvant pas être fonction de sa propre valeur au même instant, tout cycle du graphe du réseau doit contenir au moins une arête dont le retard n'est pas nul.

Un réseau de neurones bouclé à temps discret réalise une (ou plusieurs) équation(s) aux différences non linéaires, par composition des fonctions réalisées par chacun des neurones et des retards associés à chacune des connexions.

A la fin des années 1980, les RNF bouclés ont commencé à être utilisés comme modèles dynamiques de processus, et comme filtres non linéaires.

Tout réseau de neurones bouclés est un système dynamique non linéaire que l'on peut mettre sous la forme d'une représentation d'état faisant intervenir un (ou des) réseau(x) de neurones non bouclé(s) :

$$\begin{aligned} x(k+1) &= f(x(k), E(k), w) \\ Y(k) &= g(x(k), E(k), w) \end{aligned} \tag{1.7}$$

Où k est l'instant discret, $E(k)$ est le vecteur des entrées externes du système, $Y(k)$ est le vecteur des sorties, et $x(k)$ est le vecteur des variables d'état, à l'instant k. Les fonctions f et g sont réalisées par les neurones des réseaux, interconnectés avec les paramètres W Cette représentation d'état est appelée *forme canonique*, et est particulièrement utile pour une formulation simple des algorithmes d'apprentissage. Elle est illustrée par la **Figure.1.9.**

5.3. Réseau de neurones de type perceptron multicouches

L'idée principale du perceptron multicouches (noté MLP pour *Multi Layer Perceptron*) est de grouper des neurones en couches. La première couche est reliée aux entrées, puis ensuite chaque couche est reliée à la couche précédente. C'est la dernière couche qui produit les sorties du MLP. Il a été ainsi démontré qu'un perceptron multicouche avec une seule couche cachée pourvue d'un nombre suffisant de neurones, peut approximer n'importe quelle fonction avec la précision souhaitée.

Néanmoins, cette propriété ne permet pas de choisir, pour un type de fonction donnée, le nombre optimal de neurones dans la couche cachée. Autrement dit, ce résultat ne mène pas vers une technique de construction d'architecture.

La **Figure 2.10**, représente la structure d'un MLP à une seule couche cachée.

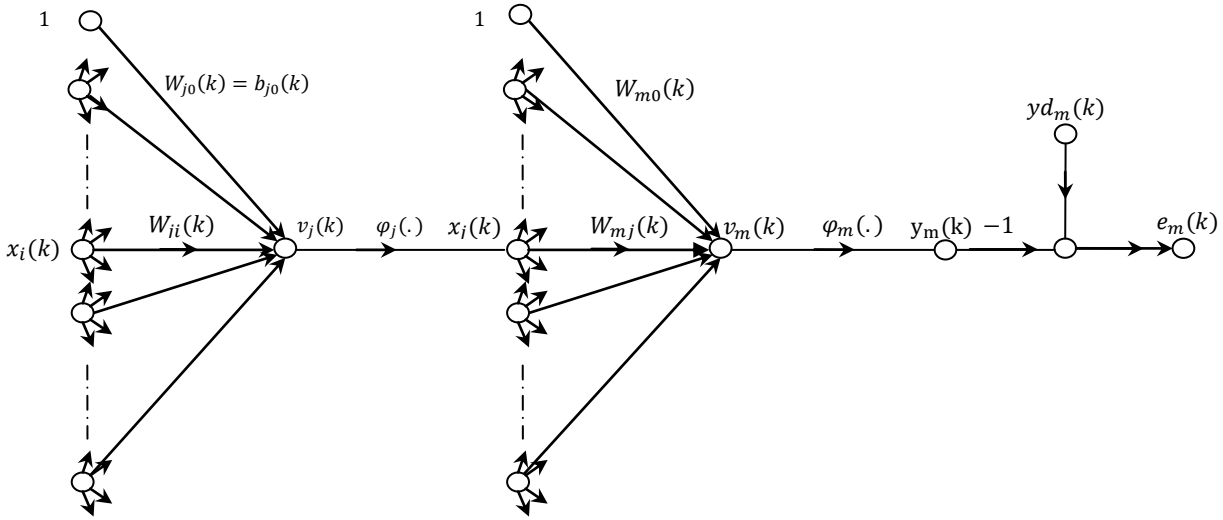

Figure.1.10. Structure du MLP à une seule couche cachée.

Dans le cas où une seule couche cachée est présente, les fonctions discriminantes réalisées par un tel réseau de neurones sont de la forme :

$$y_m(k) = \phi_m(v_m) = \phi_m(\sum_{j=0}^{n_j} w_{mj}(k) x_j(k))$$

$$y_m(k) = \phi_m(\sum_{j=0}^{n_j} w_{mj}(k) \phi_j(\sum_{i=0}^{n_i} w_{ji}(k) x_i(k)))$$

$$y_m(k) = \phi_m(\sum_{j=1}^{n_j} w_{mj}(k) \phi_j(\sum_{i=1}^{n_i} w_{ji}(k) x_i(k) + b_{j0}(k)) + b_{m0}(k)) \qquad (1.8)$$

Avec :

$x_i(k)$, $x_j(k)$ et $y_m(k)$: représentent respectivement les valeurs à la sortie des neurones de la couche d'entrée, de la couche cachée et de la couche de sortie,

$\varphi_m(.) et\ \varphi_j(.)$: Les fonctions d'activation des neurones de la couche de sortie et de la couche cachée (généralement, φ_m est une fonction linéaire et φ_j est une fonction non linéaire),

$b_{m0}(k)$ et $b_{j0}(k)$: Les biais des neurones de la couche de sortie et de la couche cachée.

n_i et n_j : Le nombre des neurones de la couche d'entrée et de la couche de sortie.

W_{ji} : Le poids synaptique (ou de connexion) entre le neurone amont i , et le neurone aval j

$yd\ (k)$: représente la valeur de la sortie désirée du système à l'instant k .

5.4. Réseau de neurones à espace d'état

Les réseaux de neurones à espace d'état (State-Space Neural Network, ou SSNN) est un paradigme de réseau de neurones qui permet d'assurer la stabilité, et d'analyser le comportement dynamique avec les poids synaptiques du réseau.

Il possède une architecture de réseau de neurones récurrent (RNR), dont la structure reflète exactement une modélisation dans l'espace d'état d'un système dynamique non linéaire. (Zamarreno et al. 2000).

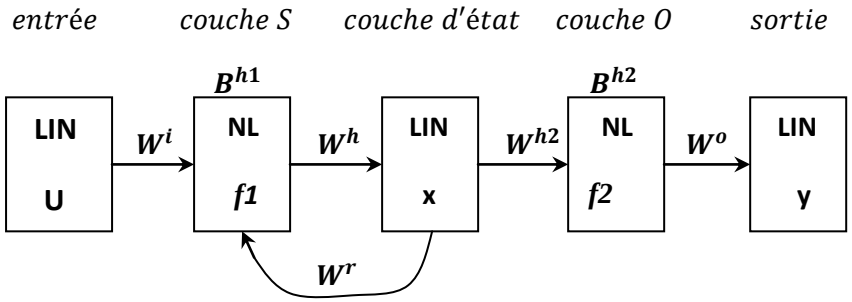

Figure.1.11. Schéma bloc d'un réseau de neurones à espace d'état.

LIN : éléments linéaire,
f1, f2 : fonctions non linéaires.

Le schéma bloc représenté par la **Figure 1.11**, est équivalent à la représentation d'état d'un système non linéaire déterministe qui a pour représentation mathématique :

$$\begin{cases} x(k+1) = F(x(k), u(k)) \\ y(k) = G(x(k)) \end{cases} \tag{1.9}$$

Où $x \in R^s$ est le vecteur des états, $u \in R^n$ est le vecteur d'entrée, $y \in R^m$ est le vecteur de sortie, $F : R^s \times R^n \to R^s$, et $G : R^s \to R^m$, sont deux fonctions non linéaires statiques.

La **Figure f.11** représente un réseau de neurones à espace d'état, ce réseau de neurones récurrent spécial est composé de cinq couches :

- **La couche d'entrée** *:* cette couche fournit les entrées de commandes à chaque neurone de la couche suivante, chaque entrée de commande est pondérée par le poids W^i .
- **La couche cachée (S)** *:* cette couche représente la fonction non linéaire qui représente le comportement des états, la lettre **(S)** vient du mot « State ».
- **La couche d'état** *:* chaque neurone de cette couche représente un état
- **La couche cachée (O)** *:* c'est la couche avant la couche de sortie, elle représente la fonction non linéaire qui relie la sortie du réseau aux états. La lettre **(O), «** vient du mot Output ».
- **La couche de sortie** *:* relie les signaux de la couche cachée (O) à chaque neurone de sortie, ces sorties sont finalement les sorties du SSNN.

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

1. Etant un modèle neuronal, il a la flexibilité de représenter n'importe quelle fonction non linéaire.
2. Etant un modèle d'espace d'état, le nombre de connexions extérieures est minimal, donc le nombre de paramètre à ajuster est minimal. Les entrées /sorties du modèle neuronal sont les entrées/sorties du système physique.

Il a été prouvé que n'importe quelle fonction non linéaire peut être estimée par un réseau de neurones contenant une seule couche cachée, composée de neurones dont la fonction de transfert sigmoïdale bouclé. Ainsi deux réseaux de neurones interconnectés peuvent représenter le système d'équation (1.9), l'un représente la fonction qui donne le comportement d'état et l'autre représente la fonction qui relie les sorties aux états. Ainsi le modèle (1.9) peut être représenté comme suit : (Zamarreno et al. 2000) :

$$\begin{cases} \hat{x}(k+1) = w^h \cdot f_1(w^r \cdot \hat{x}(k) + w^i u(k) + B^{h1}) \\ \hat{y}(k) = w^0 \cdot f_2(w^{h2} \cdot \hat{x}(k) + B^{h2}) \end{cases} \tag{1.10}$$

Où

$-$ W$^{\boldsymbol{h}}$, W^r, W^i, W$^{\boldsymbol{o}}$, et W^{h2} sont des matrices de dimensions $s \times h$, $h \times s$, $h \times n$, $m \times h2$, et $h2 \times s$ respectivement.

$-$ B$^{\boldsymbol{h}}$, B^{h2}sont deux vecteurs de biais avec h et $h2$, éléments qui ont pour rôle d'augmenter ou d'abaisser l'entrée nette de la fonction d'activation.

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

La **Figure 1.12** est une réalisation de l'équation (1.10), elle montre clairement les couches du réseau ainsi que les liens entre les neurones.

La différence entre ce type de réseaux et les réseaux de neurones classiques, tels que le réseau multicouches précédent, c'est que le réseau multicouches apprend et estime une fonction ou une relation entre un espace d'entrée (par exemple l'espace des entrée de commande $u(k)$d'un système) et un espace de sortie (par exemple l'espace de sorties $y(k)$ d'un système) sans tenir compte du comportement interne du système.

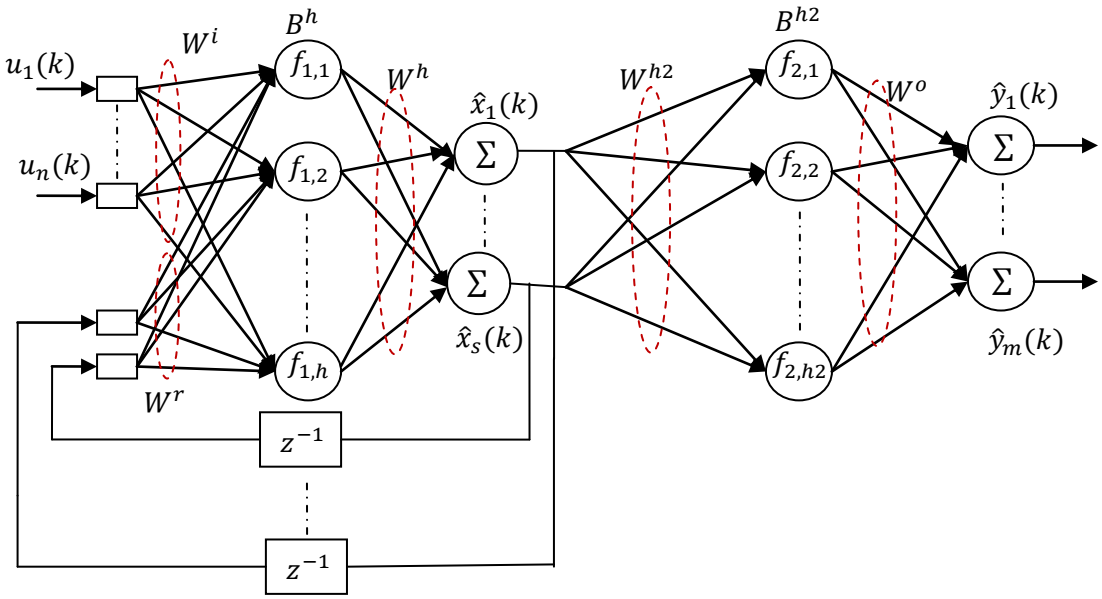

Figure 1.12 .Réseau de neurones à espace d'état.

Par contre la structure d'un SSNN cherche à estimer les deux sous fonctions qui représentent les transformations non linéaires de la représentation d'état. C'est-à-dire apprendre d'abord une relation entre un espace d'entrée (entrée de commande $u(k)$), et un

espace interne (les variables d'états $x(k)$), et ensuit apprendre une autre relation entre un espace interne (les variables d'états $x(k)$), et l'espace de sortie (sortie $y(k)$) ; dans le cas où les grandeurs internes ne sont pas accessibles (mesurables), il sera alors difficile d'estimer les sous fonctions.

6. Apprentissage des réseaux de neurones

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

L'apprentissage neuronal fait appel à des exemples de comportement. Durant cette phase de fonctionnement, le réseau adapte sa structure (le plus souvent, les poids des connexions) afin de fournir sur ses neurones de sortie les valeurs désirées.

Après l'initialisation des poids du réseau, il y a présentation des exemples au réseau et calcul des sorties correspondantes à partir desquelles une valeur d'erreur ou de correction sera calculée ce qui va induire l'application d'une correction des poids.

6.1. Apprentissage non supervisé

L'apprentissage est qualifié de non supervisé lorsque seules les valeurs d'entrée sont disponibles. Dans ce cas les exemples présentés à l'entée provoquent une auto-adaptation du réseau afin de produire des valeurs de sorties qui soient proches en réponse à des valeurs d'entrée similaires (de même nature).

6.2. Apprentissage supervisé

L'apprentissage est dit supervisé lorsque les exemples sont constitués de couples de valeurs du type : (valeur d'entrée / valeur de sortie). Tout le problème de l'apprentissage supervisé consiste, étant donné un ensemble d'apprentissage E de N couples (entrées / sorties désirées) (u_i, y_i) i=1,2,…. N, à déterminer le vecteur des poids W d'un réseau F_W capable de mettre ces informations en correspondance, c'est à dire un réseau tel que :

$$f_w(u_i) = y_i, \qquad i = 1, 2, ..., N \tag{1.11}$$

Pour l'apprentissage supervisé de ce réseau on peut appliquer les méthodes d'apprentissage de premier ou de deuxième ordre basées sur le calcul du gradient.

- **Méthodes du premier ordre :** les méthodes dites du premier ordre sont basées sur le calcul du gradient de la fonction du coût, donnée par la relation (1.12), et le minimum de cette fonction est atteint si son gradient est nul. Parmi ces méthodes, on cite la méthode du gradient simple.

$$J(w) = \sum_{K=1}^{N} J^{K}(w) = \sum_{K=1}^{N} \frac{1}{2}(e^{K}(w))^{2} = \sum_{K=1}^{N} \frac{1}{2}(y_{d}^{K} - y^{K})^{2}(w) \qquad (1.12)$$

Où : $J^{K}(w)$ est le coût partiel relatif à l'exemple k.

- **Méthodes du deuxième ordre :** les méthodes du deuxième ordre sont ainsi appelées parce qu'elles prennent en considération la dérivé seconde de la fonction de coût (1.12). Parmi ces méthodes, on cite la méthode de Levenberg Marquardt (Lucea 2006).

7. Normalisation des données

Afin d'améliorer la performance des réseaux neuronaux multicouches, il est préférable de normaliser les données d'entrée et de sortie de telle sorte qu'elles se trouvent dans l'intervalle $[-1, 1]$ ou $[0, 1]$.

- **Une des méthodes de normalisation des données**

- "***Rescaling***": simple réduction de chaque variable d'entrée *Xj* dans [0, 1].

Exemple :

$$X_{j} = \frac{X_{j} - \min(X_{j})}{\max(X_{j}) - \min(X_{j})}$$

Où max(.) et min(.) sont les valeurs min. et max. établies sur les données d'apprentissage.

8. Application des réseaux de neurones en automatique

8.1. Traitement d'image

- Compression des images médicales fixes
- Reconnaissance de caractères et de signatures
- Reconnaissance de formes et de motifs

8.2. Traitement de signal

- traitement de la parole
- identification de sources
- filtrage

8.3.Commande de processus

- Pilotage de robots mobiles
- Commande d'un véhicule autonome
- Commande des machines asynchrones

9. Un des types d'excitation utilisée en commande de processus

- **Les SBPA**

La SBPA (Séquence Binaire Pseudo Aléatoire) est un signal formé d'impulsions rectangulaires modulées aléatoirement en longueur, qui approxime un bruit blanc discret, donc riche en fréquence et de valeur moyenne nulle, ne modifiant pas le point de fonctionnement du procédé. Elle est couramment utilisée dans les procédures d'identification.

La génération de la SBPA se fait à l'aide de registres à décalage (réalisés en matériel ou logiciel) bouclés. La longueur maximale d'une séquence est $2N - 1$ où N est le nombre de cellules du registre à décalage. La figure suivante nous présente la génération d'une SBPA de longueur $7 = 2^3$-1 obtenue à l'aide d'un registre à décalage ayant 3 cellules.

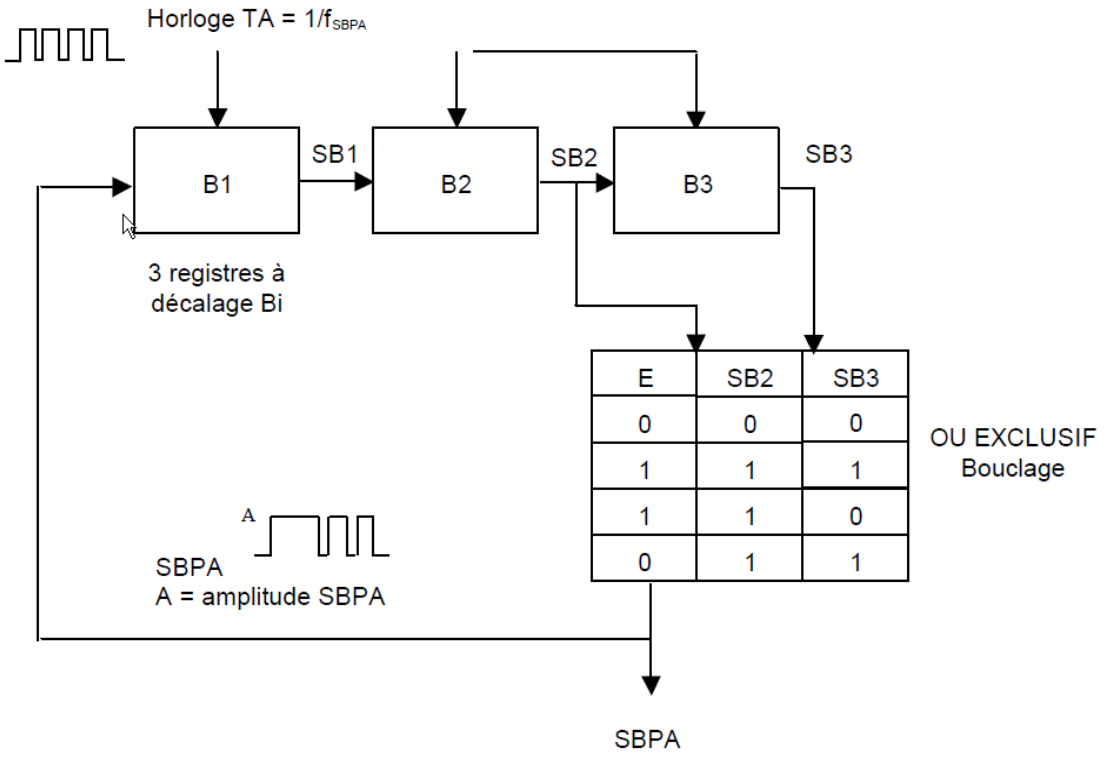

Figure 1.13. Génération d'une SBPA à base de registres à décalage

- **Chronogramme**

T_A	$SB1$	$SB2$	$SB3$	E	$SBPA = A \times E$
0	**1**	**1**	**1**	**0**	**-A**
$1T_A$	**0**	**1**	**1**	**0**	**-A**
$2T_A$	**0**	**0**	**1**	**1**	**+A**
$3T_A$	**1**	**0**	**0**	**0**	**-A**
$4T_A$	**0**	**1**	**0**	**1**	**+A**
$5T_A$	**1**	**0**	**1**	**1**	**+A**
$6T_A$	**1**	**1**	**0**	**1**	**+A**
$7T_A = 0\ T_A$	**1**	**1**	**1**	**0**	
$8T_A = 1T_A$	**0**	**1**	**1**	**0**	
…..					
……					

Le tableau suivant nous donne pour différents nombres N de cellules, la structure des bouclages ou les bits à additionner (ou exclusif) permettant de générer des SBPA de longueur $L = 2^n - 1$:

Nombres de cellules N	Longueur de la séquence L=2^N-1	Bits additionnés Bouclage B_I et B_j
2	3	1 et 2
3	7	2 et 3
4	15	3 et 4
5	31	4 et 5
6	63	5 et 6
7	127	4 et 7
8	255	2.3.5 et 8
9	511	5 et 9
10	1023	7 et 10
11	2047	9 et 11

Notons aussi un élément caractéristique très important des SBPA : la durée maximale d'une impulsion de la SBPA est égale à N (nombre de cellules). Cette propriété intervient dans le choix des SBPA pour l'identification.

10. Conclusion

Le réseau de neurones artificiels est un ensemble de neurones formels interconnectés les un aux autres par des liaisons pondérées, et selon la façon dont ces dernières sont effectuées on peut distinguer deux grandes familles de réseaux de neurones : le réseau bouclé et le réseau non bouclé.

Ce chapitre est dédié à l'étude des réseaux de neurones artificiels, En premier lieu, nous avons rapporté quelques définitions de bases sur le neurone biologique ainsi que sur le neurone formel ou artificiel. Ensuite une étude a été effectuée sur ce dernier.

Chapitre 2

Modele inverse neuronal

1. Introduction

Pour réaliser un système de commande avec modèle inverse neuronal, nous devons tout d'abord choisir une architecture et un algorithme d'apprentissage, et de commande.

Dans ce chapitre, nous allons présenter les structures d'apprentissage et de commande du modèle inverse neuronal. Ainsi que l'algorithme de rétro-propagation de l'erreur.

2. Modèle neuronal inverse

L'identification du modèle neuronal inverse commence par la détermination du vecteur d'entrée, à savoir le nombre des retards en sorties et entrées, ceci est lié à l'ordre du système. La deuxième étape est l'architecture du réseau (nombre de couches cachées et nombre de neurones). La détermination des paramètres du modèle est effectuée par un apprentissage du réseau selon trois architectures.

3. Architectures d'apprentissages

3.1. Architecture Générale d'Apprentissage

Dans la première architecture (Figure.2.2), le signal *u* est appliqué à l'entrée de système, produisant une sortie y qui est fournie au réseau. La différence entre le signal d'entrée *u* et la sortie du réseau de neurone û est l'erreur rétro-propagée à travers le réseau qui va servir pour l'apprentissage hors ligne du réseau. On tente de minimiser le critère E(w):

$$E(w) = \sum_{k=1}^{N} \frac{1}{2}(u(k) - \grave{u}(k,w))^2 \qquad (2.15)$$

Où w est un vecteur contenant les poids du réseau inverse.

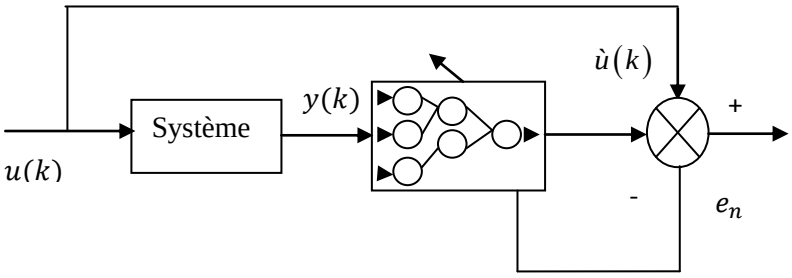

Figure.2.1. Architecture générale d'apprentissage.

Cette méthode présente plusieurs inconvénients : Les sorties y du système utilisées dans l'apprentissage ne garantissent pas que les sorties du modèle neuronal vont être dans des régions voulues pour le succès de son utilisation dans la commande. Si le système à commander est multi- variable, le modèle ainsi retenu peut ne pas imiter le système réel.

3.1. Architecture Indirecte d'Apprentissage

Cette approche est une mise en œuvre particulière de l'architecture précédente dans laquelle, le modèle inverse en cours d'apprentissage sert aussi à commander le système. La consigne ***r*** est fournie au premier réseau qui produit une commande û au système, la sortie de celui-ci est passée comme consigne à la seconde copie du modèle inverse, qui produit alors une commande ***û.*** La différence entre ***u*** et ***û*** sert de signal d'erreur afin d'effectuer l'apprentissage des paramètres du modèle par rétro-propagation. Cette architecture, est présentée sur la (Figure.2.2).

L'idée derrière cette architecture est que la minimisation de l'erreur commise sur la commande entraînera une minimisation de l'erreur en sortie du système. Cependant, une erreur nulle sur la commande ne provoque pas nécessairement l'annulation de l'erreur totale en sortie du système.

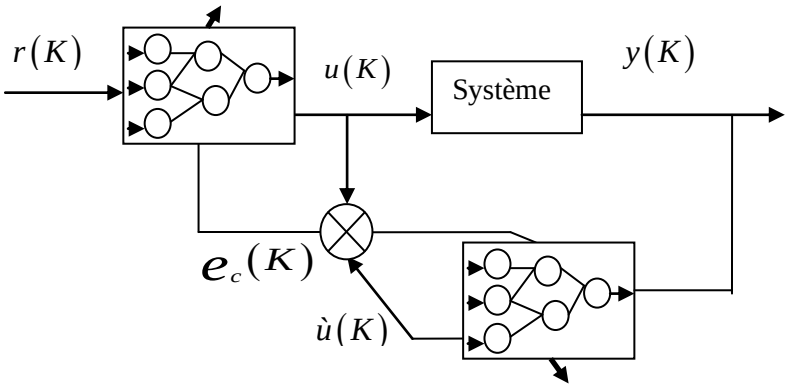

Figure.2.2. Architecture indirecte d'apprentissage.

3.2. Architecture spécialisé d'apprentissage

Contrairement aux deux précédentes architectures, l'architecture spécialisée, procède par l'utilisation de l'erreur e_c équation (2.16), la différence entre la sortie désirée r et la sortie réelle du système, pour apprendre au modèle neuronal à suivre la dynamique inverse du système **Figure.2.3**

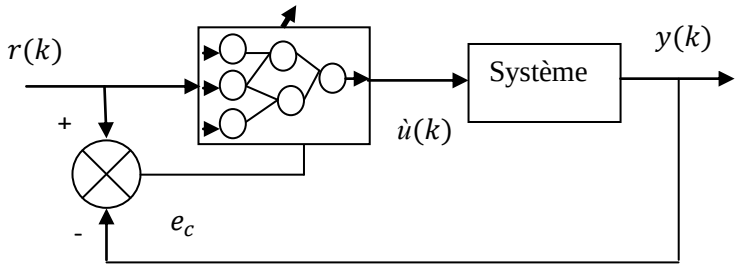

Figure.2.3. Architecture spécialisée d'apprentissage.

$$e_c(k) = r(k) - y(k) \tag{2.16}$$

Une méthode de gradient peut être utilisée :

$$w(k) = w(k-1) - \eta \frac{\partial\left(\frac{1}{2}e_c^2(k)\right)}{\partial w} \tag{2.17}$$

Où η est le pas d'apprentissage, le gradient s'écrit :

$$\frac{\partial\left(\frac{1}{2}e_c^2(k)\right)}{\partial w} = -e_c(k)\frac{\partial\left(y(k)\right)}{\partial w} = -e_c(k)\frac{\partial\left(y(k)\right)}{\partial\left(u(k-1)\right)}\frac{\partial\left(u(k-1)\right)}{\partial w} \tag{2.18}$$

Ce calcul requiert la connaissance du Jacobien du système, à cette fin, plusieurs méthodes ont été envisagées pour l'approximer. On peut remarquer que la valeur utilisée pour l'apprentissage des paramètres du modèle inverse est l'erreur en sortie du procédé. Une utilisation adéquate de la rétro propagation imposerait pourtant que l'on rétro propage l'erreur en sortie du modèle, qui est naturellement inconnue. Quoiqu'il en soit l'expérience montre qu'en général, cette approche est inefficace.

De plus, le modèle connexionniste étant utilisé dés l'initialisation pour commander le procédé, il est conseillé de disposer au départ d'un modèle relativement cohérent pour que cette architecture puisse fonctionner.

4. Commande neuronale

Parmi les structures de commande neuronales utilisant le modèle inverse, deux sont présentées dans les paragraphes qui suivent

4.1. Commande neuronale directe par modèle inverse

Comme son nom l'indique, le modèle neuronal inverse placé en amant du système, est utilisé comme contrôleur pour commander le système en boucle ouverte**.**

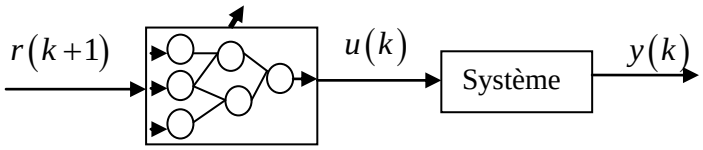

Figure.2.4. Commande directe par modèle inverse.

La valeur de y(k+1) est substituée par la sortie désirée r(k+1) et on alimente le réseau par les valeurs retardées de u(k) et y(k). Si le modèle neuronal est exactement l'inverse du système alors il conduit la sortie à suivre la consigne.

4.2. Commande neuronal hybride

La deuxième approche de commande est celle qui fait intervenir le réseau de neurones au sein d'une structure de commande à lequel participe aussi un contrôleur classique existant. Elle est présentée dans le cadre de la manipulation d'un bras de robot. Elles consistent globalement à faire fonctionner simultanément un contrôleur feedback conventionnel et un modèle connexionniste Cette approche ne vise plus à apprendre la dynamique inverse du système, mais à effectuer une régulation consistante de celui-ci. On fournit au régulateur feedback classique ainsi qu'au réseau de neurones l'erreur e_c commise

en sortie du système. Le réseau dispose de plus de la consigne r(k) Le signal de commande total est constitué de la somme des sorties des deux dispositifs. L'apprentissage des poids synaptiques est quant à lui réalisé par rétro propagation de la sortie du régulateur classique.

Cette structure est représentée sur figure2.5. L'approche est justifiée par le fait que lorsque la régulation opérée est consistante, la sortie du régulateur classique est nulle, confiant ainsi la tache de régulation au réseau de neurone. Il s'agit d'une optimisation du fonctionnement du modèle neuronal par d'autres techniques.

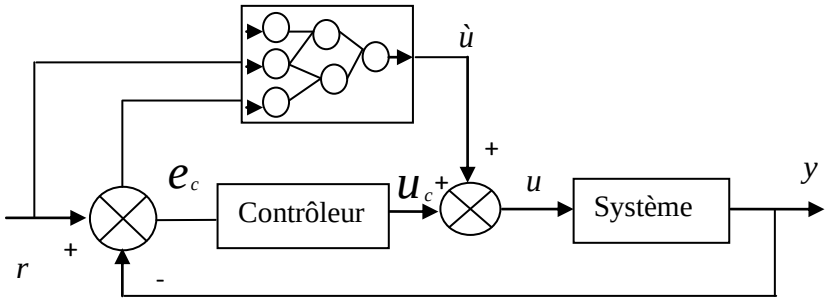

Figure.2.5. Structure de commande neuronale hybride.

5. Algorithmes d'apprentissage des réseaux de neurones

Il en existe pas mal d'algorithmes d'apprentissage des réseaux de neurones parmi eux :

- Algorithme de rétro-propagation du gradient
- Méthode de Levenberg-Marquardt (LM)
- Loi de HEBB

Dans ce qui suit nous allons se basé sur l'algorithme de rétro propagation du gradient pour une étude détaillée.

5.1. Algorithme de la rétro propagation

Avant de définir la règle d'apprentissage, on doit définir la relation entre les sorties du réseau d'une part, et les entrées et les poids d'autre part.

Dans un réseau à (l) couches ayant (n) entrées et (m) sorties les états des différents neurones sont donnés par :

$$U_i^l(k) = f^l\left(P_i^l(k)\right) \tag{2.1}$$

Avec :

$$P_i^l(k) = \sum_{j=1}^{N_{l-1}} W_{ij}^l U_j^{l-1}(k) \tag{2.2}$$

Où :

$i = 1, 2, \ldots\ldots N_l$

$J = 1, 2, \ldots\ldots N_{l-1}$

$q = 1, 2, \ldots\ldots N_{l+1}$

$U_i^l(k)$: Sortie du neurone i de la couche l.

$W_{ij}^l(k)$ **:** Coefficient synaptique (poids) de la j^{eme} entrée du neurone (i) de la couche (l).

$$U_i^0(k) = X_i(k) \qquad i = 1, 2, \ldots\ldots\ldots, n \tag{2.3}$$

$$U_i^l(k) = Y_i(k) \qquad i = 1, 2\ldots\ldots\ldots\ldots, m \tag{2.4}$$

Où :

$X_i(k)$ et $Y_i(k)$ sont respectivement les entrées et les sorties du réseau.

L'objectif de la méthode de la rétro-propagation est d'adapter les paramètres $W_{ij}^{l}(k)$ de façon a minimisé une fonction de coût donné par :

$$E(W)=\sum_{P=1}^{T}E_P(W) \tag{2.5}$$

Avec :

$$E_P(W)=\frac{1}{2}\sum_{i=1}^{m}\left[y_i^d(k)-y_i(k)\right]^2 \tag{2.6}$$

Où :

$y^d(k)$ est le vecteur de sortie désiré,

$y(k)$ le vecteur de sortie de réseau et *T* le nombre d'exemples ou longueur de l'ensemble d'entraînement.

6. Principe de la retro-propagation

L'approche la plus utilisée pour la minimisation de la fonction *E* est basée sur la méthode du gradient. On commence l'entraînement par un choix aléatoire des vecteurs initiaux du poids.

On présente le premier vecteur d'entrée, une fois on a la sortie du réseau, l'erreur correspondante et le gradient de l'erreur par rapport à tous les poids sont calculés. Les poids sont alors ajustés. On refait la même procédure pour tous les exemples d'apprentissage. Ce processus est répété jusqu'à ce que les sorties du réseau soient suffisamment proches des sorties désirées.

6.1. Adaptation des poids

L'adaptation des poids se fait par la méthode du gradient basée sur la formule itérative suivante :

$$W_{ij}^{l}(k+1) = W_{ij}^{l}(k) - \Delta W_{ij}^{l} \tag{2.7}$$

Avec **:**

$$\Delta W_{ij}^{l} = lr.\frac{\partial E(W)}{\partial W_{ij}^{l}(k)} \tag{2.8}$$

Où :

k : représente le numéro d'itération.

lr **:** est une constante appelée facteur ou pas d'apprentissage.

La vitesse de convergence dépend de la constante Ir ; qui sa valeur est généralement choisie expérimentalement.

La dérivée de la fonction du coût par rapport au poids W_{ij}^{l} est données par :

$$\frac{\partial E(W)}{\partial W_{ij}^{l}(k)} = \sum_{P=1}^{T} \frac{\partial E_P(W)}{\partial W_{ij}^{l}(k)} \tag{2.9}$$

$$\frac{\partial E_P(W)}{\partial W_{ij}^{l}(k)} = \frac{\partial E_P(W)}{\partial U_{i}^{l}(k)} \cdot \frac{\partial U_{i}^{l}(k)}{\partial W_{ij}^{l}(k)} \tag{2.10}$$

Pour la couche de sortie :

$$\frac{\partial E_P(W)}{\partial U_{i}^{l}(k)} = -\left(y_i^d(k) - y_i(k)\right) \tag{2.11}$$

Pour les couches cachées:

$$\frac{\partial E_P(W)}{\partial U_{i}^{l}(k)} = \sum_{q=1}^{N_{l+1}} \frac{\partial E_P(W)}{\partial U_{q}^{l+1}(k)} \cdot \frac{\partial U_{q}^{l+1}(k)}{\partial U_{i}^{l}(k)} \tag{2.12}$$

$$\frac{\partial U_{i}^{l}(k)}{\partial W_{ij}^{l}} = f^{l'}\left(P_i^l(k)\right) \cdot U_j^{l-1}(k) \tag{2.13}$$

Donc l'expression (2.10) s'écrit sous la forme :

$$\frac{\partial E_p(W)}{\partial W_{ij}^l} = \frac{\partial E_P(W)}{\partial U_j^l(k)} \cdot f^{l'}\left(P(k)\right) \cdot U_j^{l-1}(k) \tag{2.14}$$

Pour minimiser l'erreur totale sur l'ensemble d'entraînement, les poids du réseau doivent être ajustés après présentation de tous les exemples.

7. Etape de l'algorithme de la retro propagation

Etape 1**:** Initialiser les poids W_{ij}^l et les seuils internes des neurones à des petites valeurs aléatoires.

Etape 2 **:** Calculer le vecteur d'entrée et de sortie désirée, correspondant.

Etape 3 **:** Calculer la sortie du réseau en utilisant les expressions (2.1) et (2.2)

Etape 4 **:** Calculer l'erreur de sortie en utilisant l'expression (2.11)

Etape 5 **:** Calculer l'erreur dans les couches en utilisant l'expression (2.12)

Etape 6 **:** Calculer le gradient de l'erreur par rapport aux poids en utilisant l'expression (2.8)

Etape 7 **:** Ajuster les poids selon l'expression (2.7).

Etape 8 **:** Si la condition sur l'erreur ou sur le nombre d'itérations est atteinte, aller à l'étape 9, sinon aller à l'étape 2.

Etape 9: Fin.

Les exemples sont présentés d'une manière récursive, lorsque tous les exemples sont présentés, le test s'effectue sur l'erreur de sortie et les poids sont ajustés au fur et à mesure, jusqu'à ce que l'erreur de sortie se stabilise à une valeur acceptable.

Voici quelques remarques sur cet algorithme :

- L'algorithme de rétro propagation du gradient est une extension de l'algorithme de Widrow-Hoff. En effet, dans les deux cas, les poids sont mis à jour à chaque présentation d'exemple et donc on tend à minimiser l'erreur calculée pour chaque exemple et pas l'erreur globale.

- La méthode donne de bons résultats pratiques. Dans la plupart des cas, on rencontre peu de problèmes dus aux minima locaux, mais il y en a. Toutefois, il est moins performant que d'autres algorithmes de propagation d'erreur : il tend moins rapidement vers des poids plus ou moins optimaux.

- Il n'y a pas de condition d'arrêt pour le répéter. C'est à nous de fixer le critère. On peut par exemple répéter cela jusqu'à ce que l'erreur sur chaque exemple descende en dessous d'une certaine tolérance.

- Le choix de l'architecture initiale du réseau reste un problème difficile. Ce choix peut être fait par l'expérience. Des méthodes dites "auto-constructives" existent : il s'agit d'ajouter des neurones au cours de l'apprentissage pour que l'apprentissage se fasse bien. Mais ces méthodes rencontrent souvent le problème de "sur-apprentissage".

8. Conclusion

Nous avons vu dans ce chapitre la méthode la plus utilisée pour l'apprentissage des réseaux de neurones, la rétro-propagation qui repose sur la technique de calcul du gradient, appliquée à toute fonction dérivable. Et aussi du modèle neuronale inverse, de ses différentes architectures de commande et d’apprentissage.

Chapitre 3

Commande du MCC avec le reseau de neurones

1. Introduction

Dans ce chapitre, la modélisation du moteur à courant continu, ainsi que le réseau de neurones que seront présentés, suivis de l'apprentissage (6 cellules d'entrée, 7 neurones de couche cachée et 1 neurone de sortie) et des interprétations des résultats obtenus.

2. Moteur à Courant Continu (MCC)

C'est une machine électrique s'agissant d'un convertisseur électromécanique permettant la conversion bidirectionnelle d'énergie entre une installation électrique parcourue par un courant continu et un dispositif mécanique. Elle est aussi appelée dynamo.

- En fonctionnement moteur, l'énergie électrique est transformée en énergie mécanique.
- En fonctionnement générateur, l'énergie mécanique est transformée en énergie électrique.

Inventée par Zénobe Gramme (1826-1901), au début c'était un simple générateur de courant continu.

2.1. Constitution

Une machine électrique à courant continu est constituée :

- un **circuit magnétique** comportant une partie fixe, le stator, une partie tournant, le rotor et l'entrefer l'espace entre les deux parties.

- une source de champ magnétique nommée **l'inducteur** (le stator) crée par un bobinage ou des aimants permanents

- un circuit électrique **induit** (le rotor) subit les effets de l'interaction des deux champs magnétiques (celui de l'induit et celui du l'inducteur)

- le **collecteur** et les **balais** permettent d'accéder au circuit électrique rotorique

2.2. Schéma équivalent du MCC à excitation indépendante

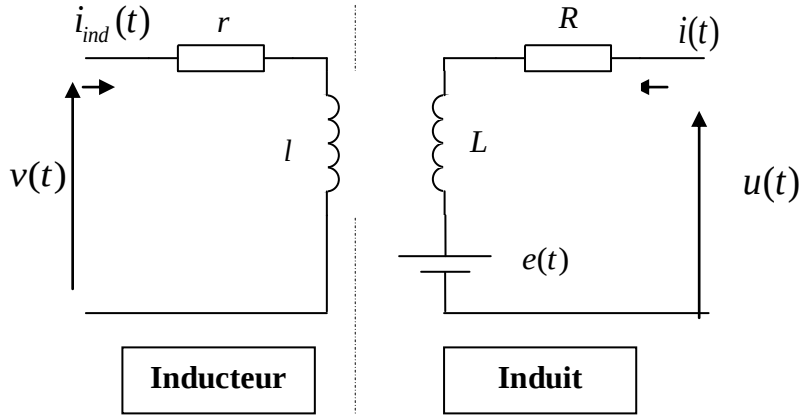

Figure 3.1. Schéma équivalent du moteur à courant continu

2.3. Mise en équation

Les équations électriques du moteur à courant continu sont données par :

$$v(t) = l\frac{di_{ind}(t)}{dt} + ri_{ind}(t)$$

$$u(t) = L\frac{di(t)}{dt} + Ri(t) + e(t)$$

$$c_{em}(t) = k\phi(t)i(t)$$

$$e(t) = k\phi(t)\Omega(t)$$

$$\phi(t) = mi_{ind}(t)$$

Les équations mécanique du moteur à courant continu sont données par :

$$c_{em}(t) - c_r(t) = J\frac{d\Omega(t)}{dt} + f\Omega(t)$$

$$\Omega(t) = \frac{d\theta(t)}{dt}$$

Où :

$u(t)$: La tension de l'inducteur

$v(t)$: La tension de l'induit

$c_{em}(t)$: Le couple électromagnétique

$e(t)$: La force contre-électromotrice

$\phi(t)$: Le flux électromagnétique

$\Omega(t)$: La vitesse de rotation du rotor

$\theta(t)$: La position du rotor

L : L'inductance du rotor

R : La résistance du rotor

l : L'inductance du stator

$i(t)$: Le courant du rotor

r : La résistance du stator

k : Constante relative au Moteur

m : Pente de la courbe de $\phi(t)$

J : Le moment d'inertie du moteur

f : la force de frottement dans le moteur

2.4. Commande par induit ($i_{ind}(t) = Canst \Rightarrow \phi(t) = Canst, k\phi = k'$)

Reprenons les équations de $u(t)$, $c_{em}(t)$ et $e(t)$:

$$u(t) = L\frac{di(t)}{dt} + Ri(t) + e(t)$$

$$c_{em}(t) = k'i(t)$$

$$e(t) = k'\Omega(t)$$

$$c_{em}(t) = J\frac{d\Omega(t)}{dt} + f\Omega(t)$$

Des équations du couple on obtient l'équation du courant en fonction de la vitesse de rotation qui est :

$$i(t) = \frac{J}{k'}\frac{d\Omega(t)}{dt} + \frac{f}{k'}\Omega(t)$$

En remplaçant la valeur de la force électromotrice et du courant dans l'équation de u(t) en obtient l'équation différentielle du deuxième ordre reliant l'entré u(t) et la sorite Ω(t) :

$$u(t) = a\frac{d^2\Omega(t)}{dt^2} + b\frac{d\Omega(t)}{dt} + c\Omega(t) \qquad (3.1)$$

Avec :

$$a = \frac{LJ}{k'}$$
$$b = \frac{Lf}{k'} + \frac{RJ}{k'}$$
$$c = \frac{Rf}{k'} + k'$$

2.5. Fonction du transfert du MCC en continu

En appliquant la transformée de Laplace à l'équation on obtient :

$$U(s) = as^2\Omega(s) + bs\Omega(s) + c\Omega(s)$$

Et la fonction de transfert du MCC sera :

$$H(s) = \frac{\Omega(s)}{U(s)} = \frac{G_0}{\frac{a}{c}s^2 + \frac{b}{c}s + 1} \qquad (3.2)$$

Où :

$$G_0 = \frac{1}{c},$$

2.6. Equation aux différences du MCC

Il en existe différente méthodes pour obtenir l'équation aux différences du moteur, parmi elles : déclaration de la fonction de transfert en continu avec la commande « tf » dans MATLAB puis la discrétiser en utilisant « c2d » tout en fournissant la période d'échantillonnage ainsi que la méthode ('tustin, zho, matched…),

3. Modèle du système à commander

Pour réaliser la commande inverse neuronale on prend comme système la machine à courant continu LSK 1122 L 04 dont les caractéristiques sont :

$R = 1,2\ Ohm$, $L = 0.18H$, $K = 0.284N.m/A$, $f = 6.10^{-3} Nm/rad/s$, $J = 24.10^{-4} kg.m^2$.

Le réseau de neurones a 6 cellules d'entrées, 7 neurones dans la couche cachée et un neurone en sortie.

3.1.Fonction du transfert

Avec les valeurs numériques, la fonction de transfert du MCC est telle que :

$$H(s) = \frac{\Omega(s)}{U(s)} = \frac{3.23}{0.003s^2 + 0.045s + 1}$$

Afin d'obtenir l'équation aux différences du MCC, on procède comme suit :

- création de la fonction de transfert dans l'environnement MATLAB comme suit :

$$>> sys = tf(3.23, [0.003\ 0.045\ 1])$$

- Discrétisation de cette fonction de transfert avec un pas de 10 ms :

$$>> sys\_d = c2d(sys, 0.01, 'matched')$$

Où : '*matched*' est une méthode de discrétisation incluse dans MATLAB.

On obtient la fonction de transfert discrète:

$$\frac{\Omega(z)}{U(z)} = \frac{1.65\ z + 1.65}{z^2 + 0.02126z + 0.0005531}$$

Pour avoir « z^{-1} » comme variable on utilise la syntaxe suivante :

$$sys\_d = tf([1.65\ 1.65],[1\ 0.02126\ 0.0005531], 0.01, 'variable', 'z^-1')$$

Résultats : $$\frac{\Omega(z)}{U(z)} = \frac{1.65 + 1.65z^{-}1}{1 + 0.2126z^{-}1 + 0.0005531z^{-}2}$$

A partir de cette fonction de transfert discrète on obtient l'équation aux différences du moteur MCC :

$$\Omega(k) = 21.26\ \ 10^{-3}\Omega(k-1) + 55.31\ \ 10^{-5}\Omega(k-2) + 1.65u(k) + 1.65u(k-1)$$

4. Présentation du réseau de neurones (6-7-1)

Nous avons choisi un réseau de neurones à 6 cellule d'entrée une couche cachée à 7 neurones et un neurone de sortie, comme le montre la figure suivante :

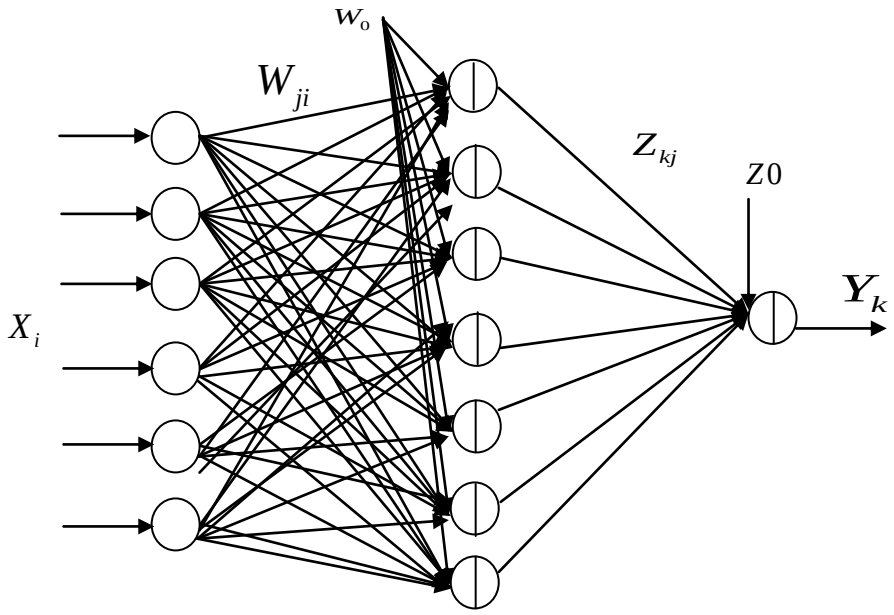

Figure 3.2. Réseau de neurones multicouche (MLP).

Soit :

$$a_j^{(1)} = \sum_{i=1}^{6} W_{ji} X_i$$

$$X_j^{(1)} = f(a_j^{(1)})$$

$$a_1^{(2)} = \sum_{1}^{7} W_{1j} X_j$$

$$y_1 = f(a_1^{(2)})$$

Tel que :

$a_j^{(1)}$: Contenu du neurone j de la couche cachée.

$a_1^{(2)}$: Contenu du neurone 1 de la couche de sortie

f : Fonction sigmoïde d'activation des neurones de la couche cachée et de sortie

4.1. Adaptation des poids du réseau

L'adaptation des poids se fait par la méthode du gradient basée sur les formules itératives suivantes :

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}$$

$$z_{1j}(k+1) = z_{1j}(k) + \Delta z_{1j}$$

Où :

$$\Delta w_{ij} = -\eta_1 \frac{\partial E}{\partial w_{ji}} \quad , \quad \Delta z_{1j} = -\eta_2 \frac{\partial E}{\partial z_{1j}},$$

Calcul de $\frac{\partial E}{\partial z_{1j}}$ et $\frac{\partial E}{\partial w_{ji}}$:

- $\frac{\partial E}{\partial z_{1j}} = \frac{\partial E}{\partial y_1} \times \frac{\partial y_1}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial z_{1j}}$

$$E = \frac{1}{2}\sum_{k=1}^{N}(y_k^d - y_k)^2$$

$$\frac{\partial E}{\partial y_1} = -(y_k^d - y_k)\ , \quad \frac{\partial y_1}{\partial a_1^{(2)}} = f'(a_j^{(1)})\ , \quad \frac{\partial a_1^{(2)}}{\partial z_{1j}} = x_i^{(1)}$$

$$\frac{\partial E}{\partial z_{1j}} = -(y^d - y\ ) \times f'(a_j^{(1)}) \times x_i^{(1)}$$

- $\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j^{(1)}} \times \frac{\partial x_j^{(1)}}{\partial a_j^{(1)}} \times \frac{\partial a_j^{(1)}}{\partial wji}$

$$\frac{\partial E}{\partial x_j^{(1)}} = \frac{\partial E}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial x_j^{(1)}} = -(y^d - y\ ) \times f'(a_j^{(1)}) \times z_{1j}$$

$$\frac{\partial x_j^{(1)}}{\partial a_j^{(1)}} = f'(a_j^{(1)})$$

$$\frac{\partial a_j^{(1)}}{\partial wji} = x_i$$

$$\frac{\partial E}{\partial w_{ji}} = -(y^d - y\ ) \times f'(a_j^{(1)}) \times z_{1j} \times f'(a_j^{(1)}) \times x_i$$

4.2. Architecture d'apprentissage

Nous avons utilisé l'architecture générale d'apprentissage illustré dans la figure suivante :

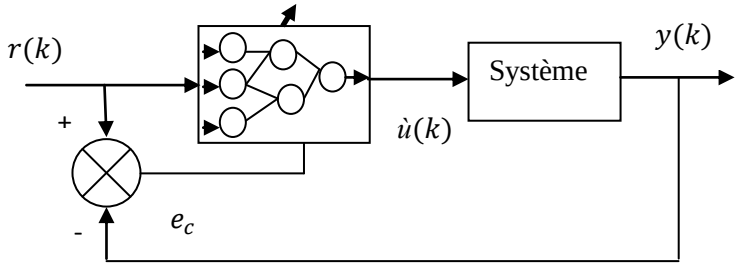

Figure 3.3. Architecture direct d'apprentissage.

5. Simulation de la SBPA et la réponse du MCC

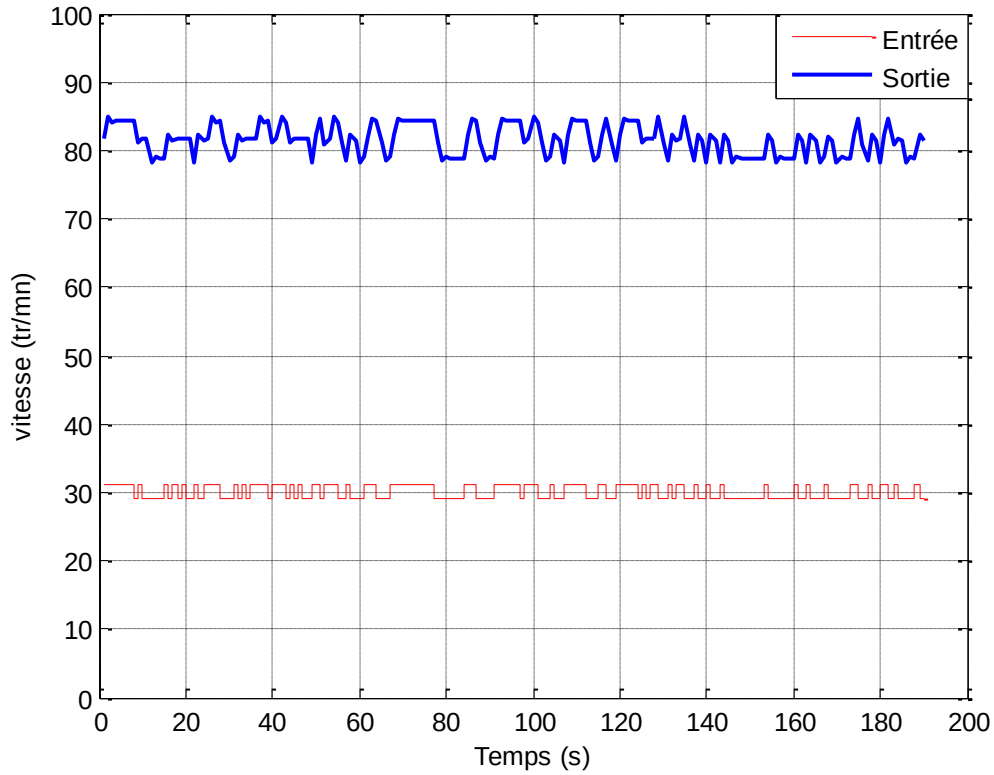

Figure 3.4. L'entrée SBPA et la réponse du moteur.

6. Simulation de l'apprentissage (eta=0,8)

Voilà en ce qui suit les figures montrant l'évolution des poids de connexions du réseau de neurones :

- **Evolution des poids et biais de la couche cachée :**

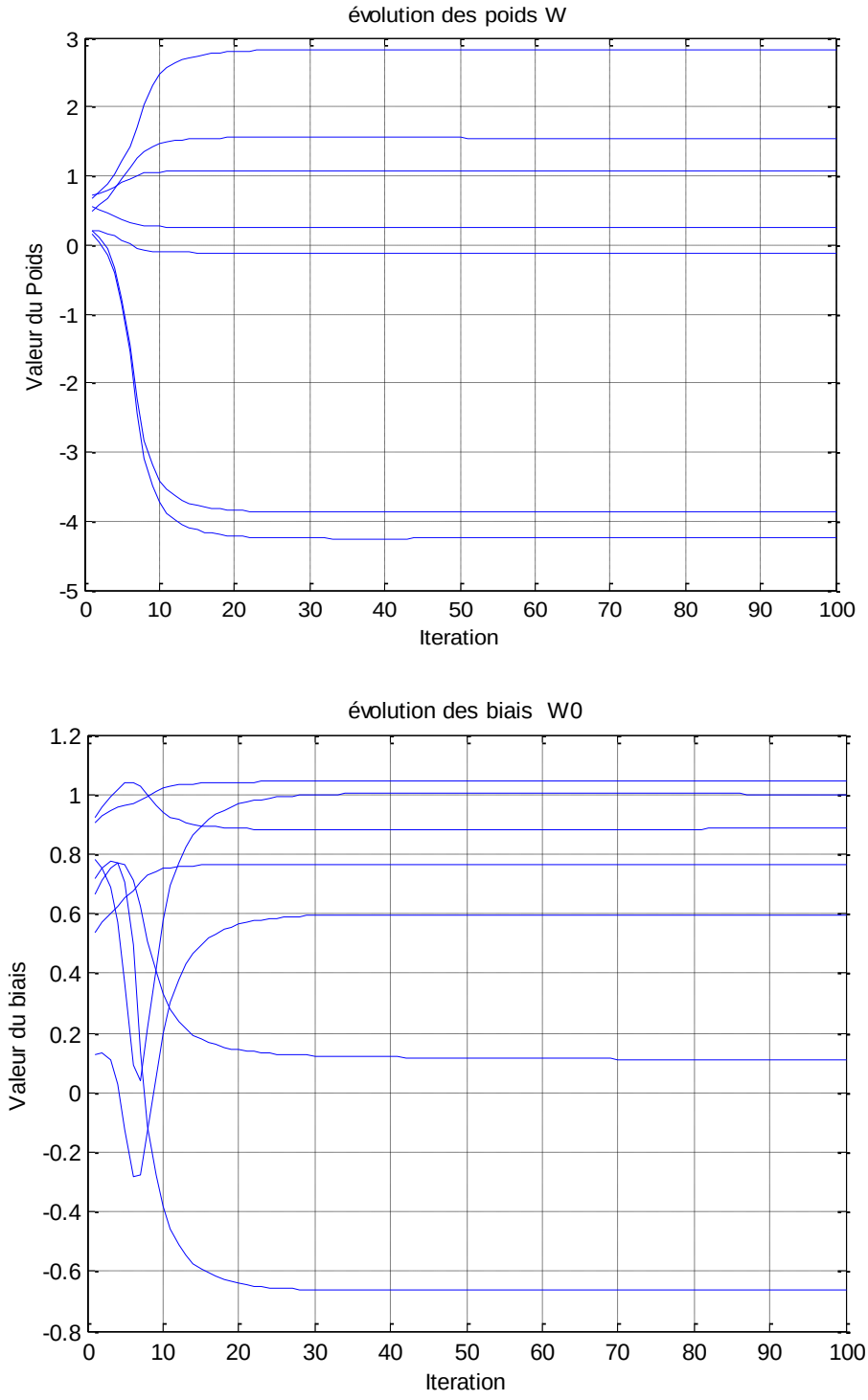

Figure 3.5. Evolution des poids et biais de la couche cachée.

- **Evolution des poids et biais de la couche de sortie :**

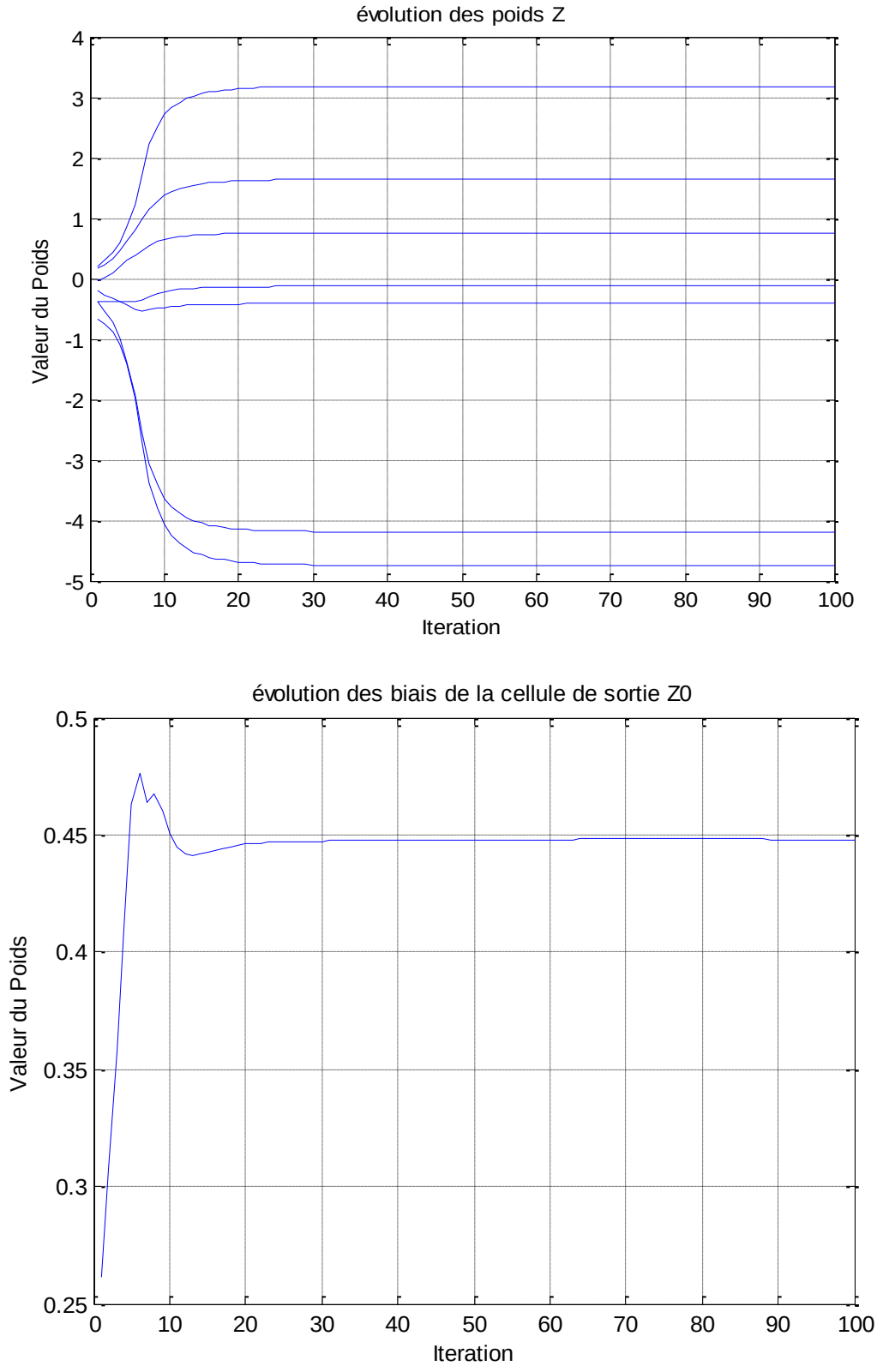

Figure 3.6. Evolution des pois et biais de la couche de sortie.

- **Interprétation des résultats**

Toutes les figures présentées ci-dessus montrent que tout les poids de connexion ainsi que leurs biais se stabilisent à des valeurs constantes et cela après un nombre d'itération qui dépasse les 50.

7. Architecture de commande

Nous avons utilisé l'architecture directe pour commander le système en boucle ouverte, cette architecture est illustrée dans la figure suivante :

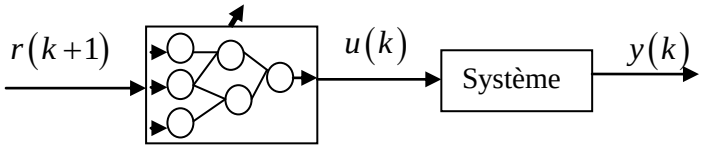

Figure 3.7. Architecture directe de commande.

8. Simulation de la commande avec le modèle inverse neuronal

8.1. Simulation sans perturbation (changement de consigne)

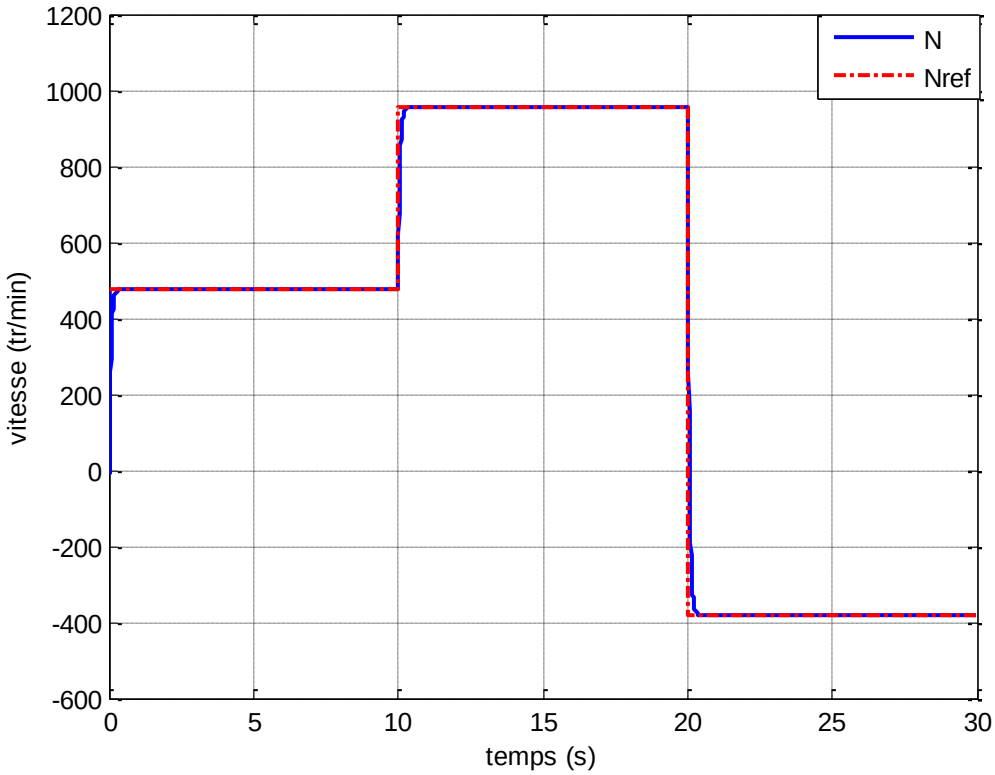

Figure 3.8. Vitesse désirée et la vitesse du moteur.

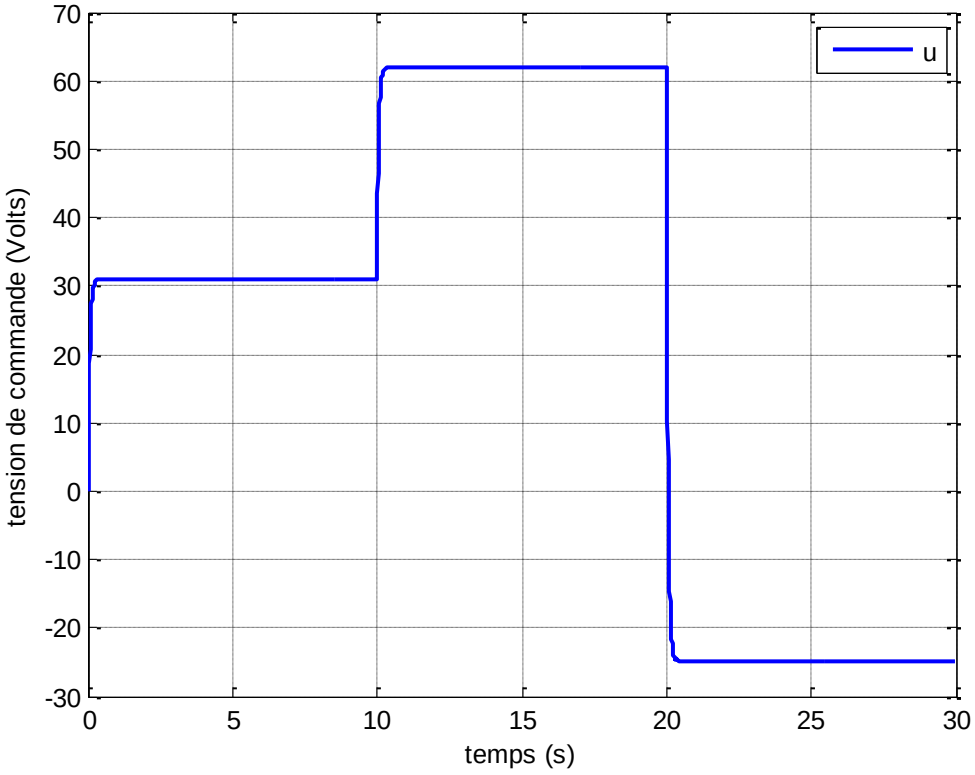

Figure 3.9. La commande appliquée au moteur.

8.2. Simulation avec perturbation (moteur en charge)

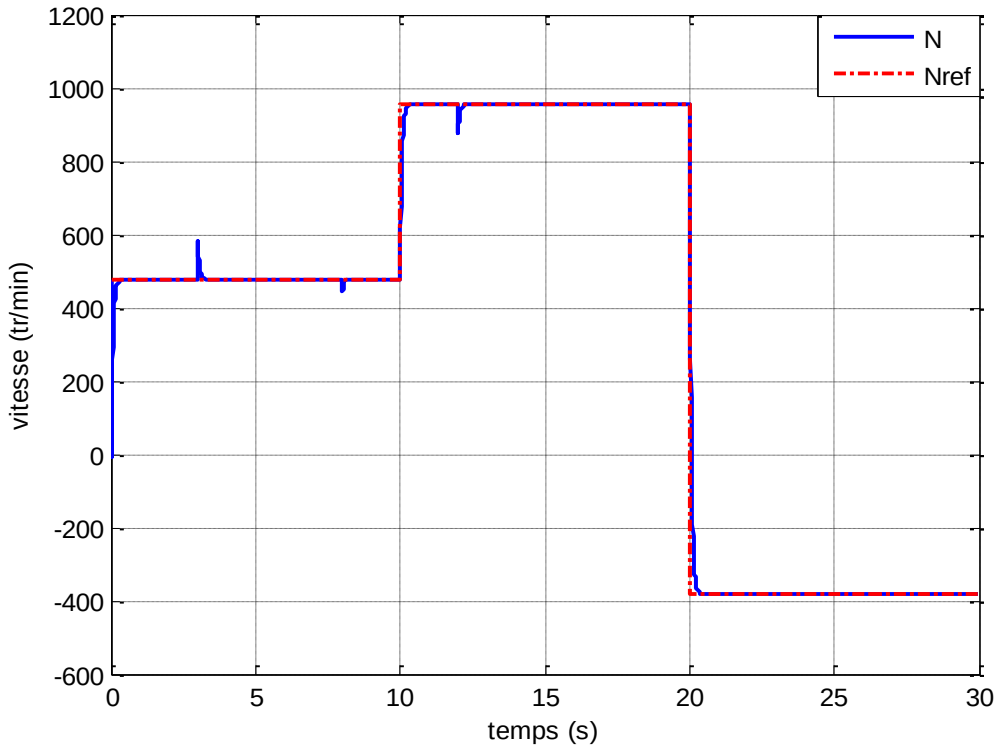

Figure 3.10. Vitesse désirée et la vitesse du moteur en charge.

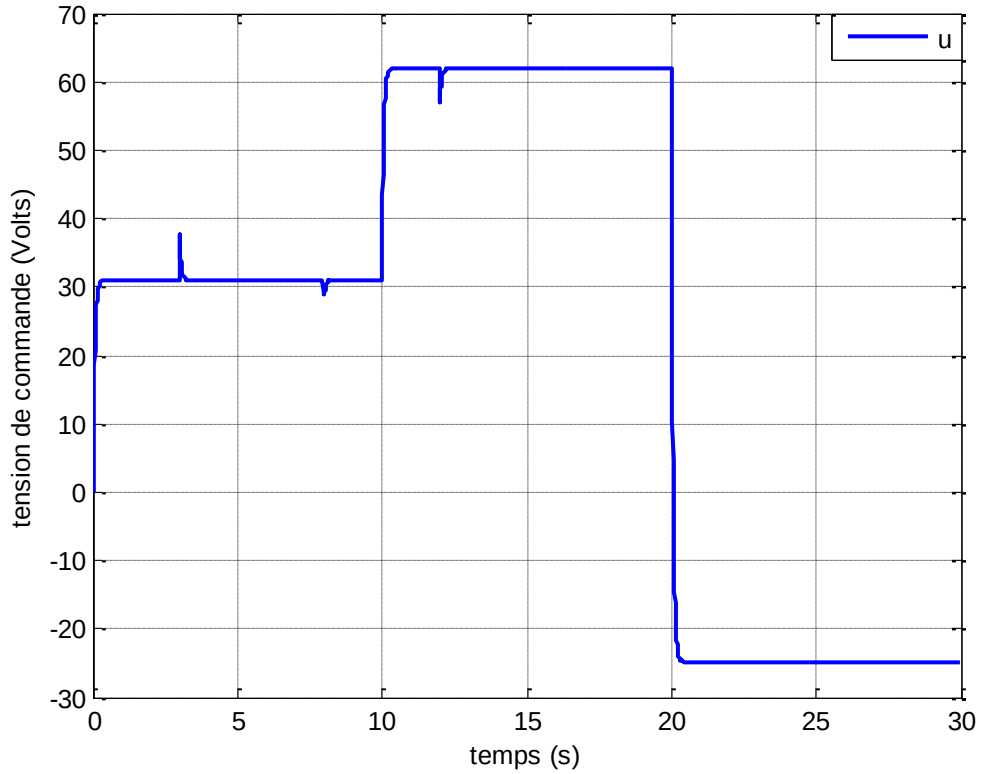

Figure 3.11. La commande appliquée au moteur en charge.

8.3. Simulation avec perturbation (variation paramétrique)

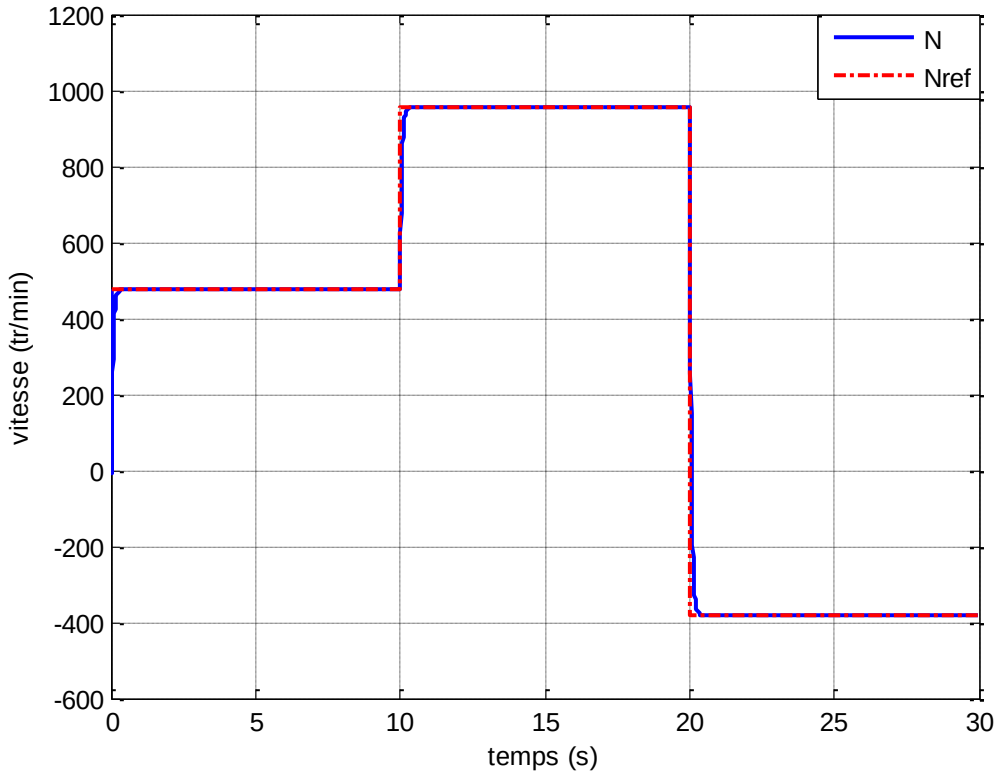

Figure 3.12. Vitesse désirée et la vitesse du moteur avec variation paramétrique.

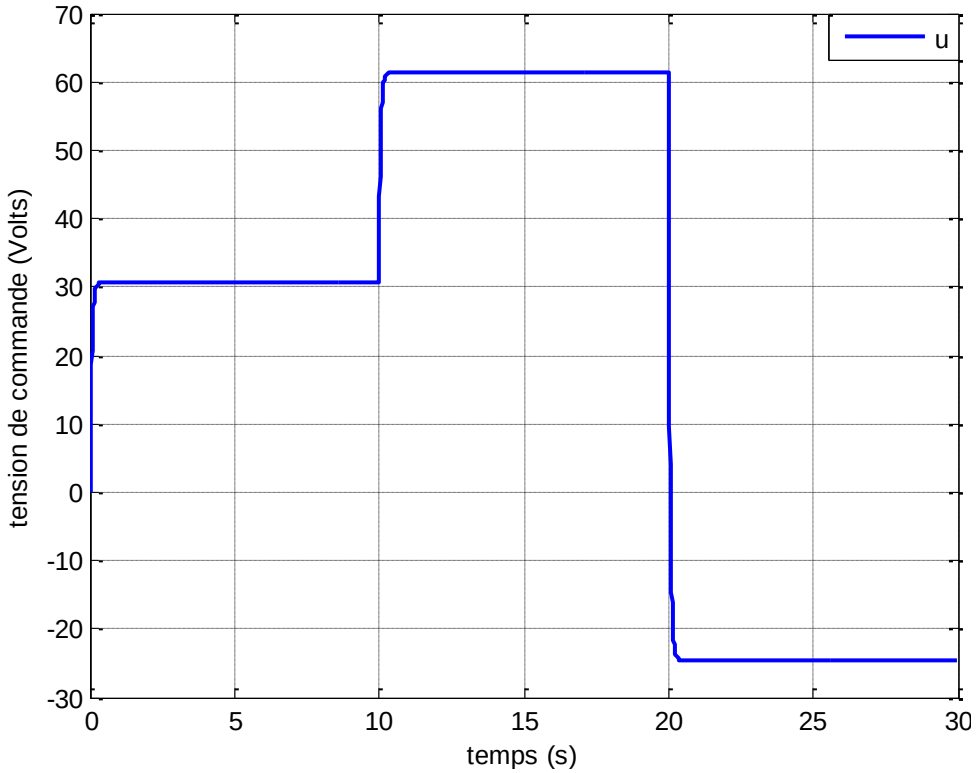

Figure 3.13. La commande appliquée au moteur.

- **Interprétation des résultats :**

La **Figure 3.8** montre que la sortie suit la consigne avec un très petit temps de réponse et se confond avec elle ce qui sous entend qu'il n'y a plus d'écart statique (écart statique nul), et sans dépassement.

Pour tester la robustesse de la commande avec le modèle inverse neuronal il a été injecté deux perturbations une à la commande l'autre à la sortie (à 3s, à 8s, 12s), et comme vous voyez à la **Figure 3.10**, la perturbation a été fortement atténuer voir même l'annuler au bout d'un petit moment, à cette effet là nous constatons que ce réseau de neurone est robuste.

On constate que le réseau de neurones a pu réaliser la poursuite de la vitesse de référence avec une tension de commande admissible de plus la commande inverse neuronale rejette les perturbations extérieures (comme le couple de charge)

La commande inverse neuronal est robuste aux imprécisions paramétriques, cela ce justifie par la poursuite effective en présence d'une variation paramétrique (50%) sur le système

9. Conclusion

Le travail mené en ce chapitre a mis en évidence les capacités impressionnantes de la commande inverse neuronale dans la commande des processus.

Conclusion générale

CONCLUSION GENERALE

Les réseaux de neurones semblent être l'outil qui pourra remplacer les régulateurs classiques vu leur capacité d'apprentissage et leurs simplicité de réalisation.

Dans ce travail, il a été vu que la commande inverse neuronale appliquée au moteur électrique à courant continu était une commande qui pouvait garantir de bonnes performances, que ce soit au niveau de la précision, du dépassement, ou de la stabilité ou aussi la robustesse aux perturbations extérieures.

Dans un premier temps, nous avons introduit différentes notions théoriques concernant les réseaux de neurones tel que l'architecture, l'activation et l'apprentissage. Chaque réseau est caractérisé par un certain nombre de paramètres (fonction d'activation, connexions des différents neurones du réseau,…etc.), en tenant compte de cela, nous aboutissons au réseau de neurones multicouches. De manière générale les réseaux de neurones présentent un moyen efficace pour résoudre des problèmes pour lesquels, les méthodes classiques ont montré leurs limites, que ce soit en identification ou en commande, et les résultats obtenus sont appréciables.

Dans la seconde partie, on s'est intéressé à la présentation de l'algorithme d'apprentissage le plus utilisé qui est l'algorithme de rétro propagation de l'erreur ainsi que la présentation de différents types d'architectures et d'apprentissages.

La troisième partie du travail a été consacrée à la modélisation du moteur à courant continu qui a établi un modèle discret sous forme d'une équation aux différences, et la création d'un réseau de neurones (6-7-1) jouant le rôle de modèle inverse. Cette partie se termine par des simulations en situation normale et perturbée. Les résultats de simulation ont montré l'efficacité de la commande en termes de poursuite des références et rejet des perturbations.

Comme perspective, il serait intéressant d'enrichir le travail en utilisant d'autres systèmes d'application (électriques, robotiques,…etc.), ce qui servirait de référence aux futurs travaux concernant la commande neuronale, en général.

BIBLIOGRAPHIE

L.Personnaz et I.Rivals,Réseaux de Neurones Formel pour la modélisation, la commande el la classification, CNRS Edition, Paris 2003.

J. Hérault & C. Jutten, Hermès ,Réseaux Neuronaux et Traitement du Signal.

[G. Cybenko,] ‘’Approximation by superposition of a sigmoidal function’’,
Math. In: Control Signals System 2nd ed, 1989, pp.303-314.

Cours de Mr MELLAH R. Maitre de conférence classe A à l’université Mouloud Mammeri de Tizi ouzou .

M.S. Boucherit, “Sur l’application de l’automatique moderne dans la commande des machines électriques”, Thèse de Doctorat d’état en G-électrique, ENP, Alger, 1995.

E. Davalo, P. Naim “Des Réseaux de Neurones”, *Edition Eyrolles*, 1993.

J.G. Taylor, “The promise of neural networks”, *Springer-Verlag*, New York, 1983.

Dr Grappa ‘Apprentissage Automatique : ‘Les réseaux de neurones’
Support de cours-Université LILLE 2005

WWW.Umoncton.ca/science/informatique
‘Réseaux de neurones artificiels’
Neural.net/