وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR- ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR ANNABA

جامعة باجي مختار - عنابـــة

Faculté des Sciences de l'Ingénieur

Année 05/06

Département d'Electronique

THESE

Présentée en vue de l'obtention du diplôme de **DOCTORAT**

CONTRIBUTION A LA MODELISATION ET LA COMMANDE DES PROCESSUS INDUSTRIELS PAR RESEAUX DE NEURONES

Option

Automatique

Par

A**b**delkrim **MOUSSAOUI**

DIRECTEUR DE THESE : Hadj Ahmed ABBASSI **Professeur** **U. ANNABA**

DEVANT LE JURY

PRESIDENT :	**Mouldi**	**BEDDA**	**Professeur**	**U. ANNABA**
EXAMINATEURS :	**Hicham**	**TEBBIKH**	**Professeur**	**U. GUELMA**
	Abdelhani	**BOUKROUCHE**	**M.C.**	**U. GUELMA**

Remerciements

Je tiens à remercier profondément Monsieur Hadj Ahmed ABBASSI, Professeur au département d'Electronique de l'Université Badji Mokhtar-Annaba, qui a assuré la direction scientifique de mes travaux, pour ses critiques et ses conseils.

Je tiens à exprimer ma vive gratitude à Monsieur M. BEDDA, Professeur au département d'Electronique d'Electronique de l'Université Badji-Mokhtar de Annaba, pour l'honneur qu'il me fait en acceptant la présidence du jury de ma thèse.

J'adresse mes plus vifs remerciements à Monsieur Hicham TEBBIKH, Professeur au département d'Electronique de l'Université 08 Mai 1945–Guelma et Monsieur Abdelhani BOUKROUCHE, Maître de Conférence au département d'Electronique de l'Université 08 Mai 1945-guelma, qui ont accepté de faire partie de mon jury de thèse.

Enfin, j'exprime ma reconnaissance à Monsieur Salah Bouhouche, chercheur au CSC/ex.DRA, El-Hadjar pour son encouragement et ses précieux conseils, sans oublier de remercier mes ex-collègues du Laminoir à chaud de Mittal-Steel Algérie ainsi que mes collègues des départements d'Electronique, d'Electrotechnique et de Mécanique de l'Université de Guelma, pour leur soutien constant et amical qu'ils m'ont apporté.

INTRODUCTION GENERALE **1**

Introduction 1

Organisation de la thèse 3

CHAPITRE I : Modélisation de processus **5**

I.1. Introduction. 5

I.2. Définition d'un processus et d'un modèle 5

I.2.1. Processus 5

I.2.2. Modèles 6

I.3. Classification des modèles 6

I.3.1. Les modèles de simulation ou simulateurs 6

I.3.2. Les modèles de prédiction ou prédicteurs 7

I.4. Notion de modèle de connaissance et modèle « boite noire » 7

I.5. Modèle hypothèse et forme prédicteur. 9

I.5.1 Modèle hypothèse. 9

I.5.2 Forme prédicteur théorique et système d'apprentissage 11

I.5.3 Forme prédicteur théorique associée à un modèle hypothèse. 12

I.5.3.1. Le modèle hypothèse est déterministe 13

I.5.3.2. Le modèle hypothèse est NARMAX. 14

I.5.3.3. Cas particulier : Modèle hypothèse NARX. 15

I.5.3.4. Le modèle hypothèse est NBSX 15

I.6. Conception de modèles NARMAX 16

I.7. Conclusion 16

CHAPITRE II : Estimation des paramètres d'un modèle **17**

II.1. Introduction. 17

II.2. Position du problème et notations. 18

II.3. Les algorithmes de minimisation de la fonction de coût. 18

II .3.1 Méthode des moindres carrés ordinaires. 19

II .3.2 Principe des algorithmes de gradient. 20

II .3.3 La méthode du gradient simple. 21
II .3.3.1 Présentation de la méthode 21
II .3.3.2 Techniques de réglage du pas 22
II .3.4 Les méthodes de gradient du second ordre 22
II .3.4.1 L'algorithme de BFGS 23
II .3.4.2 L'algorithme de Levenberg-Marquardt 25
II.4. Conclusion 28

CHAPITRE III : Les réseaux de neurones artificiels, Historique et bref état de l'art 29

III.1. Introduction 29
III.2. Historique 30
III.2.1. Les débuts 30
III.2.2. Les premiers succès 30
III.2.3. L'ombre 31
III.2.4. Le renouveau 31
III.2.5. La levée des limitations 32
III.2.6. La situation actuelle 33
III.3. Les neurones formels 33
III.3.1. Différents types de neurones 34
III.4. Les réseaux de neurones formels 36
III.4.1. Les réseaux de neurones non bouclés (statiques) 36
III.4.1.1. Perceptron multicouches (PMC) 36
III.4.2. Les réseaux de neurones bouclés (dynamiques) 38
III.4.3 Propriétés Les réseaux de neurones formels 39
III.4.3.1. Le parallélisme 39
III.4.3.2. La capacité d'adaptation 40
III.4.3.3. La facilité de construction 40
III.4.3.4. Propriété fondamentale : L'Approximation universelle parcimonieuse 40
III.4.3.4.a. l'approximation universelle 40

III.4.3.4.b. La parcimonie 41
III.5. Réseaux de neurones et régressions non linéaires 42
III.6. Apprentissage d'un réseau de neurones 44
III.6.1. Algorithme de Rétro-propagation (RP) 45
III.6.2. Variantes de l'algorithme de RP 49
III.6.2.1. Choix de la fonction coût 49
III.6.2.2. Introduction d'un terme de moment 50
III.6.2.3. Taux d'apprentissage adaptatifs 50
III.6.2.4. Autres procédures d'optimisation 52
III.7. Etude du pouvoir de généralisation des réseaux de neurones 53
III.7.1. Contrôle de la complexité 54
III.7.2. Méthodes d'élagage (Pruning) 55
III.7.2.1. Principe de la méthode OBD 56
III.7.2.2. Algorithme d'élagage par ODB 58
III.8. Choix des séquences d'apprentissage 59
III.8.1. Séquence des entrées de commande 59
III.8.1.a. Contraintes sur les entrées de commande 59
III.8.1.b. Fréquence d'échantillonnage 60
III.8.2. Séquences d'apprentissage et estimation de la performance 60
III.8.3. Le sur-ajustement 62
III.8.4. Problème des minima locaux 63
III.9. Conclusion 63

CHAPITRE IV : Commande des processus par réseaux de neurones 65
IV.1. Introduction 65
IV.2. Les étapes de la conception d'un organe de commande 65
IV.2.1. Choix d'un modèle du processus 65
IV.2.2. Estimation des paramètres du modèle (identification) 65
IV.2.3. Conception de l'organe de commande 66
IV.2.4. Estimation des paramètres du correcteur 66

IV.3. Conditions de mise en œuvre de la commande par réseaux de neurones 67
IV.3.1. Stabilité et sécurité 68
IV.3.2. Acceptation et utilisation 69
IV.4. Etude de la commande des processus par réseaux de neurones 70
IV.4.1. Commande par modèle inverse 70
IV.4.1.1. Apprentissage indirect 71
IV.4.1.2. Apprentissage général 71
IV.4.1.3. Apprentissage spécialisé 72
IV.4.2. Commande par identification neuronale directe 73
IV.4.3. Commande avec apprentissage spécialisé 74
IV.4.4. Commande par anticipation (feed-forward) 74
IV.4.5. Commande Hessienne 75
IV.4.5.1. Choix du coefficient de pondération ρ 79
IV.4.6. Variante adaptative de la loi commande Hessienne 80
IV.5. Exemples de commande de processus par RNA 80
IV.5.1. Exemple de commande d'un processus linéaire par modèle inverse 81
IV.5.2. Exemple de commande d'un processus non linéaire par modèle inverse 89
IV.5.2.1. Comportement du processus en boucle ouverte 89
IV.5.2.2. Synthèse du contrôleur neuronal 91
IV.6. Conclusion 96

CHAPITRE V : Application à la commande d'un laminoir à chaud 98
V.1. Introduction 98
V.2. Définition du laminage 98
V.3. Différents Types de laminages (Généralités) 99
V.4. Laminage à chaud des produits plats 101
V.4.1 Description du processus de laminage à chaud 101
V.4.2. Modèle physique de laminage à chaud 103
V.4.2.1. Elaboration du modèle de contrôle d'épaisseur (HAGC) 104
V.5. Etude de possibilité de l'implémentation des stratégies neuronales 108

V.6. Synthèse d'une stratégie de commande neuronale « en ligne » 109
V.6.1. Acquisition des données du processus 110
V.6.1.1. Modélisation des perturbations extérieures 110
V.6.2. Apprentissage hors ligne du modèle de prédiction 112
V.6.3. Elaboration de la loi de commande adaptative 114
V.7. Conclusion 117

Conclusion générale 118

Bibliographie 121

Annexe 128

Introduction

Au cours des dernières années, l'une des évolutions les plus marquantes des réseaux de neurones formels a été, pour la communauté scientifique, l'abandon de la métaphore biologique au profit de fondements théoriques solides dans le domaine des statistiques : on sait à présent que la propriété fondamentale des réseaux de neurones est l'approximation universelle parcimonieuse (Hornik et al., 1994).

De plus, le développement d'algorithmes performants pour l'apprentissage de ces réseaux leur a ouvert de nouvelles perspectives d'utilisation. En particulier, les réseaux de neurones se sont avérés particulièrement adaptés, dans le domaine de l'Automatique, comme éléments de systèmes de commande de processus dynamiques non linéaires.

En effet, avec des propriétés mathématiques bien établies et des algorithmes d'apprentissage très performants, les réseaux de neurones surpassent, lorsqu'ils sont convenablement mis en œuvre, les techniques classiques de modélisation non linéaire.

Néanmoins, pour la modélisation de processus, il faut déterminer les performances optimales réalisables par un modèle, compte tenu notamment de perturbations aléatoires non mesurables, afin de poser correctement le problème de l'apprentissage.

De même, pour la commande, il est nécessaire de caractériser les propriétés des systèmes de commande en matière de stabilité et de performance, indépendamment de la réalisation éventuelle du correcteur par un réseau de neurones.

Naturellement, la mise en œuvre d'un réseau neuronal est d'autant plus simple que la taille du réseau est plus petite.

Ainsi, la qualité des résultats et la facilité d'implantation matérielle orientent les études dans la même direction : l'optimisation des réseaux, en termes de nombre d'entrées comme de nombre de neurones, pour obtenir des modèles aussi parcimonieux que possible pour respecter les contraintes de l'implémentation en temps réel.

Notre travail comporte deux parties : une étude théorique consacrée à la modélisation et à la commande de processus non linéaires par réseaux de neurones, et une étude appliquée, consacrée à l'implémentation d'une loi de commande neuronale dans un environnement industriel.

Sur le plan théorique, nous présentons la modélisation et la commande de processus par réseaux de neurones dans un cadre aussi général que possible.

Nous illustrons notre démarche par une application industrielle, le contrôle d'épaisseur de bande dans un train finisseur d'un laminoir à chaud.

Les trains de laminage à chaud font parties des outils sidérurgiques les plus sophistiqués. Ils transforment un produit épais (*brame*) en une bobine de tôle mince. Pour la commercialisation du produit, les problèmes de tolérance dimensionnelle sont d'un intérêt capital. Des systèmes d'automatisation et de régulation de plus en plus efficaces ont été développés au cours des années afin d'obtenir une épaisseur la plus proche possible de l'épaisseur visée en sortie du laminoir (Hsu et al., 2000).

Actuellement, les performances demandées sont de plus en plus sévères. Ce qui nous a donc motivé à entreprendre une étude ayant pour but ultime la conception de nouveaux types de régulation d'un train finisseur de laminage à chaud multicages basés sur les réseaux de neurones artificiels (Moussaoui et al., 2003).

Notre choix sur l'application considérée est effectué dans un souci majeur d'améliorer la rentabilité de ce processus industriel complexe. En ce qui concerne l'évaluation de la rentabilité, il est pris en compte :

- L'amélioration de la qualité des produits ;
- L'augmentation de la production ;
- L'économie d'énergie ;
- La détection précoce des anomalies.

Organisation de la thèse

La thèse est constituée d'une introduction générale, de cinq chapitres et d'une conclusion générale:

Le chapitre I est consacré au rappel de quelques concepts fondamentaux pour la modélisation de processus. Nous montrons le lien qui existe entre, d'une part, les hypothèses a priori formulées sur le processus, qui conduisent à la définition d'un modèle hypothèse et de sa forme prédicteur théorique associée, et, d'autre part, le système d'apprentissage qu'il est nécessaire de mettre en œuvre pour l'identification d'un tel système. En particulier, nous mettons en évidence l'influence de la modélisation des perturbations dans le modèle hypothèse sur le choix du prédicteur du système d'apprentissage.

Dans le chapitre II, nous présentons les différentes méthodes pour l'estimation des paramètres des modèles non linéaires. Nous considérons des modèles « boite noire » non linéaire par rapport aux paramètres à ajuster. Il est montré que le recours à des algorithmes d'estimation (d'apprentissage) qui recherchent une solution suivant une procédure itérative est indispensable.

Dans le chapitre III, il est montré, à partir de principes simples, ce que sont réellement les réseaux de neurones. Après un aperçu historique sur les réseaux de neurones, et le rappel de quelques théorèmes relatifs à leurs capacités d'approximation, il est présenté un algorithme d'apprentissage très utilisé dans la communauté connexionniste : l'algorithme de rétro-propagation (RP). Il est rappelé dans ce chapitre ces principales variantes.

De plus, il est effectué une étude mettant en valeur le compromis entre les performances d'apprentissage et les performances de généralisation des réseaux de neurones en se basant sur l'optimisation de la structure des réseaux de neurones considérés.

Dans le chapitre IV on s'intéresse à la commande des processus non linéaires par réseaux de neurones. Après la description des différentes stratégies de commande basées sur les réseaux de neurones, il est étudié les conditions de leur mise en œuvre au sein des processus industriels, telle que le temps de traitement des algorithmes de commande en temps réel. Les performances de ces stratégies de commande sont vérifiées à travers des exemples illustratifs de processus dynamiques linéaire et non linéaire.

De même, et dans le souci de satisfaire les contraintes de l'implémentation en temps réel de la commande neuronale, il est développé dans ce chapitre une nouvelle stratégie de commande rapide basée sur le calcul « en ligne » de l'Hessien d'un critère de performance quadratique et l'utilisation du modèle neuronal de prédiction du processus.

Et afin d'éliminer l'effet des erreurs de modélisation, il est introduit dans la loi de commande Hessienne une étape d'adaptation des paramètres du modèle de prédiction neuronal. Il est proposé un ajustement en ligne des poids du modèle neuronal à chaque période d'échantillonnage selon un mode d'apprentissage incrémental.

Le chapitre V est consacré à l'application des stratégies neuronales à un processus sidérurgique d'élaboration à chaud de bandes en acier (laminage à chaud). Il est présenté dans ce chapitre une description assez détaillée du processus. De même, il est donné son modèle empirique utilisé pour la prédiction des efforts de laminage (forces de laminage).

Une étude par simulation, basée sur l'exploitation de données entrée/sortie de fonctionnement réel du laminoir à chaud, est effectuée afin de montrer les bonnes performances du schéma de commande proposé.

Enfin, la conclusion générale regroupe un ensemble de remarques relatives à la modélisation, l'estimation et la commande par réseaux de neurones des processus non linéaires (et particulièrement le processus industriel sidérurgique considéré).

I.1. Introduction

Dans le présent chapitre, nous rappelons les notions de processus et de modèle, ainsi que divers termes utilisés fréquemment dans le cadre de la modélisation. Nous montrons aussi le lien qui existe entre les hypothèses a priori formulées sur le processus, qui conduisent à la définition d'un modèle hypothèse et de sa forme prédicteur théorique associée.

I.2. Définition d'un processus et d'un modèle

I.2.1. Processus

Un processus est un objet soumis à des actions externes, à l'intérieur duquel des grandeurs interagissent, et sur lequel on peut faire des mesures. Il existe de nombreux types de processus, de natures très différentes : artificiels, naturels (écologiques, biologiques, etc.), financiers ou sociaux. Les processus que nous considérons dans ce travail sont des objets sur lesquels on peut agir par des actionneurs dans un but déterminé de pilotage de systèmes divers, de production ou de transformation de matière, etc.

Les grandeurs d'intérêt mesurées (variable pilotée, débit et qualité d'un produit, etc.) sont appelées les *sorties* du processus. Les variables externes qui lui sont imposées par un opérateur (par exemple un régulateur ou un opérateur humain) sont appelées *commandes* ; les autres variables externes sont des *perturbations*, et l'on distinguera les perturbations mesurables des perturbations non mesurables. Les commandes et les perturbations mesurables sont appelées les entrées du processus.

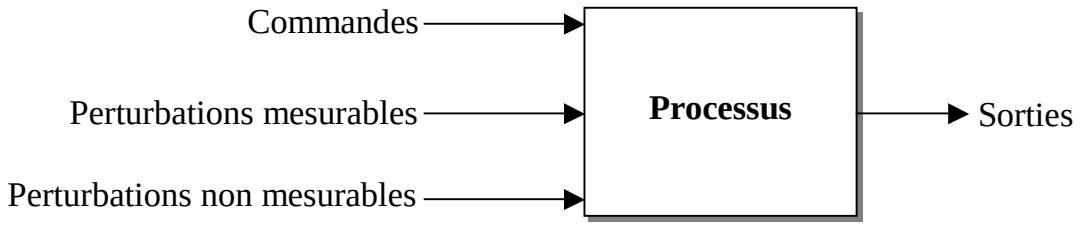


Figure I.1 : Schéma d'un processus

I.2.2. Modèles

Le but de toute modélisation de processus est de construire un modèle, en d'autres termes une représentation mathématique de son fonctionnement. Nous nous intéressons ici aux modèles mathématiques, qui représentent les relations entre les entrées et les sorties du processus par des équations. Si ces équations sont algébriques, le modèle est dit statique. Si ces équations sont des équations différentielles ou des équations aux différences récurrentes, le modèle est dit dynamique, respectivement à temps continu ou à temps discret.

Un modèle est caractérisé par son domaine de validité, en d'autres termes par le domaine de l'espace des entrées dans lequel l'accord entre les valeurs des sorties du processus calculées par le modèle, et leurs valeurs mesurées, est considéré comme satisfaisant compte tenu de l'utilisation que l'on fait du modèle.

I.3. Classification des modèles

Nous distinguerons deux classes de modèles :

I.3.1. Les modèles de simulation ou simulateurs

Un simulateur est un système qui possède un comportement dynamique analogue à celui du processus qui est destiné à fonctionner indépendamment de celui-ci. Les simulateurs sont utiles dans de nombreuses applications :

- Pour valider des hypothèses sur le processus que l'on étudie, pour extrapoler son comportement dans des domaines de fonctionnement où l'on ne dispose pas de résultats d'expériences ;
- Pour concevoir un système nouveau et appréhender à l'avance son comportement ou ses caractéristiques ; on parle parfois de « modèle de conception »;
- Pour tester de nouveaux dispositifs de commande ou de régulation qu'il serait trop coûteux, ou dangereux, de tester sur le processus lui-même (par exemple si le processus est une partie d'une centrale nucléaire, un avion, etc.), ou comme outil de formation de personnel lorsque la manipulation du processus par des personnes inexpérimentées est irréalisable (personnel de surveillance d'une raffinerie de pétrole, etc.). Ces simulateurs devront être aussi fidèles que

possible au processus, quel que soit le prix à payer en termes de complexité du modèle et de temps de calcul ;

- Ø Pour la synthèse d'une stratégie de commande du processus.

I.3.2. Les modèles de prédiction ou prédicteurs :

Un prédicteur fonctionne en parallèle avec le processus modélisé, et il prédit la valeur de sortie du processus à l'instant t+d, à partir des valeurs des entrées et des sorties du processus disponibles à l'instant t.

Dans le domaine de la conception de systèmes de commande et de régulation de processus, les modèles de prédiction sont utilisés aussi bien dans les phases de conception d'un modèle du processus et (d'apprentissage) d'un correcteur que pendant la phase d'utilisation. De tels systèmes tiennent une place très importante dans l'industrie.

La distinction entre « modèles de prédiction » et « simulateurs » est essentiellement liée à l'utilisation que l'on fait du modèle, et un même modèle peut, dans certains cas être utilisé soit comme prédicteur, soit comme simulateur.

I.4. Notion de modèle de connaissance et modèle « boite noire »

Lors de la modélisation d'un processus, deux démarches sont envisageables :

- Ø La première consiste à construire un modèle de connaissance. La conception des modèles de connaissance découle d'une analyse physique des phénomènes mis en jeu dans le processus ; lorsque cela est nécessaire, on décompose le processus étudié en éléments plus simples, pour lesquels on dispose déjà d'un modèle de connaissance éprouvé. Des données expérimentales sont ensuite utilisées, d'abord pour estimer les valeurs des paramètres du modèle, ensuite pour valider le modèle obtenu.

En particulier, la recherche scientifique a pour objet essentiel la construction de modèles de ce type, qui permettent non seulement de comprendre, mais aussi d'extrapoler le comportement d'un processus (réaction chimique, impact écologique d'un insecticide nouveau, etc.).

- La seconde démarche est de construire un modèle de type « boîte noire ». Plus précisément, on cherche une expression mathématique qui traduise de manière la plus fidèle possible le comportement « entrée-sortie » du processus dans un domaine de fonctionnement défini par l'utilisation ultérieure. Les paramètres n'ont généralement pas de signification physique. L'estimation numérique de ces paramètres repose essentiellement sur un ensemble d'observations expérimentales dont on dispose sur le processus; dans le domaine des réseaux de neurones par exemple, cet ensemble d'observations est appelé ensemble d'apprentissage.

Les modèles « boîte noire » sont en général économiques en temps de calcul. Leur validité est limitée à un domaine de fonctionnement déterminé par l'ensemble d'apprentissage, tandis que celle des modèles de connaissance est déterminée par l'exactitude des hypothèses et la pertinence des approximations faites lors de l'analyse physique des phénomènes et de leur mise en équation.

Dans le cadre de la conception de modèles « boîte noire » de processus non linéaires, les modèles neuronaux sont d'excellents candidats qui permettent généralement d'approcher le comportement dynamique du processus de façon satisfaisante: en effet, les réseaux de neurones sont des approximateurs universels de fonctions non linéaires (cf. Chapitre III).

Dans ce mémoire, nous nous intéressons à l'élaboration de modèles du type « boîte noire » de processus dynamiques non linéaires. Nous ne considérerons que le cas de processus stationnaires, c'est-à-dire tels que les lois qui régissent leur comportement n'évoluent pas au cours du temps.

Les modèles « boîtes noires » sont bien adaptés à la conception de modèles pour la commande de processus non linéaires, où l'on a souvent besoin de systèmes de structure relativement simple, afin de pouvoir effectuer de nombreux calculs, et ajuster si nécessaire les paramètres du modèle à des ensembles de données expérimentales nouvelles.

Les modèles de connaissance, bien que fondés sur une analyse approfondie du processus et ayant des domaines de validité généralement moins limités, sont souvent trop complexes pour effectuer des calculs de façon rapide.

De plus, leur conception est fortement liée au processus particulier que l'on cherche à modéliser, et aux connaissances dont on dispose sur la physique de celui-ci.

I.5. Modèle hypothèse et forme prédicteur

I.5.1 Modèle hypothèse

La conception d'un modèle de prédiction de comportement repose sur l'hypothèse selon laquelle, pour le processus que l'on considère, il existe une description analytique de la relation entre les entrées et les sorties du processus. Cette description est généralement inconnue, et on l'exprime de façon formelle. Cette représentation formelle, appelée modèle hypothèse , prend en considération les connaissances a priori et les hypothèses concernant le comportement du processus. Elle constitue la base de départ de toute procédure de modélisation.

L'élaboration d'un modèle hypothèse consiste à effectuer des hypothèses concernant la nature et les caractéristiques du processus : caractère statique ou dynamique du processus; présence de perturbations (leur nature, leur mode d'action); caractère linéaire ou non linéaire du processus. Il faut en général fixer ou estimer un certain nombre de caractéristiques numériques du modèle.

La forme la plus générale d'un modèle hypothèse *entrée-sortie* non linéaire, déterministe, stationnaire, sans retard, à temps discret, est donnée par l'expression :

$$y_p(t) = \varphi\left(y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u)\right) \qquad \text{(I.1)}$$

où $y_p(t)$ et $u(t)$ sont les sorties et les entrées du modèle du processus non linéaire à l'instant t.

Pour un modèle dynamique, il faut fixer ou estimer l'ordre n_y du modèle, ainsi que la valeur de la « mémoire » n_u sur la commande. Pour un modèle statique, la fonction $\varphi(.)$ ne dépend que de la variable de commande u aux instants $t-1$, $t-2$, …

Si l'on dispose de connaissances particulières, ou si l'on fait des hypothèses sur le comportement du processus, elles doivent être exprimées dans la formulation du modèle hypothèse : par exemple, si le comportement du processus est linéaire, la fonction $\varphi(.)$ est alors une somme pondérée des arguments $y_p(t-1)$, …, $y_p(t-n_y)$, $u(t-1)$, …, $u(t-n_u)$.

Supposons que des perturbations non mesurées agissent sur le processus. Ces perturbations peuvent être de deux types : les perturbations déterministes, et les perturbations de type « bruit ».

Les perturbations déterministes peuvent être modélisées par une entrée non commandable (sinusoïde, constante, etc.) qui survient ou se modifie à des instants aléatoires. Par exemple, l'encrassement d'un appareil au cours de son utilisation peut se modéliser par une rampe. Dans le cadre de la conception d'un système de commande, la présence de telles perturbations conduit à concevoir un système de commande adaptatif.

Les perturbations de type « bruit » sont des perturbations qui peuvent être modélisées par une séquence de variables aléatoires. Un bruit de mesure est souvent modélisé par un bruit additif sur la sortie du processus; un bruit d'état est un bruit additif sur les variables d'état (variables bouclées) du système.

Le choix du mode d'action du bruit, qui entre dans l'élaboration d'un modèle hypothèse, détermine le choix de l'algorithme d'estimation des paramètres du modèle.

Il est possible de représenter une large classe de systèmes non linéaires bruités à temps discret par le modèle entrée-sortie NARMAX (**N**on linéaire **A**uto **R**égressif à **M**oyenne **A**justée avec entrée e**X**ogène), (Leontaritis et Billings, 1985). Son expression la plus générale est :

$$y_p(t) = \varphi\left[y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u),w(t-1),\ldots,w(t-n_w)\right] + w(t) \qquad \text{(I.2)}$$

où n_w est la « mémoire » sur le bruit *w*.

Les différentes variables intervenant dans la forme NARMAX sont :

- la sortie mesurée y_p du processus, qui peut être vectorielle ;
- les perturbations non mesurables, modélisées à partir d'une séquence de variables aléatoires indépendantes d'espérance mathématique nulle *{w(t)} ;*
- le vecteur des entrées *u* : ses composantes sont les commandes, grandeurs sur lesquelles on peut agir pour influer sur le comportement du processus, et éventuellement des perturbations mesurables;
- les variables d'état du modèle qui sont, dans la représentation NARMAX, des valeurs passées de la sortie.

I.5.2 Forme prédicteur théorique et système d'apprentissage

Une fois le modèle hypothèse choisi, il faut déterminer ou estimer, à partir de séquences d'entrées *{u(t)}* et de sortie *{$y_p(t)$}* du processus, l'ensemble de ses caractéristiques inconnues. Pour cela, on définit, d'une part, une fonction de coût théorique, et, d'autre part, une forme prédicteur théorique associée au modèle hypothèse.

La *fonction de coût théorique* est une mesure de l'erreur de prédiction faite par un système de prédiction de la sortie du processus. On choisit généralement comme fonction de coût théorique la variance de l'erreur de prédiction.

La *forme prédicteur théorique* est un système qui permet de calculer, à l'instant *t*, une prédiction *y(t+d)* de la sortie $y_p(t+d)$ du processus telle que, si l'on suppose le processus parfaitement décrit par le modèle hypothèse, la fonction de coût théorique soit minimale.

Comme le modèle hypothèse, la forme prédicteur théorique est un système formel, qui s'exprime à l'aide des mêmes caractéristiques que le modèle hypothèse (en d'autres termes à l'aide de $\varphi(.)$, n_y , n_u *et* n_w pour un modèle NARMAX).

Une fois le modèle hypothèse postulé, la démarche conduisant à la forme prédicteur théorique est la suivante :

- on suppose que le modèle hypothèse est une description parfaite du processus ;
- on définit la sortie du processus comme variable aléatoire $Y_p(t)$, dont une réalisation est notée $y_p(t)$.

On peut alors exprimer la sortie du processus à l'instant $t+d$ sous la forme :

$$Y_p(t+d) = E\left[Y_p(t+d\,|\,t)\right] + \upsilon(t+d) \qquad \text{(I.3)}$$

où :

- $E[Y_p(t+d)|t]$ est l'espérance mathématique conditionnelle de $Y_p(t+d)$, lorsque l'on dispose de toutes les informations disponibles à l'instant t.
- $\nu(t+d)$ est la partie non prédictible de $Y_p(t+d)$ à l'instant *t*.

En supposant que le modèle hypothèse est exact, la meilleure prédiction que l'on puisse faire de $Y_p(t+d)$ à l'instant t est $E[Y_p(t+d)|t]$.

La sortie de la forme prédicteur théorique, que l'on exprime à l'aide de n_y, n_u, n_w, et $\varphi(.)$ est une expression analytique égale à chaque instant à $E[Y_p(t+d)|\ t]$.

Lorsque la structure du modèle et de sa forme prédicteur théorique (entrées et ordre du modèle, nombre de neurones et architecture d'un modèle neuronal, etc.) sont définies, il faut déterminer la fonction $\varphi(.)$, ou en trouver la meilleure approximation possible. Pour cela, on met en œuvre un *système d'apprentissage*, constitué d'un *prédicteur* et d'un *algorithme d'apprentissage*.

Ce prédicteur est un système dont la structure est identique à celle de la forme prédicteur théorique, et qui réalise une fonction paramétrée $\phi(.,\theta)$. Les arguments de $\varphi(.)$ qui sont des variables aléatoires sont remplacés dans l'expression de $\phi(.,\theta)$ par leurs *réalisations* lorsque ces réalisations sont mesurables, et par des *estimations* lorsqu'elles ne sont pas mesurables.

On définit alors une *fonction de coût empirique* à partir des écarts entre les sorties mesurées du processus et les valeurs calculées par le prédicteur, qui est une estimation de la fonction de coût théorique.

A l'aide de l'algorithme d'apprentissage, on calcule la valeur de θ qui minimise cette fonction de coût empirique. Si la structure du modèle n'est pas parfaitement définie, on considère alors un ensemble de modèles hypothèses, qui sont des cas particuliers du modèle hypothèse dont la structure est fixée, et l'on met en œuvre, pour chacun d'entre eux, un système d'apprentissage (d'estimation).

Nous reviendrons sur la mise en œuvre des systèmes d'apprentissage dans le chapitres II. Nous allons auparavant montrer, sur quelques exemples, comment déterminer la forme prédicteur théorique associée à un modèle hypothèse particulier.

I.5.3 Forme prédicteur théorique associée à un modèle hypothèse

Nous allons montrer, sur quelques exemples, le lien entre le modèle hypothèse et la forme prédicteur qui lui est associée. Nous ne considérons ici que des prédicteurs à un pas (d=1).

Dans le but d'illustrer l'influence du mode d'action du bruit sur le choix de la forme prédicteur, nous présentons les formes prédicteurs théoriques associées à un modèle hypothèse déterministe, à un modèle NARMAX, et à un modèle NBSX (**N**on linéaire à **B**ruit additif sur la **S**ortie et entrée e**X**ogène).

I.5.3.1. Le modèle hypothèse est déterministe

Le modèle hypothèse est de la forme (I.1) :

$$y_p(t) = \varphi\left[y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u)\right]$$

Le prédicteur calcule, à chaque instant *t-1*, la prédiction *y(t)* de la sortie $y_p(t)$ du processus. Le modèle hypothèse étant déterministe, la connaissance de $\varphi(.)$ et des valeurs passées de l'entrée et de la sortie permet de calculer exactement la valeur de $y_p(t+1)$. La forme prédicteur théorique doit donc effectivement calculer à chaque instant $y(t) = y_p(t)$. Le prédicteur suivant :

$$y(t) = \varphi\left[y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u)\right] \qquad \text{(I.4)}$$

est tel que la prédiction *y(t)* calculée à chaque instant est bien égale à la sortie $y_p(t)$ du processus, et l'erreur de prédiction est nulle. On a ainsi défini la forme prédicteur théorique associé au modèle déterministe (I.1). On remarque que cette expression correspond à un prédicteur non bouclé. Cependant, ce prédicteur est tel que, pour tout *t*, $y(t) = y_p(t)$; on peut aussi le mettre sous la forme :

$$y(t) = \varphi\left[y(t-1),\ldots,y(t-n_y),u(t-1),\ldots,u(t-n_u)\right] \qquad \text{(I.5)}$$

Ce dernier prédicteur (qui est bouclé) réalise alors également une prédiction parfaite s'il est correctement initialisé. Il existe donc, dans le cas d'un modèle déterministe, plusieurs représentations de la forme prédicteur théorique; par conséquent plusieurs structures de prédicteur peuvent être utilisées pour construire le prédicteur du système d'apprentissage.

Le choix de l'une ou l'autre de ces représentations est dicté, d'une part, par la complexité du système d'apprentissage à mettre en œuvre (l'apprentissage d'un prédicteur bouclé nécessite des algorithmes d'apprentissage plus complexes que celui d'un prédicteur non bouclé), d'autre part, par l'utilisation qui sera faite du modèle

obtenu à la fin de la procédure de modélisation (si le modèle doit être utilisé bouclé, il peut être préférable de faire son apprentissage avec un prédicteur bouclé).

I.5.3.2. Le modèle hypothèse est NARMAX

Le modèle hypothèse est de la forme (I.2) :

$$y_p(t) = \varphi\left(y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u),w(t-1),\ldots,w(t-n_w)\right) + w(t)$$

La séquence *{w(t)}*, réalisation de la séquence *{W(t)}* de variables aléatoires indépendantes d'espérance mathématique nulle et de variance σw^2 (*bruit pseudo-blanc*), est, par définition, imprédictible.

La forme prédicteur théorique du modèle *NARMAX* (I.2) est donc le prédicteur tel que $y_p(t) - y(t) = w(t)$, soit :

$$y(t) = \varphi\left(y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u),w(t-1),\ldots,w(t-n_w)\right) \quad (1.6)$$

Cependant, un tel prédicteur est irréalisable, puisque les valeurs $[w(t-1), .., w(t-n_w)]$ ne sont pas mesurables.

Il faut donc les estimer, et pour cela, on utilise les valeurs de l'erreur de prédiction $[e(t-1), e(t-2), \ldots, e(t-n_w)]$ comme estimations des valeurs passées du bruit. L'expression du prédicteur devient alors :

$$y(t) = \varphi\left(y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u),e(t-1),\ldots,e(t-n_w)\right) \quad (I.7)$$

Ce prédicteur est la forme prédicteur théorique associée au modèle *NARMAX* que l'on considère. En effet, si l'on suppose que les erreurs passées estiment parfaitement les valeurs passées du bruit, c'est-à-dire que l'on a, à l'instant *t-1*, la relation $[e(t-1), e(t-2),\ldots, e(t-n_w)] = [w(t-1), w(t-2), \ldots, w(t-n_w)]$, l'erreur de prédiction $e(t)$ à l'instant *t* est bien égale à $w(t)$, qui est la meilleure prédiction que l'on puisse faire. De plus, on retrouve, à l'instant t,

$$[e(t), e(t-1), \ldots, e(t-n_w+1)] = [w(t), w(t-1), \ldots, w(t-n_w+1)].$$

I.5.3.3. Cas particulier : Modèle hypothèse NARX

Le modèle hypothèse et la forme prédicteur théorique s'écrivent respectivement :

$$\begin{aligned} y_p(t) &= \varphi\left(y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u)\right) + w(t) \\ y(t) &= \varphi\left(y_p(t-1),\ldots,y_p(t-n_y),u(t-1),\ldots,u(t-n_u)\right) \end{aligned} \qquad \text{(I.8)}$$

Les modèles *NARX* conduisent donc à des prédicteurs non bouclés.

I.5.3.4. Le modèle hypothèse est NBSX

Le modèle hypothèse est de la forme:

$$\begin{aligned} y_p(t) &= \varphi\left(x_p(t-1),\ldots,x_p(t-n_y),u(t-1),\ldots,u(t-n_u)\right) \\ y_p &= x_p + w(t) \end{aligned} \qquad \text{(I.9)}$$

Le prédicteur associé s'écrit simplement :

$$y(t) = \varphi\left(y(t-1),\ldots,y(t-n_y),u(t-1),\ldots,u(t-n_u)\right) \qquad \text{(I.10)}$$

En effet, si à l'instant *t-1*, la relation [$y(t-1)$, …, $y(t-n_y)$] = [$x_p(t-1)$, …, $x_p(t-n_y)$] est vérifiée, la prédiction à l'instant *t* est alors égale à $x_p(t)$, et $e(t) = y_p(t) - y(t) = w(t)$.

On retrouve à l'instant t la relation [$y(t)$, …, $y(t-n_y+1)$] = [$x_p(t)$, …, $x_p(t-n_y+1)$]. A chaque instant, l'erreur de prédiction est donc bien égale à la valeur de la perturbation $w(t)$.

On constate, en comparant les formes prédicteurs obtenues pour les modèles *NARMAX, NARX et NBSX*, l'influence de la modélisation des perturbations sur la détermination de la forme prédicteur.

Si le modèle hypothèse est exact, la forme prédicteur théorique fournit une variance de l'erreur de prédiction minimale et égale à la variance du bruit (Ljung, 1987).

Ceci est illustré par Nerrand sur des exemples *NARX* et *NBSX* dans le cas où les prédicteurs sont des prédicteurs neuronaux (Nerrand, 1992).

I.6. Conception de modèles NARMAX

En général, la procédure de modélisation se décompose en quatre étapes :

- conception d'un ensemble de modèles hypothèses candidats;
- définition des formes prédicteurs théoriques associées aux modèles hypothèses;
- pour chaque modèle hypothèse candidat : définition et apprentissage du modèle prédictif, défini par la forme prédicteur théorique, à l'aide de séquences d'entrées-sorties du processus (séquences d'apprentissage);
- sélection du meilleur candidat.

Dans le cas de modèles *NARMAX*, nous venons de montrer que l'on peut facilement déduire la structure du prédicteur théorique de l'expression du modèle hypothèse. Cependant, les valeurs de n_y, n_u *et* n_w, ainsi que l'expression de $\varphi(.)$, sont généralement inconnues.

Lors de l'identification du processus, on doit donc trouver de bonnes valeurs de ces caractéristiques, et déterminer, dans une famille de fonctions $\phi(.,\theta)$ que l'on se donne (un réseau de neurones artificiels, par exemple), la fonction qui approche au mieux $\varphi(.)$.

I.7. Conclusion

Nous pouvons à présent définir précisément le cadre dans lequel se place notre travail :

Nous nous intéressons à la conception de modèles « boîte noire » de processus dynamiques non linéaires. Nos efforts porteront sur la conception du prédicteur et l'estimation de ses paramètres.

Les connaissances *a priori* nous conduisent à faire l'hypothèse que le processus peut être décrit par un modèle *NARMAX*. L'expression et les arguments de la fonction $\varphi(.)$ définissant ce modèle doivent être déterminées. Les processus que nous étudierons étant non linéaires, les modèles neuronaux sont des candidats naturels (Nerrand, 1992).

Nous disposons de données expérimentales, et l'on supposera que l'on peut effectuer autant d'expériences que nécessaire pour mener à bien la modélisation. Cette condition n'est évidemment pas toujours réalisable dans la réalité industrielle.

Cette procédure permet de s'approcher de la méthode de Newton plus rapidement que la méthode précédente. En revanche, étant donné que plusieurs ajustements de paramètres sont testés, elle nécessite un plus grand nombre d'inversions de matrice.

II.4. Conclusion

Nous avons présenté dans ce chapitre un aperçu sur les différents algorithmes d'estimation (d'apprentissage). Il est constaté que la difficulté essentielle lors de l'application de l'algorithme de BFGS réside dans le choix de la condition de passage du gradient simple à la méthode de BFGS. Ce problème ne se pose pas pour l'algorithme de Levenberg-Marquardt, mais le volume de calculs nécessaires à chaque itération de cet algorithme croît avec le nombre de paramètres.

De la même façon que dans le cas de l'algorithme de BFGS, une recherche unidimensionnelle doit être appliquée pour la recherche d'un pas de descente et ceci à chaque itération de l'algorithme. Une stratégie communément utilisée (Bishop, 1995), (Walter, 1994) consiste à appliquer la procédure suivante:

- Soit $r > 1$, (généralement égal à 10) un facteur d'échelle pour μ_k. Au début de l'algorithme, on initialise μ_0 à une grande valeur (Bishop propose 0.1).
- A l'étape *k* de l'algorithme :

- Calculer $J(\theta^k)$ avec μ_k déterminé à l'étape précédente
- **Si** $J(\theta^k) < J(\theta^{k-1})$ **, alors** accepter le changement de paramètres et diviser μ_k par *r*.
- **Sinon** récupérer θ^{k-1} et multiplier μ_k par *r*. Répéter cette dernière étape jusqu'à ce qu'une valeur de μ_k correspondant à une décroissance de *J* soit trouvée.

Cet exemple de procédure présente l'avantage de nécessiter peu d'inversions de matrice à chaque itération de l'algorithme. En revanche, le choix du pas initial possède une influence sur la vitesse de convergence de l'algorithme. Ces observations nous mènent à proposer la procédure suivante :

- Au début de l'algorithme, initialiser μ_0 à une valeur positive quelconque. En effet ce choix n'a pas d'influence sur le déroulement de l'algorithme.
- A l'étape *k* de l'algorithme :

1. Calculer $J(\theta^k)$ avec μ_k disponible (le dernier calculé).

2. Si $J(\theta^k) < J(\theta^{k-1})$, **alors** récupérer θ^{k-1}, diviser μ_k par *r* et aller à l'étape 1.

3. Sinon récupérer θ^{k-1} et multiplier μ_k par *r*. Répéter cette dernière étape jusqu'à ce qu'une valeur de μ_k correspondant à une décroissance de *J* soit trouvée.

*** Inversion indirecte :**

Le lemme d'inversion de matrices permet de calculer la matrice inverse suivant une loi récurrente (Friedland, 1987). En effet, soient *A*, *B*, *C* et *D* quatre matrices. On a la relation suivante :

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B[C^{-1} + DA^{-1}B]^{-1}DA^{-1} \qquad \text{(II.18)}$$

D'autre part, en posant $X^n = \frac{\partial y^n}{\partial \theta^k}$, l'approximation de la matrice *H* peut être calculée à partir de la loi de récurrence suivante :

$$\tilde{H}^n = \tilde{H}^{n-1} + X^n[X^n]^T \qquad \text{(II.19)}$$

avec : n=1,…..N

De ce fait, on a $\tilde{H} = \tilde{H}^N$. Si l'on applique le lemme d'inversion de matrices à la relation précédente en choisissant $A = \tilde{H}$, $B = X^n$, $C = I$ et $D = [X^n]^T$, on obtient la relation suivante :

$$[\tilde{H}^n]^{-1} = [\tilde{H}^{n-1}]^{-1} - \frac{[\tilde{H}^{n-1}]^{-1} X^n [X^n]^T [\tilde{H}^{n-1}]^{-1}}{1 + [X^n]^T [\tilde{H}^{n-1}]^{-1} X^n} \qquad \text{(II.20)}$$

En prenant, à la première étape ($n = 1$), $\tilde{H}^0 = \mu_k I$, on obtient, à l'étape *N* :

$$[\tilde{H}^N]^{-1} = [\tilde{H} + \mu_k I]^{-1} \qquad \text{(II.21)}$$

*** Inversion directe :**

Plusieurs méthodes d'inversion directes existent. Etant donné que l'algorithme est itératif et que la procédure de recherche du pas nécessite souvent plusieurs inversions de matrice, on a intérêt à utiliser une méthode économique en nombre de calculs surtout pour les applications en temps réel. Le fait que l'approximation du Hessien augmentée de μ_k reste une matrice symétrique définie positive nous permet d'utiliser la méthode de Cholesky (Gourdin et Boumahrat, 1991).

De ce fait, l'optimisation s'effectue en deux étapes : utilisation de la règle du gradient simple pour s'approcher d'un minimum, et de l'algorithme de BFGS pour l'atteindre. Le critère d'arrêt choisi, est alors le critère décrit au (*§ II.2.2)*.

II .3.4.2. L'algorithme de Levenberg-Marquardt

L'algorithme de Levenberg-Marquardt (Levenberg, 1944), (Marquardt, 1963) repose sur l'application de la formule de mise à jour des paramètres suivante :

$$\theta^k = \theta^{k-1} - \left[H(\theta^{k-1}) + \mu_k I \right]^{-1} \nabla J(\theta^{k-1}) \qquad \text{(II.15)}$$

où $H(\theta^{k-1})$ est le Hessien de la fonction de coût et μ_k est le pas. Pour de petites valeurs du pas, la méthode de Levenberg-Marquardt s'approche de celle de Newton. Inversement, pour de grandes valeurs de μ_k, l'algorithme Levenberg-Marquardt est équivalent à l'application de la règle du gradient simple avec un pas de ($1/\mu_k$).

La première question relative à cet algorithme est celle de l'inversion de la matrice $\left[H(\theta^{k-1}) + \mu_k I \right]$. L'expression exacte du Hessien de la fonction *J* est :

$$H(\theta^k) = \sum_{n=1}^{N} \left(\frac{\partial e^n}{\partial \theta^k} \right) \left(\frac{\partial e^n}{\partial \theta^k} \right)^T + \sum_{n=1}^{N} \frac{\partial^2 e^n}{\partial \theta^k \partial (\theta^k)^T} e^n \qquad \text{(II.16)}$$

avec : $e^n = y_p^n - y^n$.

Le second terme de l'expression étant proportionnel à l'erreur, il est donc permis de le négliger en première approximation, ce qui fournit une expression approchée :

$$\tilde{H}(\theta^k) = \sum_{n=1}^{N} \left(\frac{\partial e^n}{\partial \theta^k} \right) \left(\frac{\partial e^n}{\partial \theta^k} \right)^T = \sum_{n=1}^{N} \left(\frac{\partial y^n}{\partial \theta^k} \right) \left(\frac{\partial y^n}{\partial \theta^k} \right)^T \qquad \text{(II.17)}$$

Dans le cas d'un modèle linéaire par rapport aux paramètres, en d'autres termes, si y est une fonction linéaire de θ, le second terme de l'expression de *H* est nul est l'approximation devient exacte. Plusieurs techniques sont envisageables pour l'inversion de la matrice [$\tilde{H} + \mu_k I$].

L'approximation de l'inverse du Hessien est modifiée à chaque itération suivant la règle suivante :

$$M_k = M_{k-1} + \left(1 + \frac{\gamma_{k-1}^T M_{k-1} \gamma_{k-1}}{\delta_{k-1}^T \gamma_{k-1}}\right) \frac{\delta_{k-1}^T \delta_{k-1}}{\delta_{k-1}^T \gamma_{k-1}} - \frac{\delta_{k-1} \gamma_{k-1}^T M_{k-1} + M_{k-1} \gamma_{k-1} \delta_{k-1}^T}{\delta_{k-1}^T \gamma_{k-1}} \qquad \text{(II.13)}$$

avec : $\gamma_{k-1} = \nabla J(\theta^k) - \nabla J(\theta^{k-1})$ et $\delta_{k-1} = \theta^k - \theta^{k-1}$

Nous prenons pour valeur initiale de M la matrice identité. Si, à une itération, la matrice calculée n'est pas définie positive, elle est réinitialisée à la matrice identité.

Reste la question du choix du pas μ_k. A cet effet, une méthode économique en calculs est souvent recommandée dans la littérature : la technique de (Nash, 1980). Cette technique recherche un pas qui vérifie la condition de descente :

$$J(\theta^{k-1} + \mu_k d_k) \leq J(\theta^{k-1}) + m_1 \mu_k d_k^T \nabla J(\theta^{k-1}) \qquad \text{(II.14)}$$

où m_1 est un facteur choisi très inférieur à 1 (par exemple $m_1 = 10^{-3}$).

En pratique, la recherche du pas se fait de manière itérative. On initialise μ_k à une valeur positive arbitraire. On teste la condition (II.14). Si elle est vérifiée, on accepte l'ajustement des paramètres. Sinon, on multiplie le pas par un facteur inférieur à 1 (par exemple 0.2) et on teste à nouveau la condition de descente.

On répète cette procédure jusqu'à ce qu'une valeur satisfaisante du pas soit trouvée. Si au bout d'un certains nombre d'essais, le pas atteint une valeur très petite (de l'ordre de 10^{-16}, par exemple), on peut considérer alors qu'il n'est pas possible de trouver un pas satisfaisant.

Une méthode « quasi-newtonienne », n'est efficace que si elle est appliquée au voisinage d'un minimum.

D'autre part, la règle du gradient simple est efficace lorsqu'on est loin du minimum et sa convergence ralentit considérablement lorsque la norme du gradient diminue (en d'autres termes, lorsqu'on s'approche du minimum). Ces deux techniques sont donc complémentaires.

II .3.4.1. L'algorithme de BFGS

L'algorithme de BFGS, du nom de ses inventeurs : **B**royden, **F**letcher, **G**oldfarb et **S**hanno, (Minoux, 1983) fait partie des méthodes d'optimisation dites « quasi-newtoniennes ». Ces méthodes sont une généralisation de la méthode de Newton. La méthode de Newton consiste à l'application de la règle suivante :

$$\theta^{k} = \theta^{k-1} - \left[H(\theta^{k-1})\right]^{-1} \nabla J(\theta^{k-1}) \qquad \text{(II.10)}$$

où $H(\theta)$ est le Hessien de la fonction J calculé avec le vecteur des paramètres disponible à l'étape courante. La direction de descente est dans ce cas :

$$d_{k} = -\left[H(\theta^{k-1})\right]^{-1} \nabla J(\theta^{k-1}) \qquad \text{(II.11)}$$

Le pas μ_k est constant et égal à 1.

Pour que le déplacement soit dans le sens contraire du gradient, il est indispensable que la matrice du Hessien soit définie positive. Sous cette condition, et si la fonction de coût est quadratique par rapport aux paramètres, la méthode de Newton converge vers l'unique solution en une seule itération.

En général, la fonction de coût n'est généralement pas quadratique. Elle peut néanmoins l'être localement, à proximité d'un minimum de ses minima. Donc, la méthode de Newton ne peut converger en une seule itération. De plus, cette méthode nécessite l'inversion de la matrice du Hessien à chaque itération, ce qui conduit à des calculs lourds.

L'algorithme de BFGS, ainsi que l'algorithme de Levenberg-Marquardt présenté dans le paragraphe suivant, sont des méthodes quasi-newtoniennes qui permettent de pallier ces inconvénients.

L'algorithme de BFGS est une règle d'ajustement des paramètres ayant l'expression suivante :

$$\theta^{k} = \theta^{k-1} - \mu_{k} M_{k} \nabla J(\theta^{k-1}) \qquad \text{(II.12)}$$

où M_k est une approximation, calculée itérativement, de l'inverse de la matrice Hessienne.

En pratique, la méthode du gradient simple peut être efficace lorsque l'on est loin du minimum de *J*. Quand on s'en approche, la norme du gradient diminue et donc l'algorithme progresse plus lentement. A ce moment, on peut utiliser une méthode de gradient plus efficace.

Un réglage du pas de gradient μ_k est nécessaire : en effet, une petite valeur de ce paramètre ralentit la progression de l'algorithme ; en revanche une grande valeur aboutit généralement à un phénomène d'oscillation autour de la solution. Diverses heuristiques, plus ou moins efficaces, ont été proposées.

II .3.3.2 Techniques de réglage du pas

- **Technique du pas constant :** elle consiste à adopter un pas constant $\mu_k = \mu$ tout au long de l'algorithme. Elle est très simple mais peu efficace puisqu'elle ne prend pas en considération la décroissance de la norme du gradient.

- **Technique du pas asservi :** on peut asservir le pas à l'aide de la norme du gradient de sorte que le pas évolue en sens inverse de celle-ci. A chaque étape, le pas peut être calculé par :

$$\mu_k = \frac{\mu}{1 + \|\nabla J\|} \qquad \text{(II.9)}$$

où μ est un paramètre constant. Lors de l'utilisation de la technique du pas asservi, l'adoption de la valeur $\mu = 10^{-3}$ se révéle très souvent satisfaisante.

Le dénominateur est augmenté du nombre 1 afin d'éviter une instabilité numérique au moment de la division dans le cas où la norme du gradient devient très proche de zéro. Cette technique offre un bon compromis du point de vue de la simplicité et de l'efficacité.

II .3.4 Les méthodes de gradient du second ordre

Les méthodes que nous venons de décrire sont simples mais en général très inefficaces. On a donc systématiquement recours à l'utilisation de méthodes plus performantes (Battiti, 1992). Elles sont dites du second ordre parce qu'elles prennent en considération la dérivée seconde de la fonction de coût.

Le déroulement de ces algorithmes suit le schéma suivant (Oussar, 1998) :

A l'itération 0 : Initialiser le vecteur des paramètres à θ^0. Cette initialisation de θ peut avoir une grande influence sur l'issue de l'apprentissage

A la k^{ème} itération : Calculer la fonction de coût et la norme du gradient avec le vecteur des paramètres courant (obtenu à l'itération précédente).

Si $J(\theta^{k-1}) \leq J_{max}$ **ou** $\|\nabla J\| \leq J(\theta^{k-1}) \leq \varepsilon$ **ou** $k=k_{max}$

(où J_{max} est valeur maximale recherchée pour l'EQMA, ou pour l'EQMP si les performances sont évalués pendant l'apprentissage),

Alors arrêter l'algorithme ; le vecteur θ^{k-1} est une solution,

Sinon calculer θ^k à partir de θ^{k-1} de la formule de mise à jour des paramètres suivante:

$$\theta^k = \theta^{k-1} + \mu_k d_k$$

où μ_k est un scalaire positif appelé *pas du gradient* et d_k un vecteur calculé à partir du gradient, appelé *direction de descente*. Les différences entre les méthodes de gradient résident dans le choix de la direction de descente et le choix du pas.

II .3.3 La méthode du gradient simple

II .3.3.1 Présentation de la méthode

La méthode du gradient simple consiste à la mise en oeuvre de la formule de mise à jour des paramètres suivante :

$$\theta^k = \theta^{k-1} - \mu_k \nabla J(\theta^{k-1}) \qquad \text{(II.8)}$$

La direction de descente est donc simplement l'opposée de celle du gradient ; c'est en effet la direction suivant laquelle la fonction de coût diminue le plus rapidement.

dont la solution θ_{mc} donnée par :

$$\theta_{mc} = \left[X^T X \right]^{-1} \left[X^T Y_p \right] \qquad \text{(II.6)}$$

est l'estimation des moindres carrés du vecteur des paramètres θ_p.

Cette solution existe à condition que la matrice $X^T X$ soit inversible. Cette condition est généralement vérifiée lorsque N (le nombres d'exemples) est très grand devant N_i (le nombre d'entrées du modèle) (Goodwin, 1977).

La méthode des moindres carrés peut être utilisée plus généralement pour l'estimation des paramètres de tout modèle dont la sortie est linéaire par rapport aux paramètres à estimer.

Les sorties des modèles « boite noire » que nous utilisons dans ce mémoire ne sont pas linéaires par rapport aux paramètres à ajuster. Une résolution directe du problème comme dans le cas de la solution des moindres carrés n'est donc pas possible : on a donc recours à des algorithmes d'estimation (d'apprentissage) qui recherchent une solution suivant une procédure itérative.

Dans ce qui suit, nous allons présenter les algorithmes les plus utilisés pour la minimisation de la fonction de coût.

II .3.2. Principe des algorithmes de gradient

Les algorithmes d'apprentissage fondés sur l'évaluation du gradient de la fonction de coût $J(\theta)$ par rapport aux paramètres procèdent à la minimisation de manière itérative. $J(\theta)$ est une fonction scalaire à variable vectorielle (le vecteur θ des paramètres à ajuster). Son gradient est donc un vecteur défini par :

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial \theta} \\ \mathbf{j} \\ \frac{\partial J}{\partial \theta_M} \end{bmatrix} \qquad \text{(II.7)}$$

où M est le nombre de paramètres inconnus.

Le principe des algorithmes de gradient repose sur le fait qu'un minimum de la fonction de coût est atteint si sa dérivée (son gradient) est nul.

II .3.1 Méthode des moindres carrés ordinaires

Cette méthode est applicable pour « l'apprentissage » de modèles statiques ou de prédicteurs non bouclés dont la sortie est linéaire par rapport aux paramètres inconnus. Si cette sortie est linéaire par rapport aux entrées, le prédicteur associé a pour expression (Ljung, 1987) :

$$y(n) = \sum_{i=1}^{N} \theta_i x_i(n) \qquad \text{(II.2)}$$

Ce modèle de prédiction peut être mis sous forme d'une équation matricielle. En effet, on peut l'écrire $Y = X\,\theta$, avec :

$$Y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ \vdots \\ y(N) \end{bmatrix} \quad , \quad X = \begin{bmatrix} x_1(1) & x_2(1) & \dots & x_{N_i}(1) \\ \vdots & \cdot & \cdot & \vdots \\ \vdots & \cdot & \cdot & \vdots \\ x_1(N) & \dots & \dots & x_{N_i}(N) \end{bmatrix} \quad , \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_1 \\ \vdots \\ \vdots \\ \theta_{N_i} \end{bmatrix}$$

L'estimation des paramètres est fondée sur la minimisation de la fonction de coût des moindres carrés. En utilisant la notation matricielle présentée ci-dessus, l'expression de la fonction de coût $J(\theta)$ devient :

$$J(\theta) = \frac{1}{2}\left[Y_p^T Y_p - 2\theta^T Y_p + \theta^T X^T X \theta \right] \qquad \text{(II.3)}$$

La fonction de coût étant quadratique (par rapport au vecteur des paramètres à estimer), elle atteint son minimum pour la valeur du vecteur des paramètres annulant sa dérivée. Soit θ_{mc} cette valeur du vecteur des paramètres. Elle vérifie :

$$\left. \frac{\partial J}{\partial \theta} \right|_{\theta_{mc}} = 0 \qquad \text{(II.4)}$$

Cette dernière équation fournit une équation, appelée *équation normale* :

$$\left[X^T X \right] \theta_{\text{mc}} = \left[X^T Y_p \right] \qquad \text{(II.5)}$$

II.2. Position du problème et notations

Etant données les informations dont on dispose sur le processus (c'est-à-dire la séquence d'apprentissage) on détermine, dans une famille donnée de fonctions paramétrées $\psi(x,\theta)$ (où x est le vecteur regroupant toutes les entrées du modèle et θ le vecteur des paramètres inconnus de ψ) celle qui minimise une fonction de coût qui, le plus souvent, est la fonction de coût des moindres carrés.

Soit y_p^n la sortie du processus à l'instant n (dans le cas d'une modélisation dynamique), où la valeur mesurée pour le $n^{ème}$ exemple de l'ensemble d'apprentissage (dans le cas d'une modélisation statique). De même y^n est la sortie calculée par le modèle à l'instant n, ou pour le $n^{ème}$ exemple de l'ensemble d'apprentissage. On définit la fonction de coût des moindres carrés $J(\theta)$ par :

$$J(\theta) = \frac{1}{2}\sum_{n=1}^{N} \left| y_p^n - y^n \right|^2 \qquad \text{(II.1)}$$

où N est le nombre de mesures (taille de la séquence). $J(\theta)$ dépend du vecteur des paramètres, ainsi que de la séquence d'apprentissage. Pour alléger les notations, nous n'indiquerons pas explicitement cette dernière dépendance dans la suite.

On définit l'erreur quadratique moyenne d'apprentissage (*EQMA*) comme la moyenne de la fonction de coût calculée sur la séquence d'apprentissage. Elle est donnée par : $J(\theta) = \frac{2J(\theta)}{N}$.

Lors de son exploitation, le modèle reçoit des entrées différentes de celles de la séquence d'apprentissage. On peut estimer ses performances en calculant diverses fonctions ; celle que l'on utilise le plus fréquemment est l'erreur quadratique moyenne de performance (*EQMP)* dont la valeur est calculée sur une séquence différente de celle utilisée pour l'apprentissage.

II.3. Les algorithmes de minimisation de la fonction de coût

Dans le cas où le modèle est linéaire par rapport aux paramètres à ajuster, la minimisation de la fonction de coût, et donc l'estimation du vecteur des paramètres θ, peut se faire à l'aide de la méthode des moindres carrés, qui ramène le problème à la résolution d'un système d'équations linéaires.

II.1. Introduction

L'estimation des paramètres d'un modèle consiste à mettre en œuvre un système d'apprentissage constitué d'un prédicteur de la sortie du processus associé au modèle-hypothèse, et un algorithme d'apprentissage. Elle est effectuée en minimisant une fonction de coût définie à partir de l'écart entre les sorties mesurées du processus (séquences d'apprentissages) et les valeurs prédites. La qualité de cette estimation dépend du modèle-hypothèse choisi, de la richesse des séquences d'apprentissage et de l'efficacité de l'algorithme utilisé.

L'objectif de l'apprentissage (l'estimation) n'est pas ici d'annuler l'erreur de prédiction, puisque, si tel était le cas, le modèle serait capable de reproduire l'effet des perturbations non mesurables. Il s'agit plutôt d'obtenir une erreur de prédiction dont la variance est minimale, c'est-à-dire égale à celle du bruit. Si l'on peut obtenir un tel résultat, le réseau reproduit complètement le comportement déterministe du processus, bien que l'apprentissage ait été effectué en présence de perturbations. Des résultats théoriques prouvent que cet objectif est accessible, et de nombreux exemples montrent qu'il est effectivement atteint (Rivals et al., 1995).

Le principe de l'estimation des paramètres d'un modèle est montré dans la Figure II.1, où à partir des entrées de commandes à l'instant k-1, on peut prédire la sortie qu'aurait le processus à l'instant k si les perturbations $w(k)$, non mesurables, n'existaient pas.

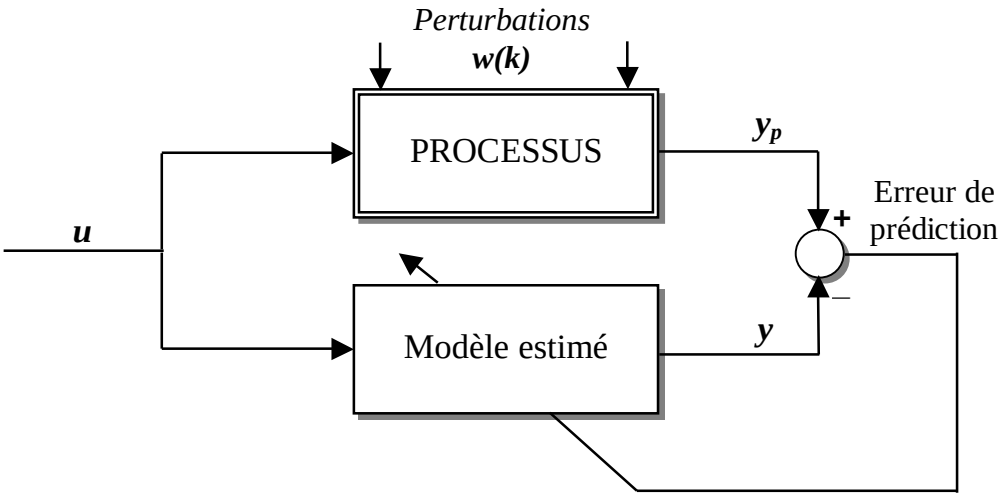


Figure II.1 : Schéma de principe de l'estimation

Les réseaux de neurones formels (artificiels) sont devenus en quelques années des outils précieux dans des domaines très divers de l'industrie. Néanmoins, ils n'ont pas encore atteint leur plein développement, pour des raisons plus psychologiques que techniques, liées aux connotations biologiques du terme, et au fait qu'ils sont considérés, à tort, comme des outils d'Intelligence Artificielle. Or l'intérêt des réseaux de neurones, dans le domaine des Sciences de l'Ingénieur, ne doit rien à la métaphore biologique : il est uniquement dû aux propriétés mathématiques spécifiques de ces réseaux. Nous expliquons dans le présent chapitre, à partir de principes simples, ce que sont réellement les réseaux de neurones, et nous délimitons leurs domaines d'excellence.

III.1 . Introduction

Lorsque apparaît une nouvelle technique, l'ingénieur se demande naturellement en quoi cette nouveauté peut lui être utile. Si elle est dotée d'un nom plus métaphorique que technique, (ce qui est évidemment le cas pour les réseaux de neurones), la réponse à cette question doit être particulièrement précise et motivée.

De plus, la mise en œuvre des réseaux de neurones est très simple ; la tentation peut être grande, d'appliquer cette technique de manière irréfléchie ou inadaptée, ce qui ne peut conduire qu'à des déceptions.

C'est pourquoi nous expliquerons ici les principes fondamentaux qui justifient l'intérêt pratique des réseaux de neurones, et nous situerons ces derniers dans la perspective des méthodes classiques de traitement statistique de données ; nous montrerons que la technique des réseaux de neurones formels doit être considérée comme une extension puissante de techniques bien connues des ingénieurs, telles que les régressions.

III.2. Historique

III.2.1. Les débuts

- 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 (Fausset, 1994).
- 1943 : J. Mc Culloch et W. Pitts laissent leurs noms à une modélisation du neurone biologique (un neurone au comportement binaire). Ceux 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, (McCulloch et Pitts, 1943).
- 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 (Hebb, 1949).

III.2.2. Les premiers succès

- 1957 : F. Rosenblatt développe le modèle du Perceptron. Il construit le premier neuro-ordinateur basé sur ce modèle et l'applique au domaine de la reconnaissance de formes. Notons qu'à cet époque les moyens à sa disposition sont limités et c'est une prouesse technologique que de réussir à faire fonctionner correctement cette machine plus de quelques minutes (Rossenblatt, 1957), (Rosenblatt , 1958).
- 1960 : B. Widrow, un automaticien, développe le modèle Adaline (Adaptative Linear Element). Dans sa structure, le modèle ressemble au Perceptron, cependant la loi d'apprentissage est différente (Widrow, 1960). Celle-ci est à l'origine de l'algorithme de rétropropagation de gradient très utilisé aujourd'hui avec les Perceptrons multicouches. Les réseaux de type Adaline restent utilisés de nos jours pour certaines applications particulières.

- 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 tous modèles 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'Intelligence Artificielle et les systèmes à bases de règles (Minsky et Papert, 1969).

III.2.3. L'ombre

- 1967 à 1982 : Toutes les recherches ne sont, bien sûr, pas interrompues. Elles se poursuivent, mais déguisées, sous le couvert de divers domaines comme : le traitement adaptatif du signal, la reconnaissance de formes, la modélisation en neurobiologie, etc. De grands noms travaillent durant cette période tels : S. Grosberg (Grosberg, 1976) et T. Kohonen (Wu, 1994).

III.2.4. Le renouveau

- 1982 : J. J. Hopfield est un physicien reconnu à qui l'on doit le renouveau d'intérêt pour les réseaux de neurones artificiels. A cela plusieurs raisons : Au travers d'un article court, clair et bien écrit, il présente une théorie du fonctionnement et des possibilités des réseaux de neurones (Hopfield, 1982). Il faut remarquer la présentation anticonformiste de son article. Alors que les auteurs s'acharnent jusqu'alors à proposer une structure et une loi d'apprentissage, puis à étudier les propriétés émergentes ; Hopfield fixe préalablement le comportement à atteindre pour son modèle et construit à partir de là, la structure et la loi d'apprentissage correspondant au résultat escompté. Ce modèle est aujourd'hui encore très utilisé pour des problèmes d'optimisation. D'autre part, entre les mains de ce physicien distingué, la théorie des réseaux de neurones devient respectable. Elle n'est plus l'apanage d'un certain nombre de psychologues et neuro-biologistes hors du coup.

Notons qu'à cette date, l'intelligence artificielle est l'objet d'une certaine désillusion, elle n'a pas répondue à toutes les attentes et s'est même heurtée à de sérieuses limitations. Aussi, bien que les limitations du Perceptron mise en avant par M. Minsky ne soient pas levées par le modèle d'Hopfield, les recherches sont relancées.

III.2.5. La levée des limitations

- 1985 : La Machine de Boltzmann est le premier modèle connu apte à traiter de manière satisfaisante les limitations recensées dans le cas du perceptron. Mais l'utilisation pratique s'avère difficile, la convergence de l'algorithme étant extrêmement longue (les temps de calcul sont considérables) (Ackley et al., 1985).
- 1986 : L'algorithme d'apprentissage de rétro-propagation de gradient apparaît (Rumelhart et al., 1986). C'est un algorithme d'apprentissage adapté aux réseaux de neurones multicouches (aussi appelés Perceptrons multicouches). Dès cette découverte, nous avons la possibilité de réaliser une fonction non linéaire d'entrée/sortie sur un réseau en décomposant cette fonction en une suite d'étapes linéairement séparables. De nos jours, les réseaux multicouches et la rétro-propagation du gradient reste le modèle le plus étudié au niveau des applications.

-1989 : Vingt ans après la publication de l'ouvrage où Minsky et Papert exposaient les limitations du perceptron, (Cybenko, 1989), (Funahashi, 1989) et (Hornik et al., 1989) établissent les réseaux de neurones comme une classe d'approximateurs universels. Il a été ainsi démontré qu'un perceptron multicouches 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 donné, le nombre de neurones optimal dans la couche cachée. En d'autres termes, ce résultat ne mène pas vers une technique de construction d'architecture.

-1994 : Hornik (Hornik et al., 1994) a démontré la parcimonie des réseaux de neurones à une couche cachée, ce qui a permet d'étendre le champs d'application des réseaux de neurones aux applications industrielles en temps réel (Rivals, 1995).

III.2.6. La situation actuelle

Les réseaux de neurones occupent de nos jours une place importante dans le domaine de la modélisation, de la commande et de la classification, ce qui justifie l'intérêt que le monde scientifique et industriel continuent à porter au « connexionnisme ».

III.3. Les neurones formels

Un « neurone formel » (ou simplement neurone) est une fonction algébrique non linéaire et bornée, dont la valeur dépend de paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelées « entrées » du neurone, et la valeur de la fonction est appelée sa « sortie ».

Un neurone est donc avant tout un opérateur mathématique, dont on peut calculer la valeur numérique par quelques lignes de logiciel. On a pris l'habitude de représenter graphiquement un neurone comme indiqué sur la Figure III.1.

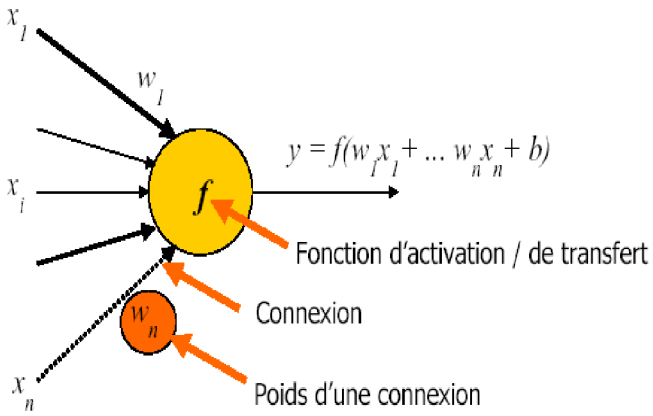


Figure III.1 : Neurone formel

où les x_i sont les variables (ou entrées) du neurone, les w_i sont des paramètres ajustables, y est la sortie du neurone et b est le *biais*.

Les neurones les plus fréquemment utilisés sont ceux pour lesquels la fonction d'activation f est une fonction non linéaire (généralement une tangente hyperbolique ou une sigmoïde) d'une combinaison linéaire des entrées.

III.3.1. Différents types de neurones

La *fonction d'activation* (de transfert) utilisée dans le modèle de McCulloch & Pitts est la fonction *échelon* (Figure III.2-a). Elle fait passer l'activation du neurone d'une valeur à une autre dès que l'entrée résultante dépasse un certain seuil. L'inconvénient de cette fonction est qu'elle n'est pas différentiable, ce qui pose un problème pour les algorithmes basés sur le gradient.

Pour remédier à cet inconvénient, on cherche à approximer cette fonction d'activation par une fonction « non linéaire » différentiable.

Deux fonctions de ce type sont particulièrement intéressantes et sont souvent utilisées : la fonction tangente hyperbolique (Figure III.2.b) définie par :

$$f(u) = \tanh(\beta u) = \frac{e^{\beta u} - e^{-\beta u}}{e^{\beta u} + e^{-\beta u}} \qquad \text{(III.1)}$$

et la fonction logistique (Figure III.2.c) dont l'expression est la suivante :

$$f_\beta(u) = \frac{1}{1+e^{-\beta u}} \qquad \text{(III.2)}$$

La fonction « tanh » est bornée entre -1 et +1 alors que la fonction logistique est bornée entre 0 et 1. Ces deux fonctions, appelées *fonctions sigmoïdes*, sont liées par la relation :

$$\tanh(\beta u) = 2 f_\beta(u) - 1 \qquad \text{(III.3)}$$

où le paramètre β est appelé le gain. Plus le gain est important, plus la saturation du neurone est rapide (dans notre travail, le paramètre β est pris égal à 1).

La fonction logistique (III.2) est appelée aussi *sigmoïde unipolaire.* Elle admet une variante appelée *sigmoïde bipolaire* (Figure III.2.d) ayant pour expression :

$$f_\beta(u) = \frac{1-e^{-\beta u}}{1+e^{-\beta u}} \qquad \text{(III.4)}$$

Les fonctions sigmoïdes ont la propriété d'être différentiables, ce qui est nécessaire pour les algorithmes basés sur le gradient. Une autre propriété intéressante est le fait que les fonctions dérivées peuvent s'exprimer facilement à l'aide des fonctions elles-mêmes, ce qui permet un gain significatif de temps de calcul (Billings et al., 1992).

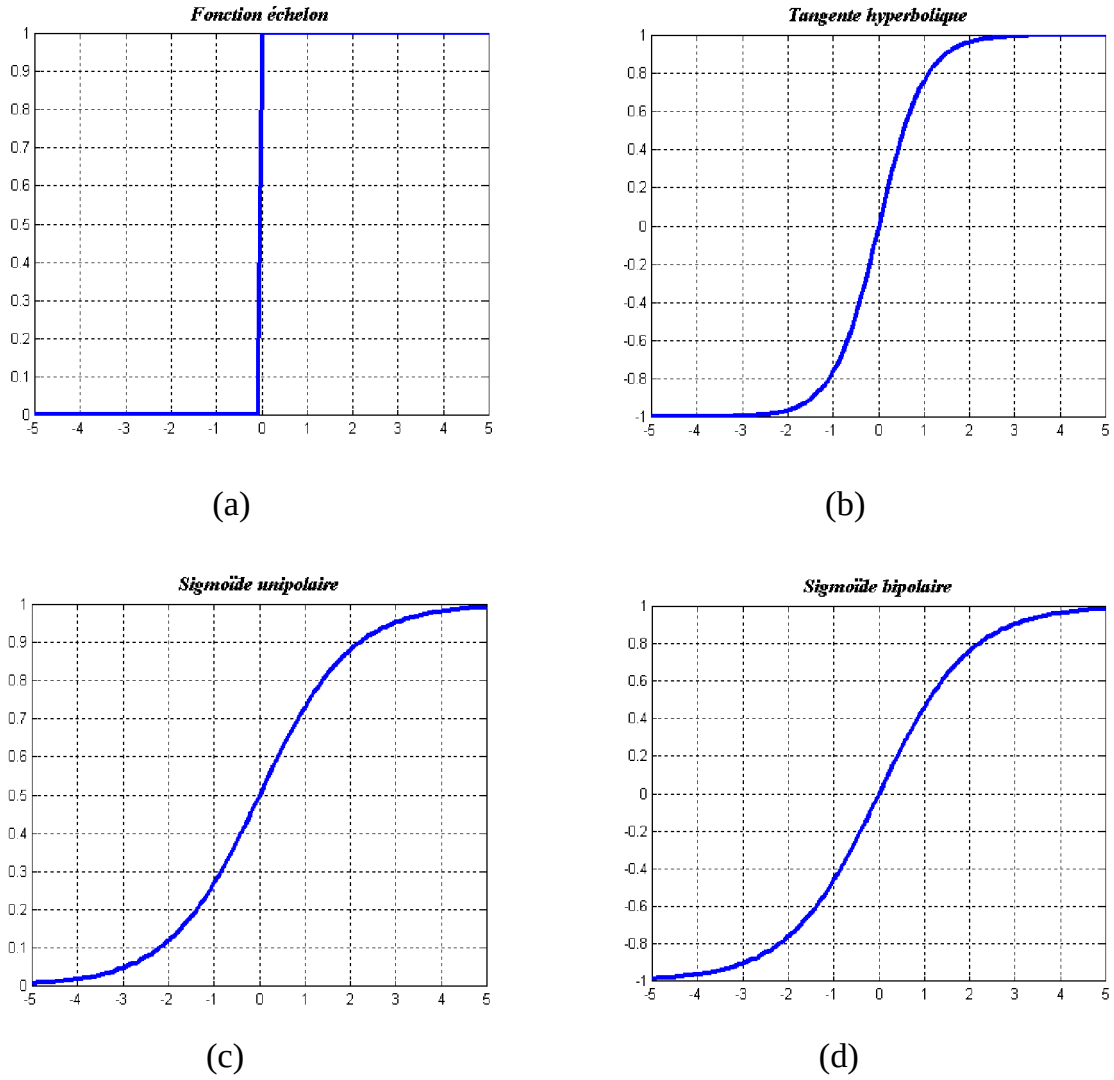


Figure III.2 : Différentes fonctions d'activation

Un neurone formel ne réalise donc rien d'autre qu'une somme pondérée suivie d'une non linéarité. C'est l'association de tels éléments simples sous la forme de réseaux qui permet de réaliser des fonctions utiles pour des applications industrielles.

III.4. Les réseaux de neurones formels

On distingue deux grands types d'architectures de réseaux de neurones : les réseaux de neurones *non bouclés (statiques)* et les réseaux de neurones *bouclés (dynamiques ou récurrents).*

III.4.1. Les réseaux de neurones non bouclés (statiques)

Un réseau de neurones non bouclé réalise une (ou plusieurs) fonctions algébriques de ses entrées, par composition des fonctions réalisées par chacun de ses neurones.

Un réseau de neurones non bouclé est représenté graphiquement par un ensemble de neurones connectés entre eux, l'information circulant des entrées vers les sorties sans retour en arrière ; si l'on représente le réseau comme un graphe dont les nœuds sont les neurones et les arêtes les connexions entre ceux-ci, le graphe d'un réseau non bouclé est acyclique. Le terme de connexions est une métaphore : dans la très grande majorité des applications, les réseaux de neurones sont des formules algébriques dont les valeurs numériques sont calculées par des programmes d'ordinateurs, non des objets physiques (circuits électroniques spécialisés) ; néanmoins, le terme de connexion, issu des origines biologiques des réseaux de neurones, est passé dans l'usage, car il est commode quoique trompeur. Il a même donné naissance au terme de « connexionisme ».

III.4.1.1. Perceptron multicouches (PMC)

La Figure III.3 représente un réseau de neurones non bouclé qui a une structure particulière, très fréquemment utilisée : c'est le PMC. Ses neurones sont organisés en couches successives. Chaque neurone d'une couche reçoit des signaux de la couche précédente et transmet le résultat à la suivante, si elle existe.

Les neurones d'une même couche ne sont pas interconnectés. Un neurone ne peut donc envoyer son résultat qu'à un neurone situé dans une couche postérieure à la sienne. L'orientation du réseau est fixée par le sens, unique, de propagation de l'information, de la couche d'entrée vers la couche de sortie.

Pour le réseaux considéré, les notions de couches d'entrée et de sortie sont donc systématiques. Ces dernières constituent l'interface du réseau avec l'extérieur. La couche d'entrée reçoit les signaux (ou variables) d'entrée et la couche de sortie fournit les résultats. Enfin, les neurones des autres couches (couches cachées) n'ont aucun lien avec l'extérieur et sont appelés neurones cachés.

Par convention, les neurones d'entrée ont toujours une fonction d'activation « identité », laissant passer l'information sans la modifier.

En ce qui concerne les neurones de sortie, on peut leur associer une fonction d'activation linéaire ou non, dérivable ou non, suivant la nature du problème à résoudre.

En ce qui concerne la fonction d'activation associée aux neurones cachés, on utilise dans le cadre de cette thèse une fonction d'activation de la famille des sigmoïdes.

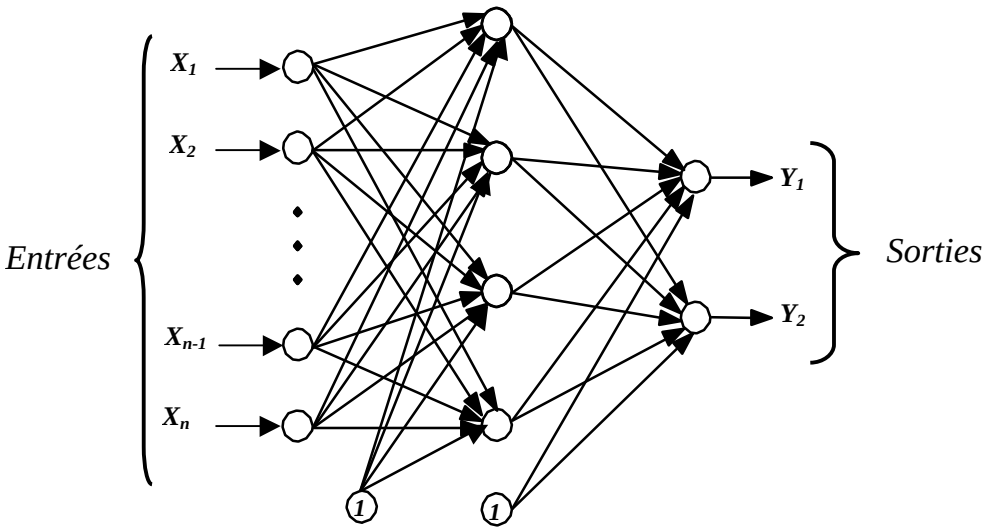


Figure III.3 : Perceptron multicouche à une couche cachée

Les réseaux de neurones non bouclés sont des objets statiques : si les entrées sont indépendantes du temps, les sorties le sont également. Ils sont utilisés principalement pour effectuer des tâches d'approximation de fonction non linéaire, de classification ou de modélisation de processus statiques non linéaires.

III.4.2. Les réseaux de neurones bouclés (dynamiques)

Contrairement aux réseaux de neurones non bouclés dont le graphe de connexions est acyclique, les réseaux de neurones bouclés peuvent avoir une topologie de connexions quelconque, comprenant notamment des boucles qui ramènent aux entrées la valeur d'une ou plusieurs sorties. Pour qu'un tel système soit causal, il faut évidemment qu'à toute boucle soit associé un retard : un réseau de neurones bouclé est donc un système dynamique, régi par des équations différentielles ; comme l'immense majorité des applications sont réalisées par des programmes d'ordinateurs, on se place dans le cadre des systèmes à temps discret, où les équations différentielles sont remplacées par des équations aux différences.

Un réseau de neurones bouclé à temps discret est donc régi par une (ou plusieurs) équations aux différences non linéaires, résultant de la composition des fonctions réalisées par chacun des neurones et des retards associés à chacune des connexions.

La forme la plus générale des équations régissant un réseau de neurones bouclé est appelée forme canonique ;

$$\begin{aligned} x(k+1) &= \varphi\left[x(k)\ ,\ u(k)\right] \\ y(k) &= \psi\left[x(k)\ ,\ u(k)\right] \end{aligned} \qquad \text{(III.5)}$$

où φ et ψ sont des fonctions non linéaires réalisées par un réseau de neurones non bouclé (par exemple, mais pas obligatoirement, un Perceptron multicouche), et où k désigne le temps (discret). Il est à noter au passage que tout réseau de neurones, bouclé aussi compliqué soit-il, peut être mis sous une forme canonique (Dreyfus, 1998). Cette forme canonique est représentée sur la Figure III.4

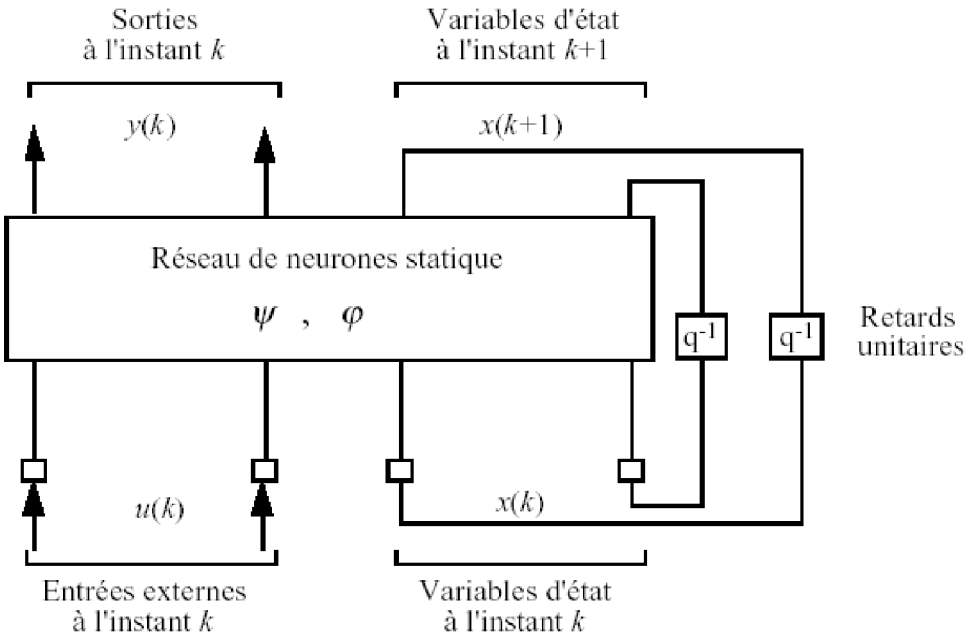


Figure III.4 : Forme canonique d'un réseau de neurones bouclé

Comme il est montré dans le Chapitre I, les états du réseau de neurones bouclé peuvent être exprimés en fonction des valeurs passées de la grandeur de sortie.

Les réseaux de neurones bouclés sont utilisés pour effectuer des tâches de modélisation de systèmes dynamiques, de commande de processus, ou de filtrage.

III.4.3 Propriétés des Réseaux de Neurones Formels

Les propriétés essentielles des réseaux de neurones sont :

III.4.3.1. Le parallélisme

Cette notion se situe à la base de l'architecture des réseaux de neurones considérés comme ensemble d'entités élémentaires qui travaillent simultanément.

Le parallélisme permet une rapidité de calcul supérieure mais exige de penser et de poser différemment les problèmes à résoudre.

III.4.3.2. La capacité d'adaptation

Elle se manifeste par la capacité d'apprentissage qui permet de tenir compte des nouvelle contraintes ou de nouvelles données du monde extérieur. Certains réseaux se caractérisent aussi par leur capacité d'auto-organisation qui assure leur stabilité en tant que systèmes dynamiques capables de tenir compte de situations non encore connues.

III.4.3.3. La facilité de construction

La simulation informatique de réseaux de neurones pour une petite application est simple et ne nécessite qu'un temps de développement assez court. Pour des applications plus complexes, l'utilisation de simulateur ou de carte accélératrice se révèle utile.

III.4.3.4. Propriété fondamentale : L'Approximation universelle parcimonieuse

III.4.3.4.a. l'approximation universelle

Les travaux de Cybenko (Cybenko,1989) et Funahashi (Funahashi, 1989) ont prouvé la possibilité d'approcher des fonctions continues, au sens de la norme uniforme sur les compacts, par des réseaux de neurones.

Les réseaux considérés sont de type réseau à une couche de neurones cachés à fonction d'activation non linéaire, et à neurones de sortie linéaires. Dans le cas du théorème de Cybenko, l'hypothèse sur la fonction d'activation est qu'elle a pour limite 0 en $-\infty$ et 1 en $+\infty$, dans celui de Funahashi, qu'elle est croissante, non constante et bornée. Les fonctions non continues, mais mesurables, peuvent aussi être approchées, mais dans un sens moins fort (Hornik et al., 1990). Il existe par ailleurs quelques résultats sur le nombre de neurones requis pour approcher une fonction avec une précision fixée, pour certaines classes de fonctions (Neelakantan et Guiver, 1998).

Ces résultats affirment donc, pour toute fonction déterministe usuelle, l'existence d'une approximation par un réseau de neurones. Les réseaux complètement connectés ou à couches, et à neurones cachés sigmoïdaux, remplissent les conditions d'application des théorèmes.

Dans ce travail, nous utilisons systématiquement ce type de réseaux, à l'exclusion par exemple des réseaux utilisant des fonctions à base radiale (RBF). Une raison en est que, même si ces réseaux jouissent également de propriétés d'approximation intéressantes, et même si leur apprentissage peut être réalisé à l'aide de la méthode des moindres carrés ordinaires, leur utilisation est souvent beaucoup moins économique, ou parcimonieuse du point de vue du nombre de connexions, que celle des réseaux à sigmoïdes. En toute rigueur, les réseaux à RBF peuvent être aussi parcimonieux que les réseaux à sigmoïdes, mais à condition d'ajuster la position des centres des RBF de manière non linéaire (Park et Sandberg , 1991), ce qui supprime le principal intérêt des RBF : la simplicité de l'apprentissage par la méthode des moindres carrés ordinaires.

III.4.3.4.b. La parcimonie

Lorsque l'on cherche à modéliser un processus à partir des données, on s'efforce toujours d'obtenir les résultats les plus satisfaisants possibles avec un nombre minimum de paramètres ajustables. Dans cette optique, (Hornik et al., 1994) a montré que : *Si le résultat de l'approximation (c'est-à-dire la sortie du réseau de neurones) est une fonction non linéaire des paramètres ajustables, elle est plus parcimonieuse que si elle est une fonction linéaire de ces paramètres. De plus, pour des réseaux de neurones à fonction d'activation sigmoïdale, l'erreur commise dans l'approximation varie comme l'inverse du nombre de neurones cachés, et elle est indépendante du nombre de variables de la fonction à approcher. Par conséquent, pour une précision donnée, donc pour un nombre de neurones cachés donné, le nombre de paramètres du réseau est proportionnel au nombre de variables de la fonction à approcher.*

Ce résultat s'applique aux réseaux de neurones à fonction d'activation sigmoïdale puisque la sortie de ces neurones n'est pas linéaire par rapports aux poids synaptiques.

La spécificité des réseaux de neurones réside donc dans le caractère parcimonieux de l'approximation : à précision égale, les réseaux de neurones nécessitent moins de paramètres ajustables (les poids des connexions) que les

approximateurs universels couramment utilisés ; plus précisément, le nombre de poids varie *linéairement* avec le nombre de variables de la fonction à approcher, alors qu'il varie *exponentiellement* pour la plupart des autres approximateurs (Hornik et al., 1994).

Qualitativement, la propriété de parcimonie peut se comprendre de la manière suivante : lorsque l'approximation est une combinaison *linéaire* de fonctions élémentaires fixées (des monômes par exemple, où des gaussiennes à centres et écarts-types fixes), on ne peut ajuster que les coefficients de la combinaison ; en revanche, lorsque l'approximation est une combinaison linéaire de fonctions non linéaires à paramètres ajustables (un perceptron multicouche par exemple), on ajuste à la fois les coefficients de la combinaison et la forme des fonctions que l'on combine. Ainsi, dans un perceptron multicouche, les poids de la première couche déterminent la forme de chacune des sigmoïdes réalisées par les neurones cachés, et les poids de la seconde couche déterminent une combinaison linéaire de ces fonctions. On conçoit facilement que cette souplesse supplémentaire, conférée par le fait que l'on ajuste la forme des fonctions que l'on superpose, permet d'utiliser un plus petit nombre de fonctions élémentaires, donc un plus petit nombre de paramètres ajustables. Nous allons voir ultérieurement pourquoi cette propriété de parcimonie est précieuse dans les applications industrielles.

Rappelons que ces résultats concernent l'utilisation de réseaux de neurones pour l'approximation uniforme de fonctions connues ; il est pourtant rare que les réseaux de neurones soient mis en œuvre dans ce cadre. Nous allons montrer dans le paragraphe suivant que la technique des réseaux de neurones est généralement utilisée comme une méthode de modélisation statistique.

III.5. Réseaux de neurones et régressions non linéaires.

Dans la pratique, on n'utilise pas les réseaux de neurones pour réaliser des approximations de fonctions connues. Le plus souvent, le problème qui se pose à l'ingénieur est le suivant : il dispose d'un ensemble de mesures de variables d'un processus de nature quelconque (physique, chimique, économique, financier, ...), et du résultat de ce processus ; il suppose qu'il existe une relation déterministe entre ces

variables et ce résultat, et il cherche une forme mathématique de cette relation, valable dans le domaine où les mesures ont été effectuées, sachant que les mesures sont en nombre fini, qu'elles sont certainement entachées de bruit, et que toutes les variables qui déterminent le résultat du processus ne sont pas forcément mesurées.

En d'autres termes, l'ingénieur cherche un modèle du processus qu'il étudie, à partir des mesures dont il dispose, et d'elles seules : on dit qu'il effectue une modélisation « boîte noire ».

Dans le jargon des réseaux de neurones, les données à partir desquelles on cherche à construire le modèle s'appellent *des exemples.*

En quoi la propriété d'approximation parcimonieuse peut-elle être utile pour résoudre ce genre de problèmes ? Ce que l'ingénieur cherche à obtenir à l'aide de son modèle, c'est la vraie fonction qui relie la grandeur de sortie que l'on veut modéliser aux variables d'entrées qui la déterminent, en d'autres termes, la fonction que l'on obtiendrait en faisant une infinité de mesures de la grandeur de sortie pour chaque valeur possible de variables d'entrées : en termes de statistiques, l'ingénieur cherche la fonction de régression de la grandeur à modéliser. Cette fonction est inconnue, mais on peut en chercher une approximation à partir des mesures disponibles : les réseaux de neurones sont donc de bons candidats pour cela, si la fonction de régression cherchée est non linéaire. Cette approximation est obtenue en estimant les paramètres d'un réseau de neurones au cours d'une phase dite *d'apprentissage*. C'est ici que la propriété d'approximation parcimonieuse des réseaux de neurones est précieuse : en effet, le nombre de mesures nécessaires pour estimer les paramètres de manière significative est d'autant plus grand que le nombre de paramètres est grand. Ainsi, pour modéliser une grandeur avec une précision donnée à l'aide d'un réseau de neurones, il faut *moins de données* que pour la modéliser, avec une précision comparable, à l'aide d'une régression linéaire multiple ; de manière équivalente, un réseau de neurones permet, avec les mêmes données disponibles, de réaliser une approximation plus précise qu'une régression linéaire multiple (Rivals et al., 1995).

De manière générale, un réseau de neurones permet donc de faire un meilleur usage des mesures disponibles que les méthodes de régression non linéaires conventionnelles.

Ce gain peut être considérable lorsque le processus à modéliser dépend de plusieurs variables.

Ainsi, à la lumière de cette propriété fondamentale, la technique des réseaux de neurones apparaît comme une puissante méthode de régression non linéaire : ce n'est donc rien d'autre qu'une extension des méthodes de régression linéaire ou multilinéaires proposées par tous les logiciels qui permettent de faire de la modélisation de données.

Contrairement à une croyance répandue, la technique des réseaux de neurones ne relève donc pas de l'Intelligence Artificielle au sens classique du terme, mais elle constitue une branche des statistiques appliquées. Il ne faut donc pas être victime du vocabulaire utilisé (neurones, apprentissage, etc.) ; le tableau ci-dessous résume les équivalences entre le vocabulaire des statistiques et celui des réseaux de neurones.

RESEAUX DE NEURONES	STATISTIQUES
Choix de l'architecture	Choix de la famille de fonctions destinées à approcher la fonction de régression
Ensemble d'apprentissage	Observations
Apprentissage	Estimation des paramètres de l'approximation de la fonction de régression
Généralisation	Interpolation, extrapolation

Tableau III.1 : Réseaux de neurones et statistiques

III.6. Apprentissage d'un réseau de neurones

Un réseau de neurones définit une famille de fonctions. L'apprentissage consiste à déterminer la solution du problème posé au sein de cette famille de fonctions. Ces fonctions pourront avoir des capacités limitées comme les fonctions linéaires ou au contraire permettre la construction de fonctions aussi complexes qu'on le désire comme les PMCs. Le principe d'apprentissage est l'optimisation d'une *fonction de coût* calculée à partir des exemples de la base d'apprentissage et de la sortie du réseau de neurones. Les méthodes numériques utilisées sont le plus souvent des méthodes approchées basées sur des techniques de gradient (cf. Chapitre II).

En d'autres termes, l'apprentissage d'un réseau de neurones consiste à déterminer les valeurs des poids permettant à la sortie du réseau de neurones d'être aussi proche que possible de l'objectif fixé.

III.6.1. Algorithme de Rétro-propagation (RP)

L'algorithme de *rétropropagation du gradient* (Back-propagation, en Anglais) est certainement à la base des premiers succès des réseaux de neurones. Sa mise en application a permis au domaine du « connexionnisme » de sortir de la période de silence qui a régné après la sortie du livre « Perceptrons » de (Minsky et Papert, 1969). Il figure aujourd'hui parmi les algorithmes d'apprentissage les plus utilisés.

Selon la littérature, la *RP* a été proposée plusieurs fois et de manière indépendante par : (Bryson et Ho, 1969), (Werbos, 1974), (Parker, 1985) , et enfin Rumelhart et les membres du groupe PDP en 1986 (Rumelhart et al., 1986). Une approche similaire a également été proposée par (Le Cun, 1985).

D'après (Hassibi et Stork, 1993), des liens peuvent être établis avec la technique de (Robbins et Monro, 1951). Cependant, la popularisation de la *RP* et son développement restent liés aux travaux du groupe PDP dirigé par Rumelhart.

On considère un réseau à trois couches illustré par la Figure III.5.

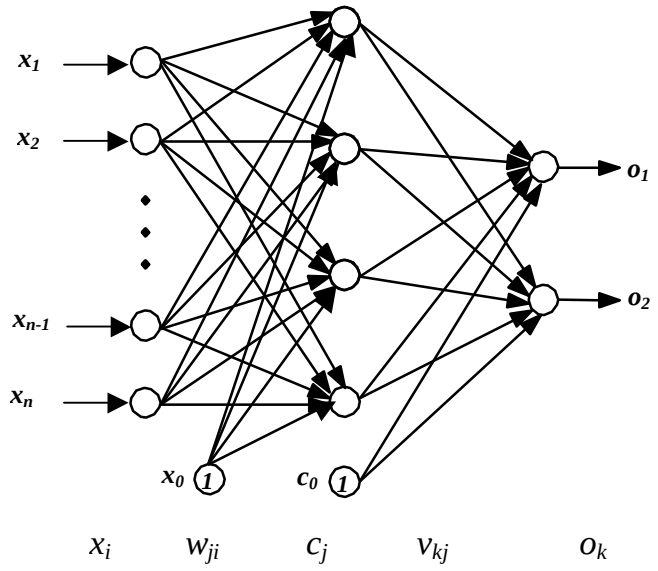


Figure III.5 : Définition des notations pour un PMC

Les conventions de notation sont les suivantes :

- o_k : activation de la k^{ème} unité de sortie, k = 1, …, M;
- t_k : activation désirée de la k^{ème} unité de sortie;
- c_j : activation de la j^{ème} unité cachée, j = 0, 1, …, n_h ; c_0 = 1 : c'est l'entrée du biais pour la couche de sortie;
- x_i : i^{ème} entrée externe du réseau ; i = 0, 1, …, n; x_0 = 1 : entrée du biais pour la couche cachée;
- w_{ji} : poids d'une connexion entre la i^{ème} entrée et la j^{ème} unité cachée;
- v_{kj} : poids d'une connexion entre la $j^{ème}$ unité cachée et la k^{ème} unité de sortie.

Les indices *i*, *j* et *k* font référence aux unités d'entrée, aux unités cachées et aux unités de sortie, respectivement. L'exposant *p* correspond au numéro de l'exemple présenté à l'entrée du réseau : p = 1,..., n_A, où n_A est le nombre d'exemples d'apprentissage. Le $p^{ème}$ exemple est noté $x^p = \left[x_0^p, ..., x_i^p, ..., x_n^p\right]$ et la $i^{ème}$ composante x_i^p désigne la $i^{ème}$ entrée lorsque le $p^{ème}$ exemple est présenté au réseau. Pour un exemple *p*, la $j^{ème}$ unité cachée a l'entrée résultante I_j^p :

$$I_j^p = \sum_{i=0}^{n} w_{ji} x_i^p \qquad \text{(III.6)}$$

et une activation c_j^p :

$$c_j^p = f\left(I_j^p\right) = f\left(\sum_{i=0}^{n} w_{ji} x_i^p\right) \qquad \text{(III.7)}$$

où *f* est la fonction d'activation. La $k^{ème}$ unité de sortie reçoit une entrée résultante I_k^p définie par :

$$I_k^p = \sum_{i=0}^{n} v_{kj} c_j^p \qquad \text{(III.8)}$$

et génère en sortie l'activation o_k^p :

$$o_k^p = f\left(I_k^p\right) \qquad \text{(III.9)}$$

Comme la fonction d'activation des neurones de sorties pour un PMC est linéaire, l'équation (III.9) devient :

$$o_k^p = I_k^p \tag{III.10}$$

La fonction de coût usuelle est l'erreur quadratique moyenne définie par :

$$E(w) = \frac{1}{2}\sum_{k,p} \left(t_k^p - o_k^p\right)^2 \tag{III.11}$$

où ***w*** est le vecteur contenant tous les poids du réseau. La fonction $E(\boldsymbol{w})$ est continue et différentiable par rapport à chaque poids. Pour déterminer les poids qui la minimisent, on peut donc utiliser l'algorithme de descente du gradient (Moussaoui et al., 1999). Pour faciliter la notation, $E(\boldsymbol{w})$ sera notée E dans ce qui suit.

Pour les poids des connexions des unités cachées vers les unités de sortie, le terme d'adaptation des poids au cours de l'apprentissage est défini par :

$$\begin{aligned} \Delta v_{kj} &= -\eta \frac{\partial E}{\partial v_{kj}} \\ &= \eta \sum_p \delta_k^p c_j^p \end{aligned} \tag{III.12}$$

avec :

$$\delta_k^p = \left(t_k^p - o_k^p\right) \tag{III.13}$$

dans le cas de sorties linéaires.

Pour les poids des connexions entre la couche d'entrée et la couche cachée, le terme d'adaptation des poids est :

$$\begin{aligned} \Delta v_{kj} &= -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \sum_p \frac{\partial E}{\partial c_j^p}\frac{\partial c_j^p}{\partial w_{ji}} \\ &= \eta \sum_{k,p} \left(t_k^p - o_k^p\right) v_{kj} f'(I_j^p) x_i^p = \eta \sum_p \delta_j^p x_i^p \end{aligned} \tag{III.14}$$

avec :

$$\delta_j^p = f'(I_j^p) \sum_k v_{kj} \delta_k^p \qquad \text{(III.15)}$$

On peut constater que les équations (III.12) et (III.14) ont la même forme et ne diffèrent que par la définition de la quantité δ. Ces formules se généralisent facilement aux cas des réseaux possédant un nombre quelconque de couches cachées.

D'après l'équation (III.15), le calcul de δ_j pour une unité cachée j nécessite les δ_k des unités de sortie, qui sont fonctions des erreurs en sortie du réseau (t_k - o_k). Ainsi, pour corriger les poids des connexions entre la couche d'entrée et la couche cachée, on a besoin de rétropropager l'erreur depuis les sorties vers les entrées, d'où le nom de l'algorithme d'apprentissage : *rétro-propagation de 1'erreur*.

D'après l'équation (III.15), le calcul de δ_j pour une unité cachée j utilise la dérivée de l'activation de cette même unité et la somme des δ_k pour toutes les unités de sortie. Le calcul de δ_j se fait donc de manière locale (indépendamment de toutes les autres unités cachées). Ceci permet d'envisager une *parallélisation* de l'algorithme de rétro-propagation *(RP).*

Après avoir calculé le terme d'adaptation des poids, la mise à jour se fait selon la formule suivante :

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji} \qquad \text{(III.16)}$$

et

$$v_{kj}(t+1) = v_{kj}(t) + \Delta v_{kj} \qquad \text{(III.17)}$$

où *t* est l'indice de l'itération. Le mode d'adaptation des poids tel qu'il est présenté par les équations (III.12) et (III.14) s'appelle mode « *batch* ». La mise à jour des poids se fait après avoir passé en revue tous les exemples d'apprentissage. Ce mode d'apprentissage est encore appelé déterministe hors ligne « *off-line » ou « by epoch ».*

Une autre approche consiste à modifier les poids après chaque présentation d'un exemple d'apprentissage. C'est l'apprentissage en mode en ligne « *on-line* » ou « *by pattern* ».

Il y a des avantages et des inconvénients pour chaque mode d'adaptation des poids, et le choix de l'un ou de l'autre dépend du problème à traiter.

Les algorithmes « *off-line* » sont faciles à analyser pour ce qui concerne les propriétés de convergence. Ils peuvent utiliser un taux d'apprentissage optimum à chaque itération et peuvent conduire à des solutions assez précises (avec de faibles variantes). En revanche, ils ont l'inconvénient d'induire un temps de calcul du terme d'adaptation de poids dépendant de la taille de l'ensemble d'apprentissage.

Les méthodes « *on-line* » peuvent être utilisées lorsque les exemples ne sont pas tous disponibles au début de l'apprentissage, et quand on désire réaliser une adaptation continue à partir d'une suite de couples *entrée-sortie* issus d'une relation qu'on cherche à identifier.

III.6.2. Variantes de l'algorithme de RP

Depuis son introduction, l'algorithme de *RP* a été largement étudié et plusieurs modifications y ont été apportées. L'algorithme de base décrit ci-dessus converge très lentement pour les réseaux multicouches.

Les variations apportées à l'algorithme de *RP* ont pour objectifs l'accélération de la convergence du processus d'apprentissage et l'amélioration de la capacité de généralisation. Nous présentons ci-dessous quelques variantes parmi les plus importantes.

III.6.2.1. Choix de la fonction coût

Le choix de la fonction d'erreur utilisée pour l'apprentissage des réseaux de neurones multicouches a une certaine influence sur la rapidité d'apprentissage et sur la qualité de généralisation du réseau. Cette question a été étudiée par plusieurs chercheurs (Gayner et Downs, 1994). Le critère d'erreur le plus utilisé est la fonction d'erreur quadratique moyenne (cf. équation (III.11)).

Cette fonction a tendance à amplifier les erreurs les plus importantes. Par conséquent, au cours de l'apprentissage, la mise à jour des poids est largement déterminée par la correction des grandes erreurs, ce qui est recherché en général (Hassibi et Stork, 1993).

Cependant le choix de la fonction quadratique n'est pas la seule possibilité. On peut remplacer $\|t_k^p - o_k^p\|^2$ par tout autre fonction $\varepsilon(t_k^p, o_k^p)$ différentiable et minimale lorsque ses deux arguments sont égaux. Le développement précédent montre que seule l'expression de l'équation (III.12) dépend de la fonction coût. Le reste de l'algorithme de *RP* reste inchangé.

III.6.2.2. Introduction d'un terme de moment

Le paramètre η (appelé taux d'apprentissage) joue un rôle important. S'il est trop faible, la convergence est lente, et, s'il est trop grand, l'algorithme oscille entre des points différents à cause de l'existence de vallées et de plateaux à la surface de la fonction coût.

Pour stabiliser la recherche des poids optimisant la fonction coût, une méthode consiste à ajouter un terme dit de « *moment* » à l'expression d'adaptation des poids (Plaut et al., 1990). L'idée est de donner une certaine « *inertie* » pour chaque poids, de sorte que sa mise à jour ne se fasse pas de manière brutale.

Ceci permet alors d'utiliser un taux d'apprentissage relativement important sans pour autant augmenter les oscillations de la trajectoire sur la surface d'erreur.

La nouvelle formule d'adaptation des poids est définie par :

$$\Delta w(t+1) = -\eta \frac{\partial E}{\partial w} + \alpha \Delta w(t) \qquad \text{(III.18)}$$

où α est le terme de moment dont la valeur est souvent prise proche de 1 (≈ 0.9). Cette méthode peut être utilisée en modes « off-line » ou « on-line ».

III.6.2.3. Taux d'apprentissage adaptatifs

Les paramètres η et α de l'algorithme *RP* ne sont pas faciles à fixer a priori pour un problème donné. En outre, des valeurs données peuvent être bien adaptées au début de l'apprentissage, sans l'être nécessairement au milieu ou à la fin du processus. Pour résoudre ce problème, certains auteurs ont proposé d'ajuster automatiquement ces paramètres au cours de l'apprentissage.

Plusieurs heuristiques d'adaptation dynamique du taux d'apprentissage ou du terme de moment ont été proposées (Jacobs, 1988), (Silva et Almeida, 1990), (Tollenaere, 1990). On se contente ici de rappeler le principe des méthodes adaptatives et de citer l'algorithme de Silva et Almeida comme exemple.

Le principe des méthodes adaptatives est d'étudier à chaque itération l'effet de la mise à jour des poids des connexions sur la fonction coût. Si cette fonction augmente, alors le taux d'apprentissage η_{pq} associé au poids w_{pq} doit être réduit. En revanche, si la fonction coût décroît régulièrement suite aux modifications des poids, alors on peut augmenter η_{pq} pour accélérer la recherche de l'optimum. Différents critères peuvent être retenus, comme par exemple la diminution de la fonction coût pendant plusieurs itérations successives. En partant de ce principe, Silva et Almeida ont proposé la règle suivante :

$$\eta_{pq}(t) = \begin{cases} \gamma\eta_{pq}(t-1), & \text{si} \quad \dfrac{\partial E}{\partial w_{pq}(t)} \dfrac{\partial E}{\partial w_{pq}(t-1)} > 0 \\ \zeta\eta_{pq}(t-1) & \text{sinon} \end{cases} \qquad \text{(III.19)}$$

où w_{pq} est un poids quelconque du réseau, γ et ζ sont des paramètres respectivement supérieur et inférieur à 1 (par exemple: $\gamma = 1.2$ et $\zeta = 0.7$). Les auteurs montrent expérimentalement que les valeurs précises de ces deux paramètres ne sont pas décisives. Ils proposent également d'utiliser un terme de moment identique et non adaptatif pour tous les poids du réseau. La procédure d'adaptation des poids est alors la suivante :

$$\begin{cases} w_{pq}(t+1) = w_{pq}(t) - \eta_{pq}(t)\, z_{pq}(t) \\ \text{et} \\ z_{pq}(t) = \dfrac{\partial E}{\partial w_{pq}(t)} + \alpha\ z_{pq}(t-1) \end{cases} \qquad \text{(III.20)}$$

où α est le terme de moment.

III.6.2.4. Autres procédures d'optimisation

L'algorithme de *RP* dans sa forme de base utilise la technique de descente du gradient. Celle-ci est parmi les plus simples, mais elle n'est pas très efficace car elle utilise peu d'information sur la surface de l'erreur. Dans la littérature, on trouve une grande quantité de techniques plus sophistiquées (Fletcher, 1987). Une synthèse des principales méthodes est présentée dans (Hertz et al., 1991).

L'une des procédures d'optimisation les plus utilisées sont les méthodes dites du deuxième ordre. Elles consistent à effectuer le développement au second ordre de la fonction coût $E(\boldsymbol{w})$ autour du point courant $\boldsymbol{w_o}$. Et en négligeant les termes d'ordre supérieur, on peut écrire :

$$E(w) = E(w_0) + (w - w_0)^T \nabla E(w_0) + \frac{1}{2}(w - w_0)^T H (w - w_0) \qquad \text{(III.21)}$$

avec $\boldsymbol{H}$ la matrice Hessienne calculée au point $\boldsymbol{w_o}$. En dérivant l'équation (III.21), on obtient :

$$\nabla E(w) = \nabla E(w_0) + H(w - w_0) \qquad \text{(III.22)}$$

En annulant la dérivée, l'équation (III.22) donne une estimation de la localisation du minimum :

$$w = w_0 - H^{-1} \nabla E(w_0) \qquad \text{(III.23)}$$

En posant $\boldsymbol{w_0} = \boldsymbol{w(t)}$ et $\boldsymbol{w} = \boldsymbol{w(t+1)}$, on obtient une procédure itérative d'estimation du vecteur des poids minimisant la fonction coût :

$$w(t+1) = w(t) - H^{-1} \nabla E(w(t)) \qquad \text{(III.24)}$$

Cette méthode est appelée *méthode de Newton*. Elle permet une convergence rapide chaque fois que les poids se trouvent au voisinage de la solution. Son inconvénient est la nécessité de calculer $\boldsymbol{H}^{-1}$.

Deux solutions ont été proposées face à ce problème. L'une consiste à négliger les termes non diagonaux de $\boldsymbol{H}$ *(méthode pseudo-Newton)* (Le Cun et al., 1989), 1'autre à estimer itérativement la matrice $\boldsymbol{H}^{-1}$ *(méthode quasi-Newton)*. Pour plus de détails sur ces techniques, (cf. Chapitre II).

Les méthodes de minimisation présentées jusqu'ici se basent sur l'utilisation de dérivées de la fonction coût. Une autre approche consiste à utiliser des procédures de minimisation stochastique (Bruenelli, 1994) ou les algorithmes génétiques (Goldberg, 1989). Ces algorithmes effectuent une recherche globale et présentent par conséquent moins de risques d'aboutir à des minima locaux assez superficiels. Cependant, ils nécessitent des calculs plus lourds.

III.7. Etude du pouvoir de généralisation des réseaux de neurones

La généralisation concerne la tâche accomplie par le réseau une fois son apprentissage achevé (Gallinari, 1997). Elle peut être évaluée en testant le réseau sur des données qui n'ont pas servi à l'apprentissage. Elle est influencée essentiellement par quatre facteurs : la complexité du problème, l'algorithme d'apprentissage, la complexité de l'échantillon (le nombre d'exemples et la manière dont ils représentent le problème) et enfin la complexité du réseau (nombre de poids).

Parmi ces quatre facteurs, la complexité du réseau constitue un facteur important surtout dans les applications industrielles. En effet, lorsque l'on doit traiter des données bruitées, ce qui est souvent le cas en pratique, l'objectif est de trouver le modèle optimal, présentant le meilleur compromis possible entre performance d'apprentissage et capacité de généralisation.

Le problème de la généralisation est souvent vu sous deux perspectives différentes. Dans la première, la taille du réseau est fixée (en accord avec la complexité du problème) et la question est : combien d'exemples d'apprentissage sont nécessaires pour atteindre une bonne généralisation ? Cette perspective est intéressante dans les applications où l'on a la possibilité d'acquérir autant d'exemples que l'on veut. Dans le second cas, le nombre d'exemples d'apprentissage est fixé et la question est : quelle taille du réseau donne la meilleure généralisation pour ces données ? On est conduit à adopter ce point de vue lorsque l'on est limité dans la possibilité d'acquérir des données d'apprentissage : il importe alors de déterminer quelle est la taille permettant au réseau de décrire au mieux les données en notre possession.

III.7.1. Contrôle de la complexité

Dans les problèmes réels, on ne peut pas toujours faire varier la taille de l'échantillon. Pour un problème donné, on se donne alors un algorithme d'apprentissage, une complexité d'échantillon et on cherche à contrôler la complexité du réseau pour avoir la meilleure généralisation possible.

En effet, l'une des caractéristiques essentielles des réseaux de neurones artificiels est leur flexibilité et leur capacité d'adaptation à des problèmes nouveaux grâce à la modification des poids des connexions selon une règle d'apprentissage.

D'après des résultats théoriques bien établis (cf. § III.4.3.4.a.), un réseau avec une seule couche cachée composée d'un nombre suffisant d'unités est capable d'approximer toute fonction continue.

Des bornes théoriques de la taille du réseau en fonction de la taille de l'échantillon d'apprentissage ont été proposées pour garantir une bonne qualité de généralisation (Baum et Haussler, 1989).

Cependant, si ces résultats sont d'une grande importance sur le plan théorique, ils le sont nettement moins dans la pratique. En effet, ces résultats ne constituent que des preuves d'existence et ne permettent pas de déterminer l'architecture d'un réseau de neurones pour un problème donné.

Comment déterminer le nombre de couches cachées et le nombre d'unités (de neurones) par couche cachée nécessaires pour réaliser une bonne approximation? La difficulté de répondre à cette question a quelquefois été considérée comme un inconvénient majeur des modèles connexionnistes puisqu'un mauvais choix peut conduire à de médiocres performances du réseau correspondant (Urbani, 1995).

Les premières tentatives de résolution du problème de détermination de l'architecture ont consisté à tester plusieurs réseaux ayant des architectures différentes jusqu'à atteindre la performance désirée.

Ces dernières années, de nombreux travaux ont été consacrés au développement de méthodes d'optimisation de l'architecture des réseaux de neurones.

Les principaux algorithmes qui ont été proposés peuvent être classés en cinq familles :

1. Les méthodes de *pénalisation (weight decay)* : consistent à modifier la fonction coût de manière à pénaliser les poids ou les unités peu utiles pour le réseau, voire nuisibles au bon fonctionnement du réseau.
2. Les méthodes *d'injection de bruit* : *e*n général, ces méthodes ajoutent une quantité de bruit aux vecteurs d'entrée avant de les présenter au réseau pour l'apprentissage (Grandvalet et al., 1997).
3. Les algorithmes *d'élagage (pruning)*: détectent et éliminent les poids ou les unités qui contribuent peu à la performance du réseau.
4. Les algorithmes *constructifs*, ou *ascendants* : partent d'une solution approchée au problème avec un réseau simple, puis ajoutent, si nécessaire, des unités ou des couches cachées pour améliorer les performances du réseau (Langellé et Denoeux, 1996).
5. Enfin, les algorithmes *directs* définissent une architecture convenable puis réalisent l'apprentissage ou effectuent les deux opérations en même temps.

Les algorithmes de pénalisation et d'élagage sont appelés algorithmes *destructifs* ou *descendants*. Dans certains articles, les méthodes de pénalisation sont considérées comme des algorithmes d'élagage. Les trois premières familles supposent que le réseau est initialement *surdimensionné*.

Cette liste n'est pas exhaustive. Dans la suite du paragraphe, on donne un aperçu sur les méthodes les plus citées dans la littérature, à savoir les approches d'élagage car elles permettent d'aboutir à une réalisation optimale du réseau de neurones.

III.7.2. Méthodes d'élagage (Pruning)

L'élagage des ressources (poids ou unités) superflues suppose que l'on ait effectué l'apprentissage d'un réseau surdimensionné par rapport à la complexité de la tâche à résoudre.

L'élagage est l'opération de suppression de connexions ou d'unités dont la présence est jugée inutile, voire nuisible à la généralisation. Une fois l'apprentissage du réseau achevé, la question est alors : quels sont les poids ou unités qui doivent être supprimés ? Comment ajuster les poids restants pour obtenir de meilleures performances ? Comment réaliser un tel élagage de manière efficace et économique en temps de calcul ? Plusieurs méthodes existent pour l'élagage des poids dans un PMC.

La méthode *Optimal Brain Damage* (OBD), que nous détaillons dans le paragraphe suivant, proposée par (Le Cun et al., 1990) a été initialement utilisée pour l'optimisation de l'architecture des réseaux de neurones dans le domaine de la reconnaissance de caractères.

La méthode *Optimal Brain Surgeon* (OBS) proposée par Hassibi et Stork, (Hassibi et Stork,1993), (Thomas et Bloch, 1998) est une extension de la méthode optimal brain damage OBD.

III.7.2.1. Principe de la méthode OBD

Cette méthode est basée sur l'estimation de l'augmentation de la fonction coût lorsqu'un poids est supprimé du réseau (Le Cun et al., 1990). Elle repose sur l'approximation de la fonction coût par un développement limité du second ordre par rapport aux poids.

On obtient l'approximation suivante de l'erreur E :

$$\begin{aligned} E &= E(w_0 + \Delta w) \\ &= E_0 + \nabla E^T \delta w + \frac{1}{2}\delta w^T H \delta w + O(\|\delta w\|^3) \end{aligned} \qquad \text{(III.25)}$$

où E_o est la valeur de la fonction erreur où l'approximation de Taylor est effectuée et $\delta \boldsymbol{w}$ la variation des poids.

On peut réduire l'équation (III.25), en faisant l'hypothèse que la fonction d'erreur E est, dans le voisinage du minimum, proche d'une fonction quadratique, donc $O(\|\delta w\|^3) = 0$.

On réduit alors l'équation (III.25) en :

$$E = E_0 + \nabla E^T \delta w + \frac{1}{2} \delta w^T H \delta w \qquad \text{(III.26)}$$

Pour simplifier les calculs, la méthode *OBD* utilise l'approximation diagonale de H : *soit* $H_{ij} = 0$, pour chaque $j \neq i$.

La méthode utilisée pour estimer la variation de la fonction erreur, quand on supprime un poids, suppose que l'apprentissage du réseau est bien effectué ; la fonction erreur a alors atteint un minimum. Avec cette condition, le gradient de la fonction erreur par rapport à chaque poids, peut être supposé nul. Suite à une variation δw_i l'augmentation de la fonction d'erreur est :

$$\delta E_i = \frac{1}{2} h_{ii} \left(\delta w_i \right)^2 \qquad \text{(III.27)}$$

où

$$h_{ii} = \left[H \right]_{ii} = \frac{\partial^2 E}{\partial w_i^2} \qquad \text{(III.28)}$$

Supprimer le poids w_i du réseau revient à l'annuler. La variation de poids est alors :

$$\delta w_i = -w_i \qquad \text{(III.29)}$$

L'estimation de la variation dans la fonction erreur suite à la suppression du poids w_i est :

$$S_i = \delta E_i = \frac{1}{2} h_{ii} w_i^2 \qquad \text{(III.30)}$$

S_i est appelée la *pertinence* ou la *sensibilité* du poids w_i. Elle représente la quantité par laquelle sera augmentée l'erreur E suite à la suppression du poids w_i. Les poids ayant les plus faibles pertinences S_i sont supprimés.

Ensuite, une opération de ré-apprentissage est effectuée pour compenser l'effet de la suppression. Les auteurs de la méthode ont conseillé de supprimer plusieurs poids avant le ré-apprentissage du réseau. Cependant, il n'y a aucune recommandation concernant le choix du nombre de poids à supprimer.

Comme *H* est diagonale, le coût résultant de la suppression de *m* poids est égal à la somme des coûts lorsque les poids sont supprimés individuellement. La pertinence des *m* poids est estimée par :

$$S_m = \sum_{i=1}^{m} S_i = \frac{1}{2} \sum_{i=1}^{m} h_{ii} w_i^2 \qquad \text{(III.31)}$$

D'après les équations (III.30) et (III.31), on peut déduire que les poids de faibles amplitudes ne sont pas obligatoirement inutiles pour le réseau. En effet, seuls ceux qui correspondent à des faibles pertinences S_i peuvent être élagués.

III.7.2.2. Algorithme d'élagage par ODB

La procédure d'élagage par OBD se déroule parallèlement à l'algorithme d'apprentissage par rétropropagation du gradient : l'élagage OBD se fait juste après avoir trouvé un minimum local. Ensuite, la procédure « apprentissage RP + OBD » continue.

L'élagage OBD démarre seulement si la moyenne relative de variation de la fonction de coût pendant deux périodes consécutives d'apprentissage est assez petite. On la mesure à chaque itération de *RP en ligne* (voir Figure III.7).

Le processus « *RP + OBD* » s'arrête selon les conditions d'arrêt de l'algorithme de *RP* : on arrête le processus dès que l'erreur mesurée sur un ensemble indépendant de validation augmente. L'algorithme d'apprentissage et d'élagage par « *RP + OBD* » est le suivant :

1. Choisir une architecture de réseaux surdimensionnée ;
2. Apprentissage du réseau en utilisant l'algorithme RP ;
3. Calculer les dérivées secondes h_{ii} pour chaque poids et évaluer la pertinence S_i de chaque poids avec l'équation (III.30) ;
4. Ordonner les pertinences et éliminer les *q* poids de plus faible pertinence ;
5. Retourner à l'étape 2 tant que l'erreur mesurée sur l'ensemble indépendant de validation n'augmente pas.

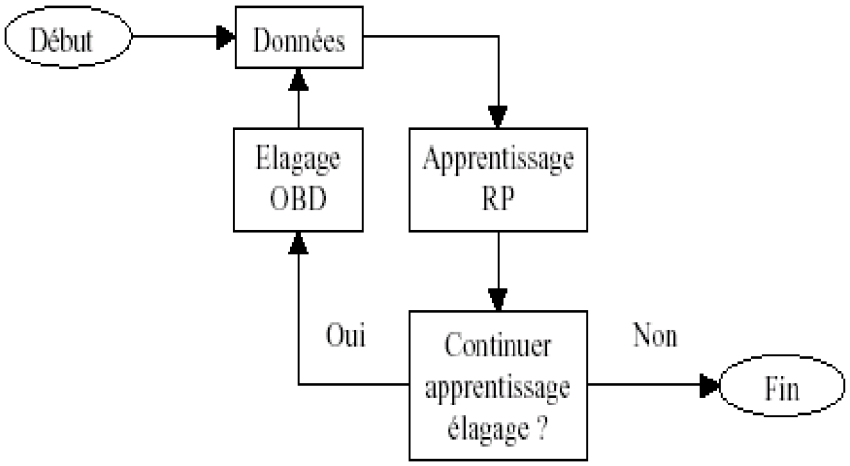


Figure III.7 : Schéma de l'Algorithme « *RP + OBD* »

Dans ce paragraphe, nous avons présenté une méthode de contrôle de complexité des réseaux de neurones : l'algorithme *OBD*. Lors de la construction d'un PMC, l'objectif est de déterminer la meilleure architecture et le meilleur jeu de poids au sens d'un critère donné (la généralisation par exemple). Sous cet angle, la construction du PMC peut être vue comme un problème de recherche d'une solution dans l'espace des architectures et des poids.

III.8. Choix des séquences d'apprentissage

Ce paragraphe donne quelques indications sur le choix des séquences d'apprentissage.

III.8.1. Séquence des entrées de commande

III.8.1.a. Contraintes sur les entrées de commande :

Elles portent sur l'amplitude et le type de signaux de commande que le processus est susceptible de recevoir pendant son fonctionnement. Les amplitudes maximales sont en général faciles à déterminer, car leur ordre de grandeur correspond aux valeurs de saturation des actionneurs, qui peuvent être estimées physiquement (puissance maximale que peut délivrer un moteur, pression maximale d'un circuit de freinage ou d'un mécanisme hydraulique, etc.).

En ce qui concerne le type de signaux à utiliser, un principe général est que les signaux utilisés pour l'identification doivent être de même nature que ceux qui seront calculés par l'organe de commande pendant l'utilisation du processus. L'idéal serait d'utiliser pour l'identification le correcteur même qui sera synthétisé à l'aide du prédicteur identifié.

Si le bruit est négligeable, une bonne démarche consiste à effectuer les expériences en asservissement, avec un correcteur simple (par exemple, un PID). Cette démarche, ou identification en boucle fermée, permet d'explorer le domaine de fonctionnement désiré, en imposant une séquence de consigne correspondant au cahier des charges.

Une autre démarche, couramment utilisée, consiste à explorer au mieux le domaine de fonctionnement, par exemple avec des créneaux de commande (riches en fréquences), d'amplitudes et de durées diverses. Notons que cette dernière solution n'est pas praticable s'il existe des contraintes sur des sorties ou des variables internes du processus, et en particulier si le processus est instable.

III.8.1.b. Fréquence d'échantillonnage :

Les réseaux de neurones étant des modèles non linéaires, il n'est pas possible de passer simplement d'un modèle discret, valable à une fréquence d'échantillonnage donnée, à un autre : il est donc nécessaire d'effectuer l'identification à la fréquence qui sera utilisée pour la commande du processus.

Si des contraintes diverses, par exemple le temps de calcul de la commande, nécessitent le choix d'une fréquence d'asservissement plus basse que la fréquence utilisée pour l'identification, il faut procéder à une nouvelle identification (un nouvel apprentissage).

III.8.2. Séquences d'apprentissage et estimation de la performance

Comme nous l'avons précisé auparavant, l'apprentissage consiste à ajuster les fonctions du prédicteur à un ensemble de points définis par des séquences d'apprentissage, ceci en minimisant une fonction de coût.

Il y a *sur-ajustement* (over-fitting) lorsque l'apprentissage conduit à annuler quasiment la fonction de coût sans que pour autant les fonctions réalisées par le réseau prédicteur approchent celles du modèle-hypothèse auquel le prédicteur est associé. Pour des séquences d'apprentissage données, le sur-ajustement se produit si le réseau prédicteur possède trop de coefficients, c'est-à-dire définit une famille de fonctions trop riche.

Pour sélectionner un réseau candidat pour une hypothèse donnée, il est donc nécessaire de répartir les données disponibles en une séquence d'apprentissage et une séquence d'estimation de la performance, dite séquence de test (cf. Chapitre II). Une séquence de test, de même type (issue de la même population) que la séquence d'apprentissage, conduit à une meilleure estimation de la variance de l'erreur de prédiction (erreur quadratique moyenne de test, notée *EQMP*) que celle obtenue avec la séquence d'apprentissage (notée *EQMA*).

L'évolution de l'EQMA et de l'EQMP lorsque l'on augmente le nombre de neurones cachés, permet de détecter un sur-ajustement, et de sélectionner le meilleur réseau de neurones parmi les candidats : dès qu'ajouter un neurone supplémentaire au réseau fait augmenter l'EQMP, même si l'EQMA continue à décroître, le nombre optimal de neurones est atteint.

Cette méthode, à condition d'utiliser des méthodes d'optimisation performantes (par exemple les méthodes quasi-newtoniennes), permet de sélectionner le réseau le plus parcimonieux.

Il est à noter au passage que dans la littérature (Sjöberg et al., 1994), la séquence de test est souvent utilisée pour arrêter l'apprentissage d'un réseau donné : dès qu'une itération d'apprentissage augmente *l'EQMP*, celui-ci est arrêté prématurément (*Early Stopping)*.

De même, la séquence de test peut servir exclusivement à estimer la performance des réseaux après apprentissage. En effet, l'apprentissage n'est arrêté que lorsqu'un minimum est atteint : l'arrêt est décidé en fonction de la valeur de la fonction de coût et de celle de la norme de son gradient.

Si le nombre de neurones est augmenté de façon incrémentale comme nous l'indiquons auparavant, on obtient forcément le prédicteur le plus parcimonieux réalisant la meilleure performance sans sur-ajustement.

Il est à signaler que le prix à payer pour ces méthodes d'optimisation des réseaux de neurones est le temps de calcul. En général, on se contente dans la pratique d'utiliser des méthodes qui ne garantissent pas l'optimalité de la solution, mais ont l'avantage d'être plus rapides.

III.8.3. Le sur-ajustement

Après cette revue des outils les plus fréquemment utilisés dans le domaine des réseaux de neurones pour sélectionner le meilleur modèle tout en évitant les solutions sur-ajustées, les questions suivantes se posent :

- Ø quelle est l'origine exacte du sur-ajustement ?
- Ø quelle démarche faudrait-il mettre en œuvre pour régler ce problème à la base ?

Le sur-ajustement caractérise une fonction dont la complexité (c'est-à-dire le nombre et la nature des degrés de libertés) est telle qu'elle est capable de s'ajuster exactement aux exemples d'apprentissage, même si ceux-ci sont entachés de bruit.

Ce phénomène est donc, à l'origine, un phénomène local : dans certains domaines des entrées, la fonction utilise localement certains de ses degrés de liberté de manière à passer précisément par certains exemples.

Cette définition du sur-ajustement suppose que tous les exemples ont la même importance et que l'on recherche effectivement une solution dont la réponse est satisfaisante.

Généralement, cette hypothèse est formalisée dans la fonction de coût choisie. Ainsi, pour éviter le sur-ajustement, il faudrait limiter l'influence de chaque exemple sur l'estimation des poids du modèle. Les valeurs de ceux-ci doivent être déterminées par l'ensemble de la base d'apprentissage, et non par un exemple particulier.

III.8.4. Problème des minima locaux

Les minima trouvés par les algorithmes d'apprentissages sont souvent des minima locaux. Le minimum trouvé dépend du point de départ de la recherche, en d'autres termes de l'initialisation des poids. En pratique, il faut effectuer plusieurs minimisations avec des initialisations différentes, pour trouver plusieurs minima et retenir le « meilleur ».

Il est néanmoins impossible et généralement inutile, de s'assurer que le minimum choisi est le minimum global (Bishop, 1995).

III.9. Conclusion

Après un aperçu historique sur les réseaux de neurones, et le rappel de quelques théorèmes relatifs à leurs capacités d'approximation, nous avons présenté un algorithme d'apprentissage très utilisé dans la communauté connexionniste : l'algorithme de rétro-propagation *(RP)*. Plusieurs travaux lui ont été consacrés et un certain nombre de modifications ont été proposées pour accélérer le temps d'apprentissage et améliorer les performances du réseau obtenu. Nous en avons rappelé dans ce chapitre les principales variantes.

Dans la pratique, l'objectif d'une modélisation statistique n'est pas d'ajuster finement un modèle sur un ensemble d'apprentissage, mais d'obtenir un bon compromis entre performances d'apprentissage et performances de généralisation.

Lors de la construction d'un réseau de neurones, l'objectif donc est de déterminer la meilleure architecture possible et le meilleur jeu de poids au sens d'un critère donné (la généralisation par exemple).

Sous cet angle, la construction du réseau de neurones peut être vue comme un problème de recherche d'une solution dans l'espace des architectures et des poids.

Le prix à payer pour cette approche est le temps de calcul. En général, on se contente dans la pratique d'utiliser des méthodes qui ne garantissent pas l'optimalité de la solution, mais qui ont l'avantage d'être plus rapides.

Résumons à présent les points fondamentaux qu'il convient de toujours garder à l'esprit lorsque l'on cherche à mettre en œuvre des réseaux de neurones :

- les réseaux de neurones sont des outils statistiques, qui permettent d'ajuster des fonctions non linéaires très générales à des ensembles de points ; comme toute méthode statistique, l'utilisation de réseaux de neurones nécessite que l'on dispose de données suffisamment nombreuses et représentatives ;
- les réseaux de neurones artificiels à une couche cachée sont des approximateurs parcimonieux ;
- les réseaux de neurones permettent de modéliser des phénomènes statiques (réseaux non bouclés) et dynamiques (réseaux bouclés ou récurrents) ;
- il est toujours souhaitable, et souvent possible, d'utiliser, pour la conception des réseaux de neurones, les connaissances mathématiques dont on dispose sur le phénomène à modéliser : les réseaux de neurones ne sont pas nécessairement des « boîtes noires ».

IV.1. Introduction

Commander un processus, c'est déterminer les commandes à lui appliquer, afin de lui assurer un comportement donné, en dépit de perturbations. Ces commandes sont délivrées par un organe de commande, qu'on élabore en plusieurs étapes.

Tout d'abord, un modèle du comportement du processus est nécessaire à la synthèse de l'organe de commande : un tel modèle sera appelé *modèle de commande*. Les qualités requises pour un modèle de commande ne sont évidemment pas les mêmes que pour un modèle de simulation ; on privilégiera notamment une structure de modèle assez simple pour faciliter la synthèse d'une commande (la rapidité du calcul), par rapport à la validité parfaite sur un très large domaine de fonctionnement qui est nécessaire à un modèle de simulation. Il faut ensuite choisir l'architecture du correcteur (correcteur par retour d'état, PID,…etc.), en fonction du modèle et du cahier des charges, correcteur qui peut également être réalisé par un réseau de neurones.

IV.2. Les étapes de la conception d'un organe de commande

IV.2.1. Choix d'un modèle du processus

Un modèle du processus est nécessaire à la synthèse de l'organe de commande. La modélisation consiste à rassembler les connaissances que l'on a du comportement dynamique du processus, par une analyse physique des phénomènes mis en jeu et une analyse des données expérimentales. Ces analyses conduisent à la définition des grandeurs caractérisant le processus, c'est-à-dire ses entrées, ses variables d'état, ses sorties, et aussi les perturbations, mesurables ou non, auxquelles il est soumis.

IV.2.2. Estimation des paramètres du modèle (identification)

Pour un modèle hypothèse donné parmi ceux retenus, le but de cette étape est de configurer et de sélectionner le meilleur modèle parmi différents modèles de la structure du modèle-hypothèse, sur la base d'un critère de performances. Bien entendu, comme le véritable but de notre démarche est la conception d'un organe de commande à partir d'un modèle, le meilleur d'entre eux est celui qui conduit aux meilleures performances du système de commande, au sens du cahier des charges.

Il est évidemment plus économique, et donc préférable, de se fonder sur un critère qui ne nécessite pas la réalisation complète du système de commande pour sélectionner ce modèle : le meilleur modèle est défini comme celui dont l'erreur de prédiction est la plus faible. Pour obtenir ce modèle, il faut le chercher au sein d'une famille de modèles paramétrés.

L'estimation des paramètres d'un modèle est donc effectuée de manière à minimiser *l'erreur de prédiction*, à partir de mesures effectuées sur le processus (ensemble d'apprentissage). La famille de modèles paramétrés que nous considérons dans ce travail est celle des réseaux de neurones, qui a l'intérêt de posséder la propriété d'approximation universelle. Leurs paramètres sont les coefficients des réseaux, et leur estimation correspond à la phase d'apprentissage.

IV.2.3. Conception de l'organe de commande

Cette étape est celle du choix de l'architecture de l'organe de commande et de la structure des éléments qui le composent, choix effectué en fonction du modèle du processus mis au point et du cahier des charges, qui spécifie les performances désirées pour le système de commande en régulation et, le cas échéant, en poursuite. L'organe de commande comprend nécessairement un élément, le *correcteur*, qui effectue le calcul de la commande à appliquer au processus à partir de la consigne et de l'état du processus, par exemple. Il comprend en général aussi d'autres éléments : un modèle de référence, un observateur, ou encore un modèle interne. Le correcteur peut avoir la structure d'un PID, ou celle d'un correcteur par retour d'état linéaire ou non. Cette structure est choisie en fonction de la tâche, définie par le cahier des charges, que doit remplir le système de commande.

IV.2.4. Estimation des paramètres du correcteur

La dernière étape consiste à configurer le correcteur pour que l'organe de commande assure la tâche définie à l'étape précédente. Cette configuration est effectuée hors-ligne à l'aide du modèle : ceci caractérise les méthodes indirectes de synthèse du correcteur.

Comme pour la modélisation, nous considérons des correcteurs réalisés par des réseaux de neurones, dont la structure a été fixée lors de la définition de l'organe de commande. L'estimation des coefficients du correcteur correspond à la phase d'apprentissage du réseau de neurones.

En réalité, la conception d'un organe de commande est une procédure itérative, surtout lorsque celui-ci n'est pas adaptatif. Car, comme nous l'avons signalé auparavant, un réseau modèle ne peut être définitivement rejeté ou validé qu'en fonction des performances du système de commande élaboré à partir de ce modèle.

Il est à signaler que les réseaux de neurones peuvent être mis en oeuvre au sein de systèmes adaptatifs, c'est-à-dire des systèmes qui apprennent en permanence. Cette propriété impose évidemment une charge de calcul en temps réel plus importante que celle des systèmes non adaptatifs, mais elle peut être indispensable dans les cas où le processus est l'objet d'évolutions lentes telles que l'usure d'un palier, la dérive d'un capteur, etc. ; si l'évolution est suffisamment lente, une remise à jour régulière, mais non permanente, des coefficients du modèle peut suffire.

IV.3. Conditions de mise en œuvre de la commande par réseaux de neurones

Le grand essor des réseaux de neurones en automatique, ne date pas de très longtemps. Après quelques tentatives vers la fin des années 80 (Psaltis et al., 1988), c'est en 1990 que K.S. Narenda et K. Parthasarathy, de Yale University aux USA, ont démontré que les réseaux de neurones peuvent être utilisés comme un outil dans le domaine de commande de processus dynamique (Narendra et Parthasarathy, 1990).

En effet, de part leur capacité d'adaptation, leur caractéristiques de traitement parallèle, leur rapidité et leur robustesse par rapport aux bruits, les réseaux de neurones sont considérés comme étant des outils très efficaces dans le domaine de la commande des processus non linéaires (Warwick, 1996). Ces propriétés ont pour conséquence la possibilité de modéliser indifféremment des non-linéarités très diverses, qu'elles soient inhérentes au processus lui-même (non-linéarité de la cinématique d'un robot mobile, ou d'un moteur à explosion) ou à ses actionneurs (non-linéarités géométriques de type saturation, hystérésis).

La parcimonie du modèle neuronal de commande permet donc de réduire le temps de calcul lorsque le système de commande utilise ce modèle de commande en temps réel. Cependant les contraintes industrielles de modernisation et d'amélioration des performances des installations sont telles qu'il n'est pas possible de stopper ou ralentir la production pour tester ces nouvelles techniques.

La mise en œuvre de ces nouvelles techniques ne doit pas donc entraîner de perturbations sur le processus et sur les produits réalisés.

Pour cela, il est indispensable d'effectuer des tests de stabilité et de sécurité en simulation et l'installation de ces nouvelles techniques sur un outil industriel ne peut avoir lieu qu'après avoir été validées.

IV.3.1. Stabilité et sécurité

Un des problèmes des techniques basées sur les réseaux de neurones est de vérifier la stabilité et, pour préciser encore la formulation du praticien, de vérifier la fiabilité et la cohérence du comportement. Cette difficulté n'apparaît pas uniquement dans le cas des réseaux neuronaux, mais aussi dans celui de nombreux circuits de régulation adaptatifs.

Garantir la stabilité et la sécurité est indispensable dans la pratique. Une application typique des réseaux de neurones en régulation est d'améliorer un procédé conventionnel existant connu. Il est rare d'opérer une substitution complète des processus conventionnels ou de pénétrer sur un terrain totalement nouveau.

Le but recherché, lorsque l'on décide d'installer un réseau de neurones, est de profiter des possibilités d'apprentissage et de facilité d'élaboration de modèles et de régulations, que les procédés conventionnels ne peuvent réaliser qu'insuffisamment ou au prix d'efforts coûteux. Cela est en particulier vrai dans le cas de régulations non linéaires ou variables en fonction du temps.

La limitation du signal de commande élaboré par le nouveau régulateur de type neuronal peut être une mesure technique concluante.

On peut ainsi garantir sécurité et stabilité, mais exclusivement pour les boucles de régulation ouvertes.

Pour les boucles de régulation fermées, il est nécessaire d'effectuer un contrôle de plausibilité, de sécurité et de stabilité supplémentaire. L'intervention la plus simple est de déclencher (désactiver) le réseau de neurones lorsque l'on atteint les limites de saturation de l'organe de commande (Hambrecht et al., 1998).

IV.3.2. Acceptation et utilisation

Outre le problème de sécurité évoqué dans le paragraphe précédent, quelques autres aspects sont déterminants pour que les réseaux de neurones soient acceptées et utilisées.

Les nouvelles technologies associées à l'utilisation des réseaux de neurones doivent être conçues pour être utilisées par des ingénieurs de mise en service ou de maintenance, sans qualifications poussées.

Par comparaison avec les techniques de régulation bien connues de type PID (proportionnel intégral dérivé), les exigences pour les applications avec des réseaux de neurones sont les suivantes :

- l'intégration dans un environnement connu avec des outils standards connus ;
- des règles et des instructions faciles à suivre pour le réglage des paramètres des applications ;
- un nombre limite de paramètres des réseaux neuronaux à régler à ne pas dépasser ;
- des règles simples et claires pour évaluer le comportement dynamique des réseaux de neurones avec assistance en cas de défauts.

Les aspects mentionnés ici sont des conditions importantes pour permettre à cette technologie d'être effectivement et rapidement utilisée et acceptée. Bien sûr, il est impossible de couvrir tous les cas particuliers. Il y a, comme pour les régulateurs PID, des limites de validité qui sont la conséquence de certaines simplifications et hypothèses.

IV.4. Etude de la commande des processus par réseaux de neurones

La commande neuronale est étroitement liée à l'identification. En effet dans la plupart des stratégies de commande, on utilise un modèle neuronal direct ou inverse du processus à commander.

Ainsi, on peut trouver plusieurs stratégies qui sont proposées dans différents travaux. Pour faire l'étude de ces stratégies, il est d'abord important de les classifier selon le rôle que les réseaux de neurones doivent remplir dans chaque cas.

Le premier critère selon lequel ces commandes peuvent être distinguées se rapporte directement à l'apprentissage des réseaux. Nous distinguons deux cas différents :

- Ø Le premier cas est celui de la commande adaptative où les poids du réseau de neurones continuent à être adaptés pendant son utilisation. Le réseau effectue un apprentissage en ligne, ce qui est très délicat en pratique.
- Ø Le second cas est celui où le réseau de neurones est entraîné hors-ligne (en temps différé), pour être utilisé par la suite sans modification de ses paramètres.

Parmi les principales stratégies de commande utilisant les réseaux de neurones, on distingue:

IV.4.1. Commande par modèle inverse

Cette commande consiste à chercher d'abord, un modèle inverse du processus, en utilisant ses sorties *y(t)* (résultant d'une entrée donnée) comme entrée du réseau (Figure IV.1).

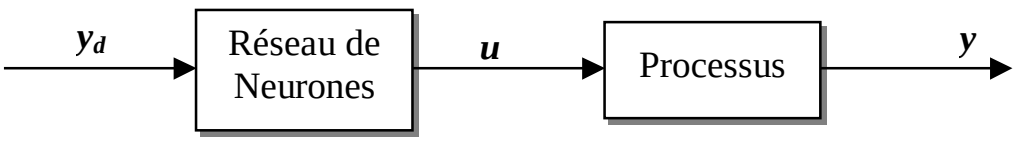


Figure IV.1 : Schéma de principe de la commande inverse

Après apprentissage, le réseau est placé devant le système pour le commander en boucle ouverte. L'apprentissage de ce réseau peut s'effectuer de trois manières différentes :

IV.4.1.1. Apprentissage indirect

On injecte au réseau les sorties désirées y_d afin qu'il fournisse un signal de commande u. La réponse du système pour ce signal de commande est injectée à un second réseau qui doit être une copie du premier (Figure IV.2).

L'idée est de faire tendre l'erreur entre la sortie du réseau qui assure la commande et celle du second réseau ($e = u - v$), vers zéro. Mais ceci n'implique pas nécessairement, que l'erreur de sortie $\varepsilon = y_d - y$, s'annule. En effet, lors de l'apprentissage, le réseau peut fournir le même signal de commande pour des sorties désirées différentes. Un tel apprentissage peut conduire à une solution triviale indésirable, un réseau à sortie identiquement nulle.

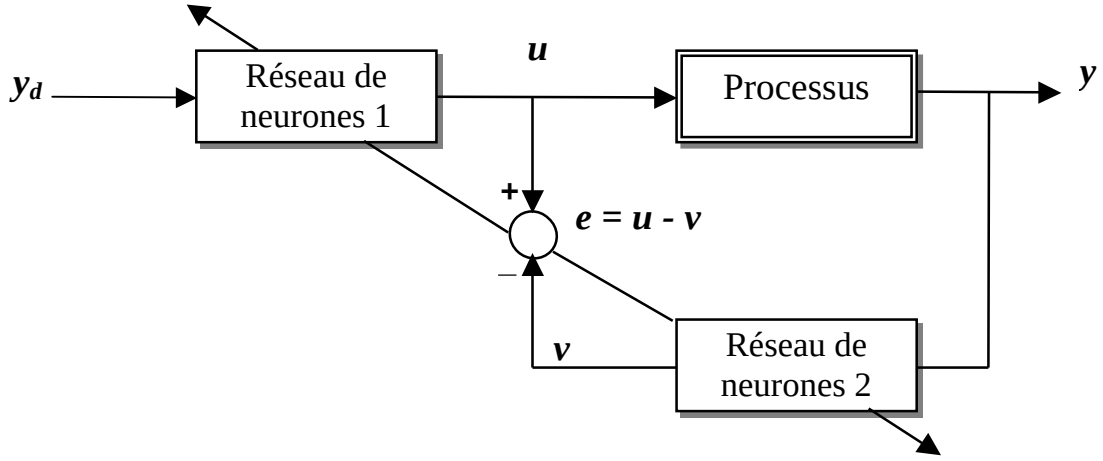


Figure IV.2 : Schéma de principe de l'apprentissage indirecte

IV.4.1.2. Apprentissage général

Dans l'apprentissage général, on utilise les signaux de commande que l'on injecte au système. Les sorties du système constituent les entrées (Figure IV.3). Contrairement au cas précédent, dans cette stratégie on peut choisir les signaux de commande qui constituent les entrées du système.

Le problème cette fois ci est que nous ne disposons pas d'informations sur les sorties que ces dernières peuvent produire.

En effet, les entrées choisies pendant l'apprentissage peuvent ne pas du tout convenir aux objectifs recherchés en sortie. Cet entraînement peut, d'autre part, sortir de la zone d'intérêt que l'on recherche, à savoir même de la zone de fonctionnement du système (l'entraînement se fait hors ligne). En outre, comme dans le cas précédent, l'existence et l'unicité d'un processus inverse n'est pas évidente notamment pour les systèmes dynamiques.

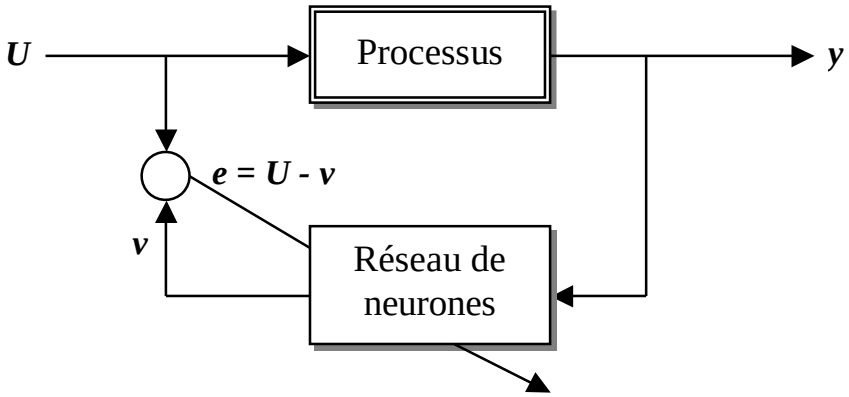


Figure IV.3 : Schéma de principe de l'apprentissage général

IV.4.1.3. Apprentissage spécialisé

Dans cette approche, on remédie aux problèmes rencontrés dans les deux techniques précédentes. Les sorties désirées sont choisies, et l'erreur à minimiser est celle à la sortie du processus. Un réseau de neurones est entraîné pour fournir les signaux de commande nécessaires pour ramener les sorties du système aux valeurs désirées (Figure IV.4).

Cette structure, qui n'est qu'un cas particulier de la commande à apprentissage spécialisé, constitue l'approche la plus efficace. Néanmoins, si les principaux inconvénients de l'entraînement générale sont évités, l'apprentissage du réseau pose un problème, que nous examinerons dans le cadre de la commande spécialisée.

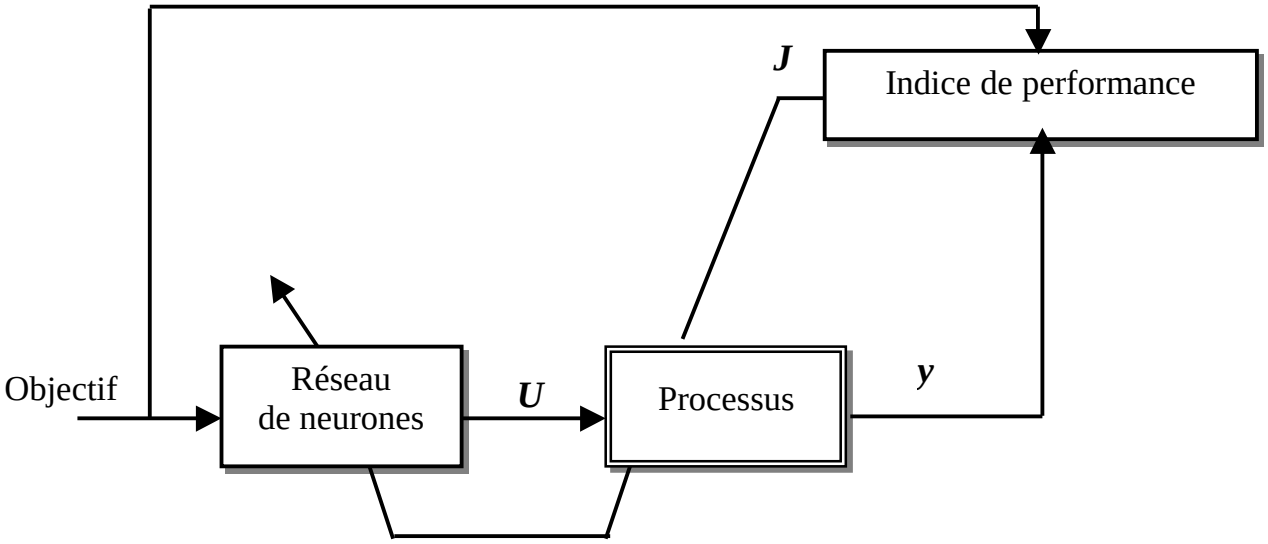


Figure IV.4 : Schéma de principe de l'apprentissage spécialisé

IV.4.2. Commande par identification neuronale directe

Cette structure est très générale, et peut être souvent utilisée. Un modèle suffisamment précis du processus (modèle de prédiction) doit être établi, afin de l'utiliser pour la construction d'une stratégie de commande. Un réseau de neurones est entraîné, pour modéliser le système avec suffisamment de précision par les procédures d'identification neuronales déjà étudiées dans le chapitre III. Les sorties de ce réseau sont utilisées pour l'optimisation d'un critère, sur la base duquel la commande est calculée. Cette approche englobe plusieurs types de commande, et est largement utilisée notamment en commande adaptative (Eki et al., 1998).

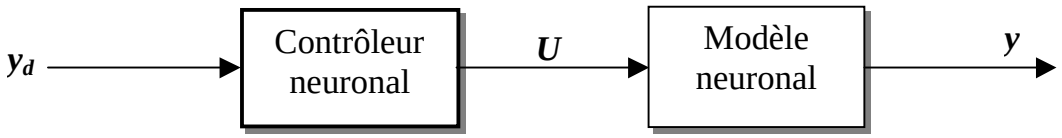


Figure IV.5 : Commande par identification neuronale directe

IV.4.3. Commande avec apprentissage spécialisé

Dans cette approche, un réseau de neurones commande le processus directement, et utilise l'erreur à la sortie du système (ou, en général, un indice de performance), pour ajuster ses poids en ligne (Figure IV.6).

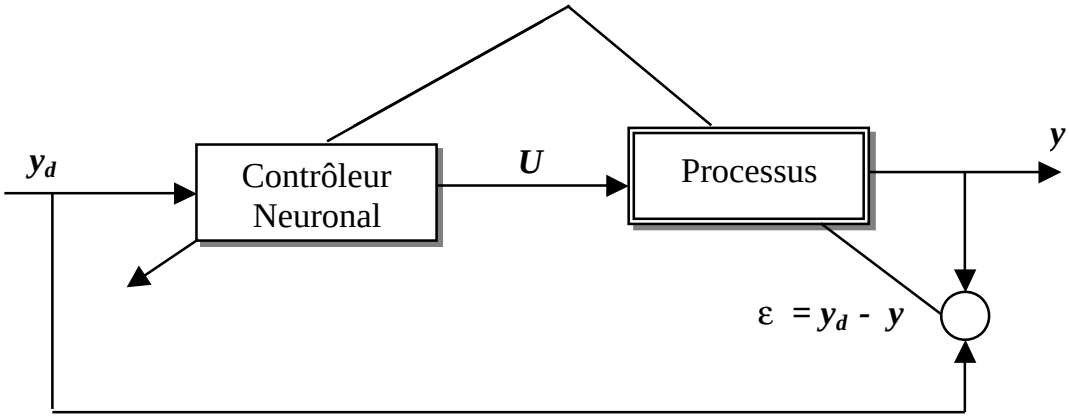


Figure IV.6 : Commande avec apprentissage spécialisé

Cette méthode, qui est très proche de la commande adaptative traditionnelle, est très intéressante, notamment si l'on dispose d'assez de connaissances sur le système à commander. Cependant, le problème qui est inhérent à l'apprentissage des réseaux de neurones, l'entraînement du neuro-contrôleur peut poser de sérieuses difficultés. En effet, l'apprentissage des régulateurs à réseaux de neurones nécessite, en général, la rétro-propagation du Jacobien du critère à minimiser, de la sortie à travers les différentes couches du réseau. Ceci nécessite des connaissances, a priori, sur le système à commander, pour pouvoir calculer correctement les modifications à apporter aux poids synaptiques (Linden et Kindermann, 1989).

IV.4.4. Commande par anticipation (feedforward)

L'utilisation des modèles inverse pour la commande en boucle fermée entraîne généralement un retard indésirable dans la loi de commande. Si un contrôleur classique (un PID, par exemple) est déjà utilisé pour la stabilisation du processus, le modèle inverse peut être utilisé comme un contrôleur par anticipation (Sorensen, 1994) dans un but d'optimisation des performances de la régulation.

En effet, La sortie issue du modèle neuronal inverse est élaborée à partir du signal de référence, ce qui a pour effet l'amélioration des performances du processus (la boucle de retour élimine l'effet des erreurs de modélisation tout en stabilisant le processus et la transmission directe (feed-forward) élimine le retard tout en assurant une robustesse du processus par rapport aux perturbations extérieures). Le schéma de cette implémentation est donné dans la Figure IV.7.

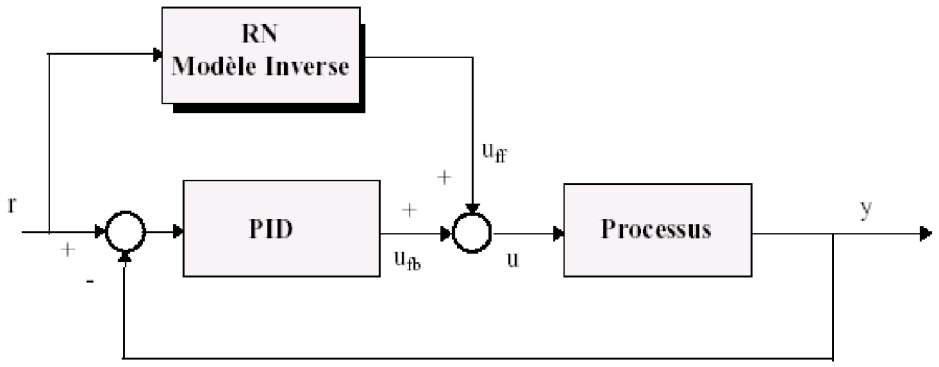


Figure IV.7 : Schéma d'une commande neuronale par anticipation

IV.4.5. Commande Hessienne (Moussaoui et al., 2000)

La stratégie de commande « en ligne » proposée dans ce paragraphe est basée sur l'utilisation de la dérivée seconde de la fonction coût (critère de performance) donnée dans l'équation (IV.1), d'où sa nomination commande Hessienne.

$$J_H = \frac{1}{2}\left\{(Y_{NN}(t+1) - Y_d(t+1))^2 + \rho u^2(t)\right\} \qquad \text{(IV.1)}$$

où $Y_{NN}(t+1)$ est la variable de sortie prédite par le réseau de neurone, $Y_d(t+1)$ est la variable de sortie désirée du processus et ρ est un coefficient de pondération permettant d'assurer un compromis entre la précision de la loi de commande et la valeur l'énergie consommée.

Cette loi de commande est une commande par anticipation (feed-forward). Elle est caractérisée par une estimation rapide (non itérative) de la commande en utilisant seulement le modèle neuronal de prédiction du processus.

Pour élaborer la loi de commande, on effectue une approximation de la valeur de $u(t)$ dans la fonction coût J_H, que l'on note $U(t)$ satisfaisant $\partial Y_{NN}(t+1)/\partial u(t)=0$.

La valeur estimée de cette approximation est notée $U^*(t)$ et l'écart $U(t)$-$U^*(t)$ est égal à une petite valeur notée ε.

Le développement de Taylor tronqué de la dérivée de la fonction coût par rapport à la commande $u(t)$ pour une valeur connue de $U(t)$, donne :

$$\left.\frac{\partial J_H}{\partial u(t)}\right|_{U(t)} = \left.\frac{\partial J_H}{\partial u(t)}\right|_{U^*(t)} + \varepsilon \left.\frac{\partial^2 J_H}{\partial u^2(t)}\right|_{U^*(t)} \qquad \text{(IV.2)}$$

La fonction coût J_H est minimale pour l'expression (IV.2) égale à zéro. De ce fait, l'écart entre la valeur approchée de la commande à l'instant t et sa valeur estimée peut être exprimée par :

$$\varepsilon = -\frac{\left.\frac{\partial J_H}{\partial u(t)}\right|_{U^*(t)}}{\left.\frac{\partial^2 J_H}{\partial u^2(t)}\right|_{U^*(t)}} \qquad \text{(IV.3)}$$

D'où l'on peut obtenir la loi de commande comme suit:

$$U(t) = U^*(t) + \varepsilon = U^*(t) - \frac{\left.\frac{\partial J_H}{\partial u(t)}\right|_{U^*(t)}}{\left.\frac{\partial^2 J_H}{\partial u^2(t)}\right|_{U^*(t)}} \qquad \text{(IV.4)}$$

où

$$\frac{\partial J_H}{\partial u(t)} = \left(Y_{NN}(t+1) - Y_d(t+1)\right)\frac{\partial Y_{NN}(t+1)}{\partial u(t)} + \rho u(t) \qquad \text{(IV.5)}$$

et

$$\frac{\partial^2 J_H}{\partial u^2(t)} = \left(Y_{NN}(t+1) - Y_d(t+1)\right)\frac{\partial^2 Y_{NN}(t+1)}{\partial u^2(t)} + \left(\frac{\partial Y_{NN}(t+1)}{\partial u(t)}\right)^2 + \rho \qquad \text{(IV.6)}$$

A chaque période d'échantillonnage, la valeur de la commande $U(t)$ est affectée à la valeur estimée $U^*(t)$, d'où l'on peut donc assurer une amélioration de la précision de la loi de commande après un certain nombre de périodes d'échantillonnage.

Les expressions des quantités $\partial Y_{NN}(t+1)/\partial u(t)$ et $\partial^2 Y_{NN}(t+1)/\partial u^2(t)$ peuvent être obtenues en utilisant la structure du modèle (multi-entrées / mono-sortie) du prédicteur neuronal à un pas du processus représenté dans la Figure IV.8.

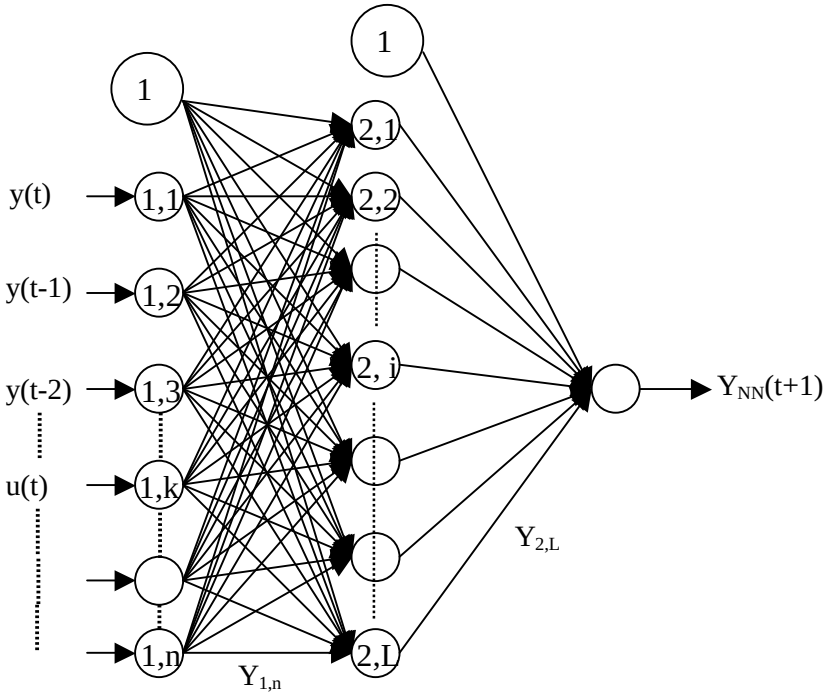


Figure IV. 8 : Structure d'un modèle neuronal à un pas de prédiction

En supposant que la loi de commande u(t) est appliquée à la k^{ème} entrée du modèle neuronal montré dans la Figure VI.8, la quantité $\partial Y_{NN}(t+1)/\partial u(t)$ peut être exprimée comme suit :

$$\frac{\partial Y_{NN}(t+1)}{\partial u(t)} = \sum_i \frac{\partial Y_{NN}(t+1)}{\partial Y_{2,i}(t)} \frac{\partial Y_{2,i}(t)}{\partial u(t)} = \sum_i f'(Net_3) W_i f'(Net_{2,i}) W_{i,k}$$

$$= f'(Net_3) \sum_i W_i f'(Net_{2,i}) W_{i,k} \qquad \text{(IV.7)}$$

La dérivée seconde de *Y(t)* par rapport à *u(t)* est alors :

$$\frac{\partial^2 Y_{NN}(t+1)}{\partial u^2(t)} = f''(Net_3) \sum_i W_i f'(Net_{2,i}) (W_{i,k})^2 + f'(Net_3) \sum_i W_i f''(Net_{2,i}) (W_{i,k})^2 \qquad \text{(IV.8)}$$

où

$$Net_{2,i} = \sum_k Y_{1,k} W_{i,k} \qquad \text{(IV.9)}$$

$$Net_3 = \sum_i Y_{2,i} W_i \qquad \text{(IV.10)}$$

avec:

- Ø W_i est le poids entre le neurone de la couche de sortie et le i^{ème} neurone de la couche cachée ;
- Ø $W_{i,k}$ est le poids entre le i^{ème} neurone de la couche cachée et le k^{ème} neurone de la couche d'entrée ;
- Ø $Y_{1,k}$ est la sortie du k^{ème} neurone de la couche d'entrée ;
- Ø $Y_{2,i}$ est la sortie du i^{ème} neurone de la couche cachée.

Pour une fonction d'activation sigmoïde unipolaire *f(,)* (cf. Figure III.2.c du chapitre III), les dérivées première et seconde peuvent être élaborées en fonction de la fonction d'activation elle-même comme suit :

$$\begin{aligned} &* \quad f' = f(1-f) \quad , \\ &* \quad f'' = f(1-f)(1-2f) \end{aligned} \qquad \text{(IV.11)}$$

Il est à signaler que l'apprentissage du modèle neuronal de prédiction s'effectue « hors-ligne » à l'aide d'un algorithme d'apprentissage du 2ème ordre (quasi-Newtonien). L'implémentation de cette loi de commande est réalisée selon les étapes suivantes :

1) Acquisition des données (Entrées/Sorties) du processus ;
2) Choix d'une structure du modèle neuronal de prédiction ;
3) Apprentissage « hors ligne » du modèle ;
4) Elaboration de la loi de commande « en ligne » en utilisant les équations de (IV.4) à (IV.11).

Cette stratégie de commande peut être implémentée dans un environnement industriel en temps réel selon le schéma synoptique de la Figure IV.9.

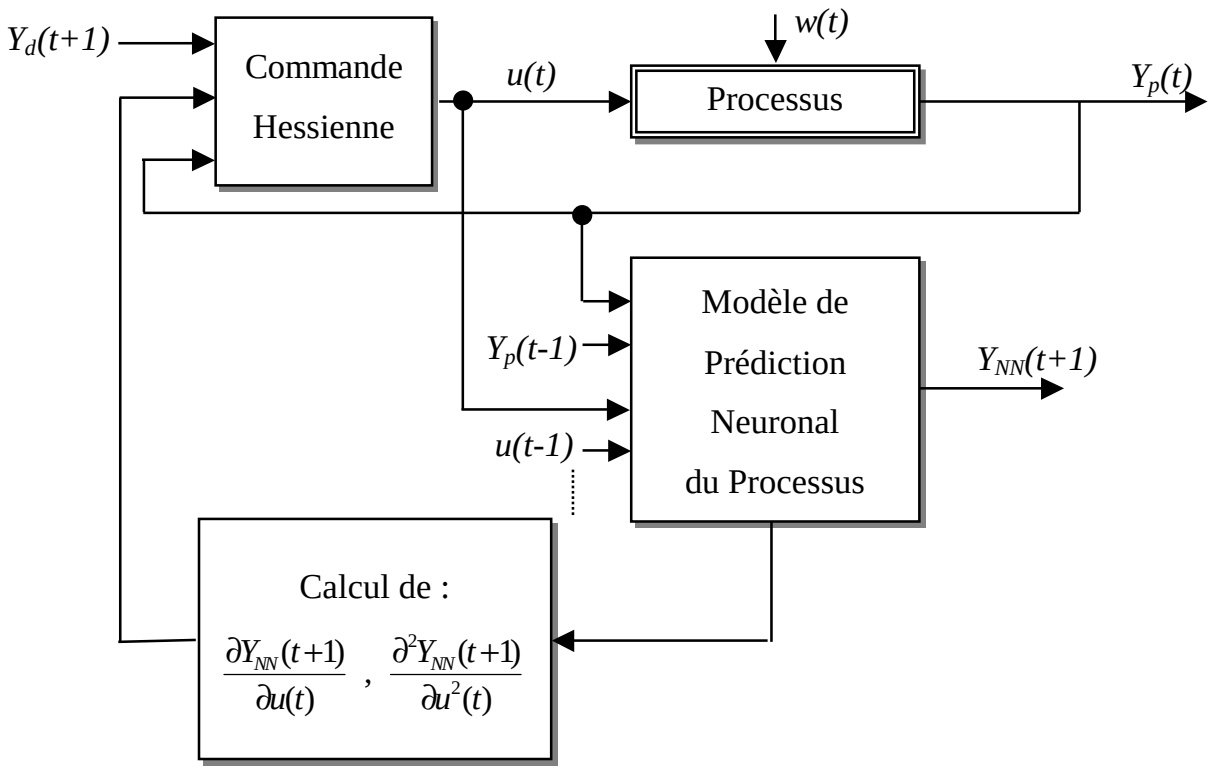


Figure IV.9 : Schéma synoptique de la commande Neuronale Hessienne

IV.4.5.1. Choix du coefficient de pondération ρ

Le choix du coefficient de pondération n'est jamais simple ; il est généralement déterminé empiriquement mais pour en faire une évaluation raisonnable, il est toujours possible de se fonder sur un calibrage physique : si le cahier des charges spécifie les écarts admissibles pour la variable de sortie et une amplitude maximale de la commande, on peut fixer la valeur du coefficient de pondération ρ à l'inverse du carré de la commande maximale (Astrôm et Wittenmark, 1989), soit :

$$\rho = \frac{1}{u_{\max}^2} \qquad \text{(IV.12)}$$

Sachant que le choix du coefficient de pondération ρ a une influence directe sur la minimisation des pics transitoires de la loi de commande.

Notons enfin qu'avec un régulateur neuronal, il est aisé d'imposer une contrainte sur l'amplitude de la commande en choisissant pour le neurone de sortie du régulateur une *fonction d'activation bornée* à la valeur maximale de la commande (tangente hyperbolique, saturation).

IV.4.6. Variante adaptative de la loi commande Hessienne

Afin d'éliminer l'effet des erreurs de modélisation qu'il est généralement impossible d'éliminer dans la plupart des stratégies de commande en boucle ouverte, on introduit dans la loi de commande Hessienne une étape d'adaptation des paramètres du modèle neuronal.

La stratégie de commande développée dans le paragraphe précédent est modifiée (Moussaoui et al., 2003) pour permettre un ajustement en ligne des poids du modèle neuronal à chaque période d'échantillonnage moyennant une adaptation des poids selon un mode d'apprentissage incrémental (by pattern).

Dans ce mode, les poids du modèle de prédiction sont ajustés selon l'équation de mise à jour suivante :

$$W(t+1) = W(t) - \alpha \frac{\partial E(t)}{\partial W(t)}, \qquad \text{(IV.13)}$$

où :

W(t) est le vecteur des poids du réseau de neurones (modèle neuronal), *E(t)* est l'erreur de poursuite, $\frac{\partial E(t)}{\partial W(t)}$ est le gradient de l'erreur , α est un gain d'adaptation, et *t* est la variable temps.

Notons que la procédure d'adaptation des paramètres du modèle donnée par l'équation (VI.13) est utilisée essentiellement pour éliminer l'effet des erreurs d'apprentissage et des perturbations externes.

IV.5. Exemples de commande de processus par RNA

Le but de ce paragraphe est d'illustrer la mise en œuvre et les propriétés des divers stratégies de commande présentées dans ce chapitre.

Et afin d'illustrer aussi complètement que possible le cadre général proposé, nous avons choisi d'étudier deux processus dynamiques simulés, l'un linéaire et l'autre non linéaire :

- Ø Le premier processus, de type entrée-sortie, linéaire discret stable en boucle ouverte est simulé par une équation aux différence récurrente ;
- Ø Le second est un processus non linéaire réel (un réacteur chimique à agitation continue *CSTR*). La synthèse du contrôleur neuronal du réacteur chimique à agitation continue est élaborée après avoir effectué une simulation en boucle ouverte en utilisant son modèle de connaissance non linéaire associé.

Dans les deux exemples, Les modèles utilisés pour l'apprentissage sont des modèles de simulation du processus non perturbé : les correcteurs ainsi synthétisés ne tiennent donc pas explicitement compte de la nature des perturbations aléatoires non mesurées qui affectent le processus pendant la phase d'utilisation.

Comme nous l'avons montré, il est nécessaire pour l'identification de tenir compte du caractère éventuellement bruité du processus ; mais si les résultats obtenus à l'issue de l'identification mettent en évidence que la composante aléatoire de l'erreur d'identification est négligeable devant sa composante déterministe, ce qui est souvent le cas, il est préférable d'utiliser des méthodes de commande axées sur la robustesse des propriétés (performance et stabilité) du système de commande par rapport à des défauts de modélisation et des perturbations déterministes. Ce qui va être considéré dans l'application industrielle du chapitre suivant.

IV.5.1. Exemple de commande d'un processus linéaire par modèle inverse

L'exemple illustratif que nous allons aborder est un processus linéaire discret du type entrées/sorties. En particulier, nous allons utiliser les réseaux de neurones pour la synthèse d'un contrôleur non linéaire neuronal en boucle ouverte.

Une procédure d'adaptation des gains du contrôleur est prévue moyennant une modification de l'architecture du schéma de commande pour éliminer l'effet des erreurs d'apprentissage du contrôleur.

Soit le processus dynamique linéaire discret représenté par l'équation aux différences récurrente (Marie et Mokhtari, 1998) :

$$y(k) = 0.7\, y(k-1) + 0.3\, u(k-1) + 0.05\, u(k-2) \qquad \text{(IV.14)}$$

Le modèle inverse neuronal du processus considéré est un réseau de neurones qui permet de calculer l'entrée *u(t)*, de l'instant discret $t = kT$, (où T est la période d'échantillonnage), à partir des valeurs antérieures de ces entrées-sorties.

L'élaboration du modèle inverse nécessite une simulation en boucle ouverte du processus exprimé par (IV.14). Pour que les différents modes du processus soient excités, on appliquera un signal riche en fréquences : une *SPBA* (Séquence Binaire Pseudo-Aléatoire) autour du point de fonctionnement désiré.

Le signal de sortie du système est élaboré moyennant l'application d'un signal de commande *u(t)*, formé d'une séquence binaire pseudo-aléatoire d'amplitude 1 que l'on superpose à une valeur constante égale à 5, correspondant à un point de fonctionnement du processus. Les caractéristiques entrées/sorties du système sont données dans les courbes de la Figure IV.10.

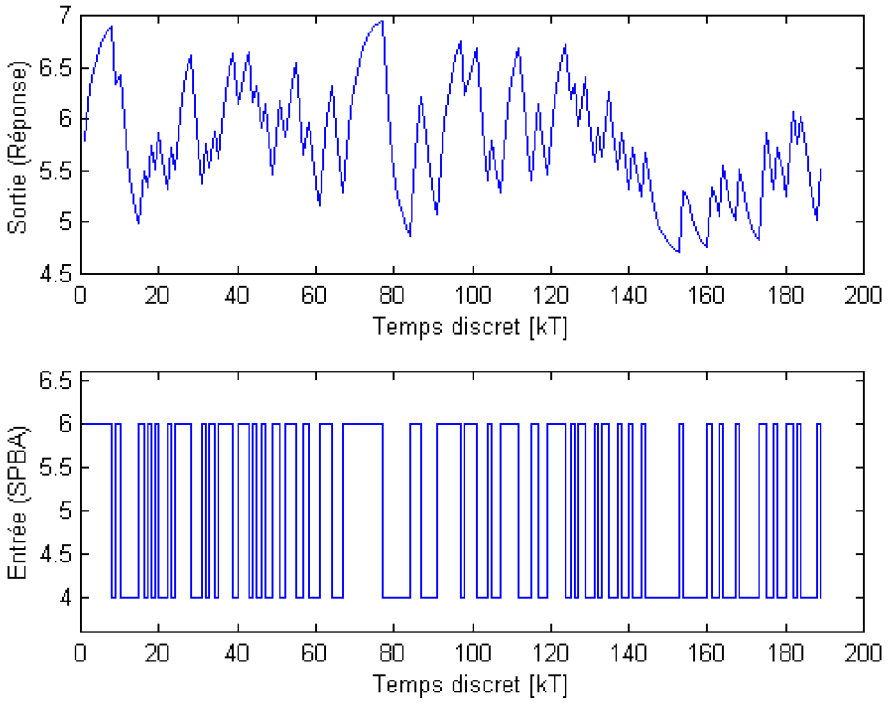


Figure IV.10 : Caractéristiques entrées/sorties du processus

La structure proposée pour le modèle inverse est montrée dans la Figure IV.11. Dans l'architecture proposée, on cherche à obtenir directement le signal de commande *u(t),* en fonction d'un ensemble de variables entrées/sorties appliquées au réseau. Le caractère prédictif de la loi de commande (prédiction à un pas) est illustré par la consigne future qui sera appliquée à la première cellule de la couche d'entrée.

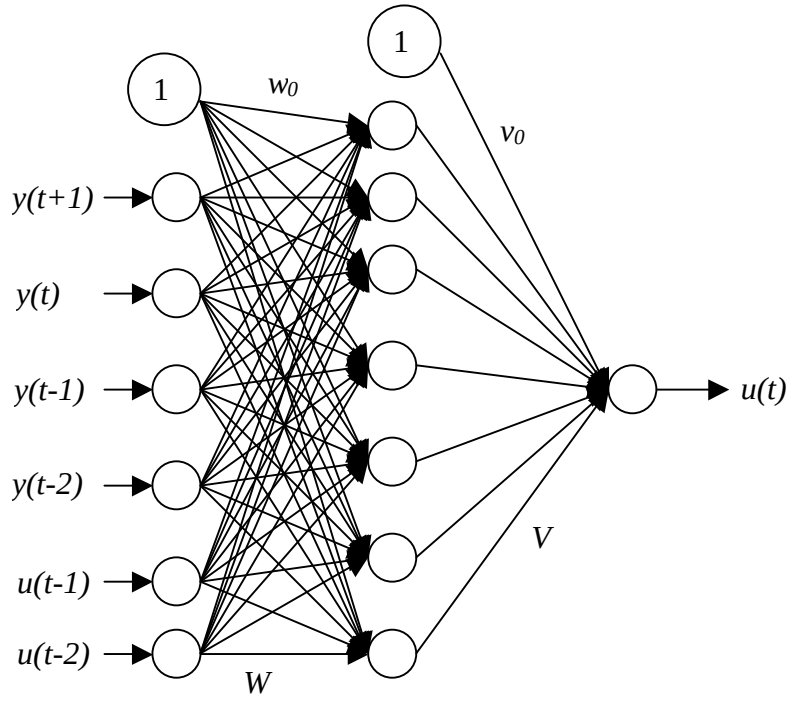


Figure IV.11 : Architecture du modèle inverse neuronal

L'apprentissage du modèle inverse neuronal est effectué « hors ligne » en présentant sous forme de paquet les données entrées/sortie du processus *normalisées* entre 0.1 et 1 pour des commodités de calcul. On effectue ainsi un « *batch learning* ».

Comme les algorithmes de rétro-propagation de l'erreur standards basés sur la descente du gradient sont lents (cf. Chapitre III), on a choisi d'effectuer un apprentissage moyennant l'algorithme d'optimisation de Levenberg-Marquardt du deuxième ordre, développé au chapitre III, car celui-ci paraît le plus rapide des algorithmes d'apprentissage (Hagan et Menjah, 1994).

Les courbes d'évolution des différents poids et des biais sont représentées dans la Figure IV.12.

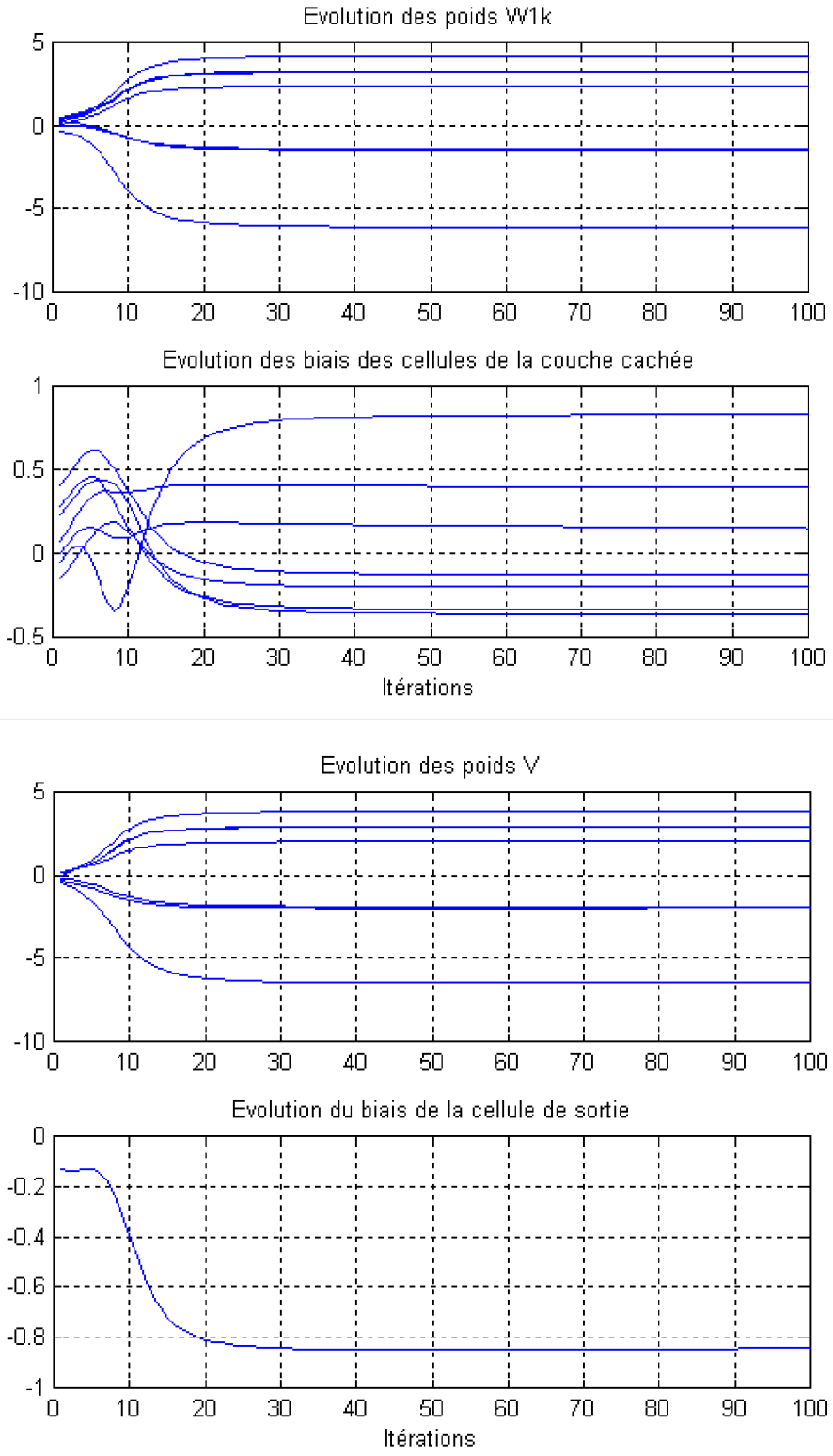


Figure IV.12 : Les courbes d'évolution des différents poids et des biais

On observe bien la convergence des poids et des biais, ce qui démontre la réussite de la phase d'apprentissage du modèle inverse.

Néanmoins, selon la courbe d'évolution de l'erreur de prédiction donnée dans la Figure IV.13, la variance est sensiblement égale à sa valeur maximale en valeur absolue soit ($\approx 10^{-2}$).

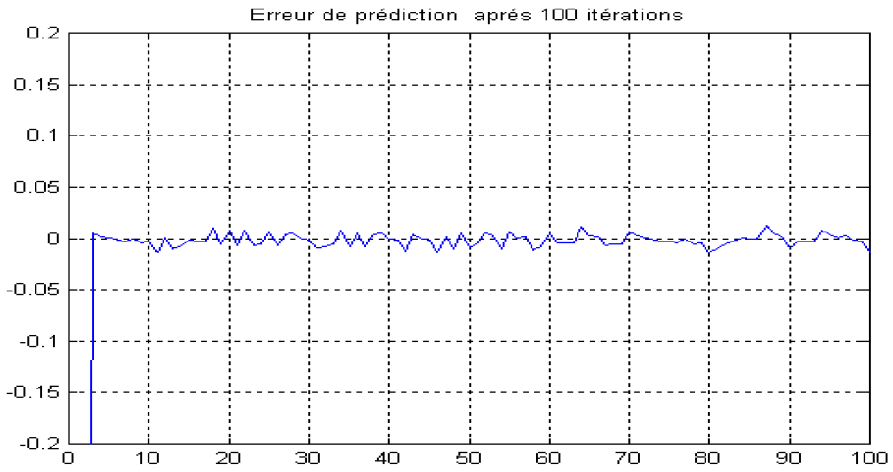


Figure IV.13 : Evolution de l'erreur de prédiction

Il est à noter que cette erreur a lieu sur le signal de commande, ce qui entraînera nécessairement une erreur de poursuite en régime permanent, lors de la mise en œuvre de la loi de commande, du fait que la structure du réseau, ne contient pas d'intégration dans la loi de commande (Weerasooriya et Sharkawi, 1991).

La structure proposée de la commande par modèle inverse neuronal est schématisée dans la Figure IV.14.

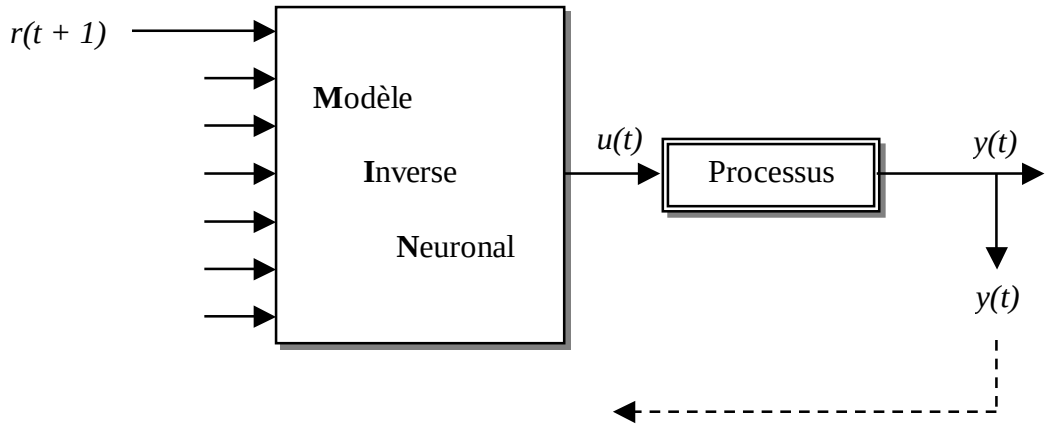


Figure IV.14 : Structure du schéma de commande du processus

Dans cette structure, la première cellule de la couche d'entrée reçoit la valeur future du signal de consigne (valeur de référence), ce qui a pour effet de supprimer le retard d'un pas d'échantillonnage de la sortie sur la consigne.

Pour tester la propriété de poursuite de référence du contrôleur neuronal, on a choisi d'utiliser un signal de référence évoluant par paliers.

A partir des courbes de réponse du processus et du signal de commande données dans la Figure IV.15, la propriété de poursuite est bien mise en évidence et ce malgré l'application d'une perturbation en sortie à un instant donné de la simulation (t=300T). En effet, il est constaté que la perturbation est rejetée avec un léger dépassement.

Notons, que comme il a été prévu auparavant, l'absence d'une action intégrale de la structure du contrôleur a entraîné effectivement une erreur statique en régime établi entre la consigne et la sortie du processus.

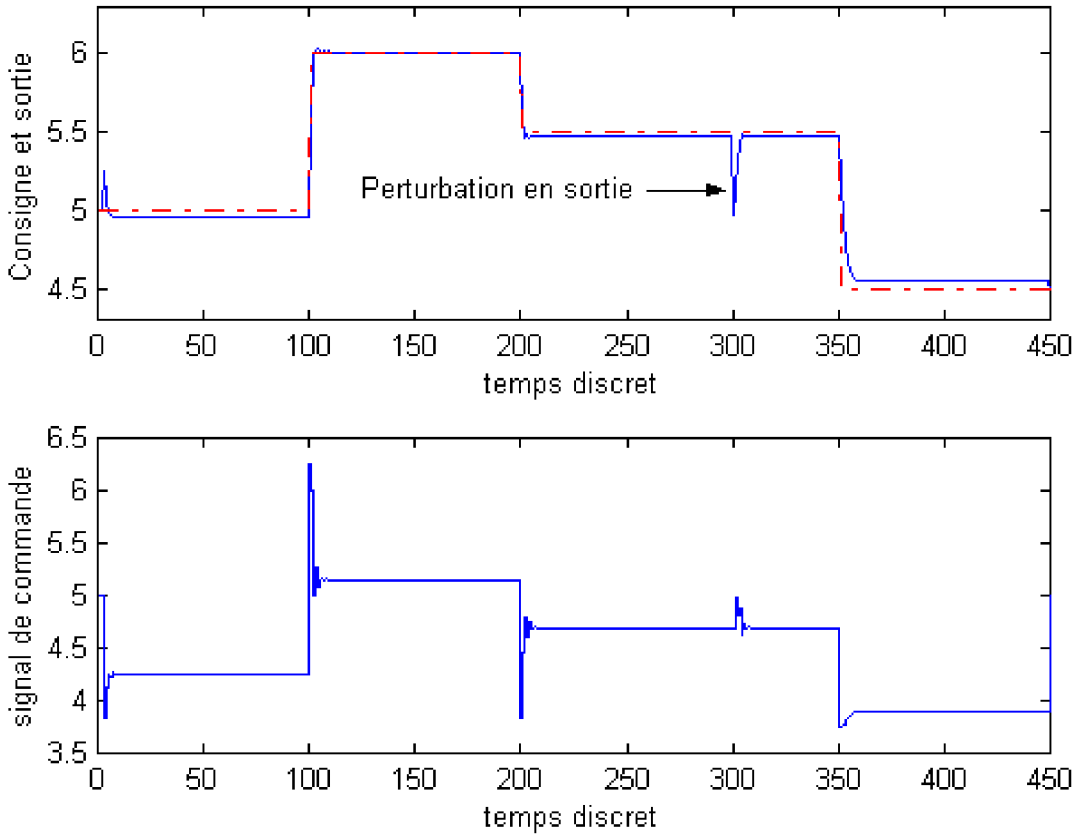


Figure IV.15 : Courbe d'évolution des signaux de sortie et de commande

Afin d'éliminer l'erreur statique en régime établi, on a choisi de rétro-propager l'erreur de poursuite à travers le réseau pour modifier les différents poids et biais.

Ceci revient à effectuer un apprentissage en ligne du réseau à chaque période d'échantillonnage selon l'équation de mise à jour (IV.13). L'apprentissage s'effectue par la présentation au réseau de neurones qu'un seul *exemple* (*pattern*) entrée/sortie à chaque période d'échantillonnage. Cette procédure est appelée, comme on l'a signalé au chapitre III, apprentissage incrémental.

La structure de cette loi de commande (adaptative) par modèle inverse est montrée dans la Figure IV.16.

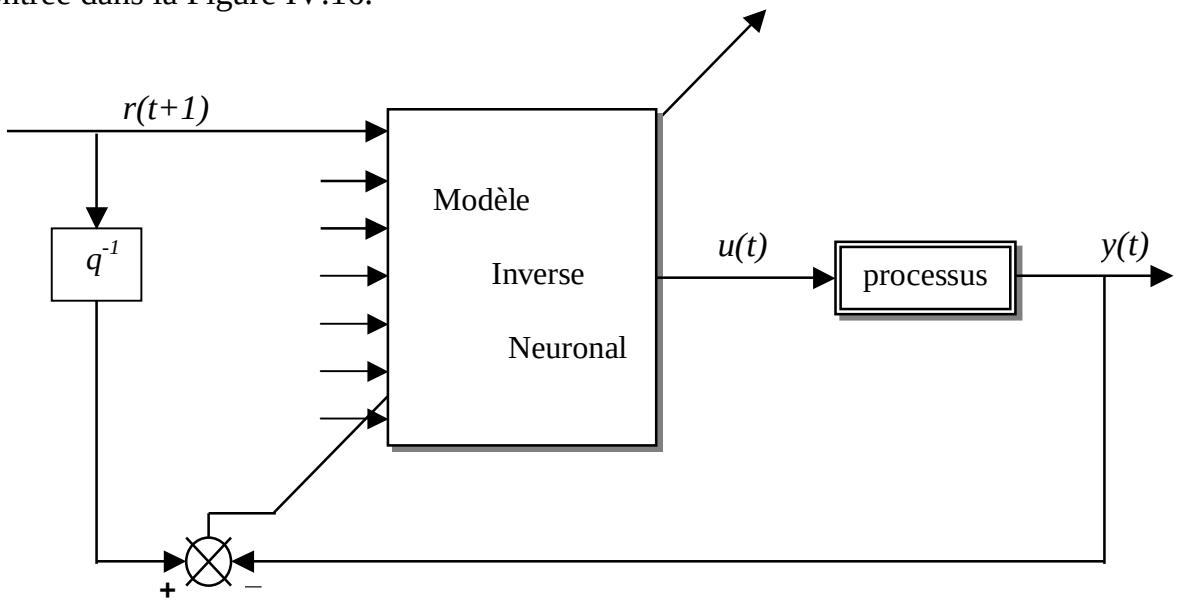


Figure IV.16 : Structure d'une Commande Neuronale Adaptative

Pour réaliser cette loi de commande, les poids et les biais sont initialisés aux valeurs obtenues lors de l'apprentissage.

Pour cette structure de commande, et à partir des courbes de simulation des caractéristiques entrées/sorties ainsi que celles de l'adaptation des poids et des biais, il est montré que l'erreur statique est complètement annulée, mais la perturbation en sortie est encore rejetée avec un léger dépassement (Figure IV.17). Celui-ci peut être diminuer en jouant sur le gain d'adaptation α de l'équation IV.3. D'après les courbes de la Figure IV.18 et de la Figure IV.19, il est constaté que seuls les biais ont subi une modification car l'erreur dans le cas de notre exemple est une erreur statique.

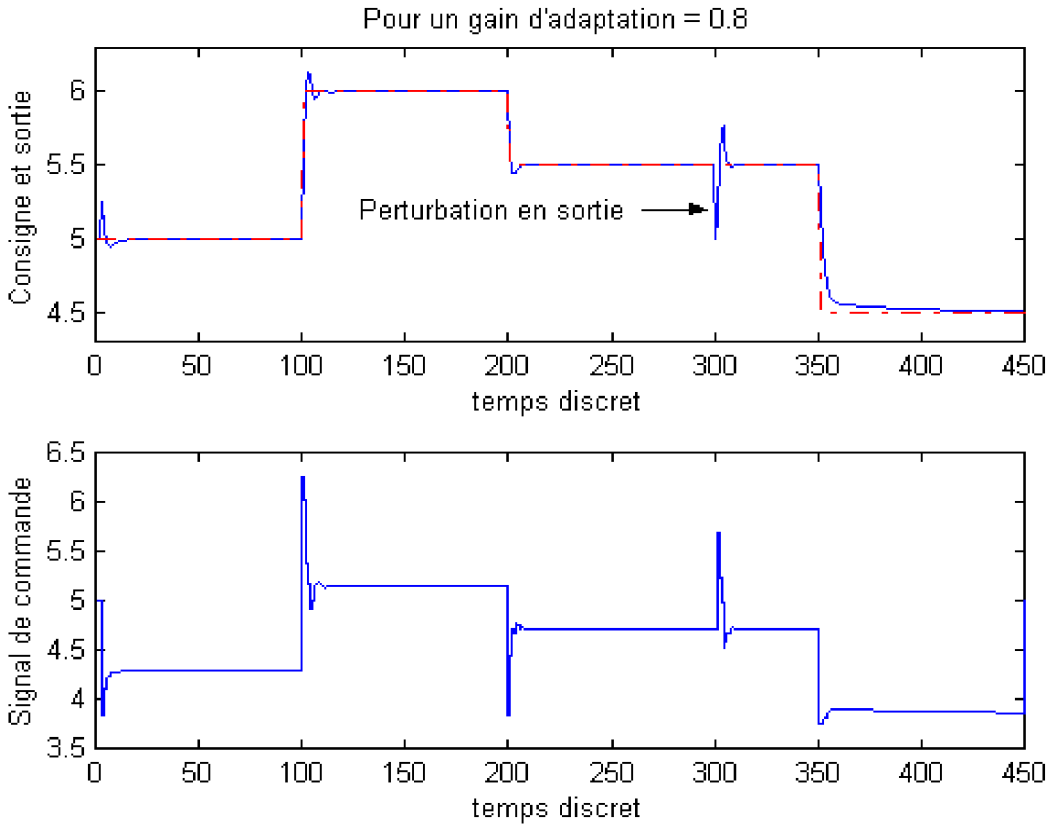


Figure IV.17 : Courbe d'évolution des signaux de sortie et de commande

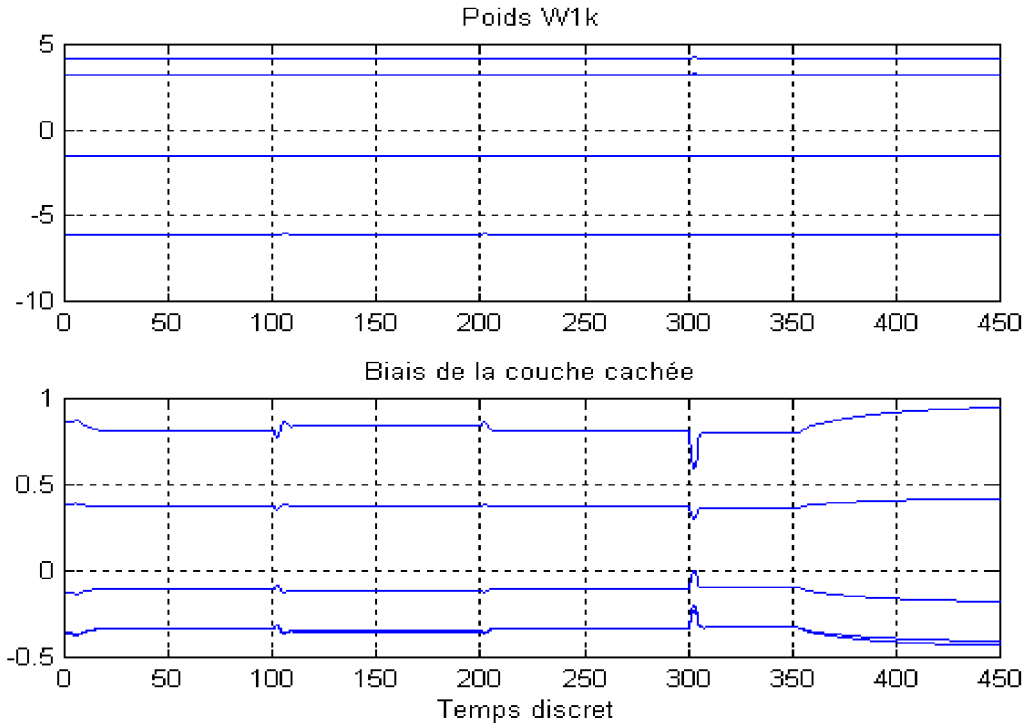


Figure IV.18 : Evolution des poids et des biais de la couche cachée

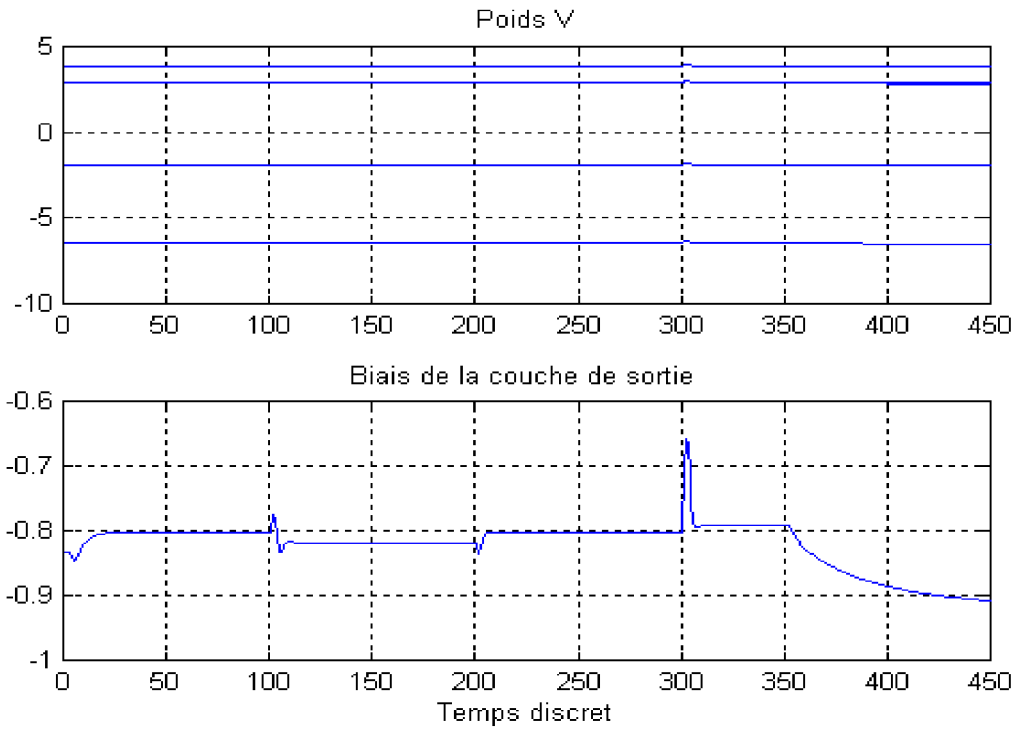


Figure IV.19 : Evolution des poids et des biais de la couche de sortie

IV.5.2. Exemple de commande d'un processus non linéaire par modèle inverse

Afin de mieux estimer les résultats issus de l'application des contrôleurs neuronaux élaborés dans le présent chapitre, un processus non linéaire réel est considéré (réacteur chimique à agitation continue : *CSTR*).

Dans le réacteur s'effectue une réaction chimique exothermique. En d'autres termes, il y a dégagement de la chaleur pendant le déroulement de la réaction ce qui entraîne une augmentation de la température (Huang et al., 1982) ayant pour effet direct le ralentissement de la réaction et la diminution de la concentration du produit.

En introduisant un fluide refroidissant à un débit variable $q_c(t)$, la température peut être modifiée et donc la concentration du produit (Cebuhar et Costanza, 1984).

IV.5.2.1. Comportement du processus en boucle ouverte

Afin de bien se saisir du problème, nous proposons d'étudier le comportement du CSTR en boucle ouverte. Nous avons donc discrétisé le modèle non linéaire du processus avec un pas d'échantillonnage $\Delta t=0.01$ seconde, telle que $t=k\Delta t$.

Selon (Cebuhar et Costanza, 1984), le modèle continu du CSTR est décrit par les équations différentielles non linéaires suivantes:

$$\frac{dC_a(t)}{dt} = \frac{q}{v}(C_{a0} - C_a(t)) - k_0 C_a(t)\, e^{-\left(\frac{E}{RT(t)}\right)}$$
$$\frac{dT(t)}{dt} = \frac{q}{v}(T_0 - T(t)) + k_1 C_a(t)\, e^{-\left(\frac{E}{RT(t)}\right)} + k_2 q_c(t)\left(1 - e^{-\left(\frac{k_3}{q_c(t)}\right)}\right)(T_0 - T(t)) \qquad \text{(IV.15)}$$

Où:

- Ø $T(t)$: Température à l'intérieur du CSTR ;
- Ø $q_c(t)$: Débit du réactif de refroidissement qui représente la commande $u(t)$;
- Ø C_{a0} : Concentration de l'alimentation en entrée ;
- Ø T_0 : Température du produit en entrée ;
- Ø q : Taux d'écoulement du processus ;
- Ø v : Volume du réacteur ;
- Ø k_0 : Constante définissant le taux (la vitesse) de réaction ;
- Ø k_1 , k_2 et k_3 : Constantes caractéristiques du réacteur.

Après la discrétisation du modèle continu décrit par les équations (IV.15), on obtient le modèle échantillonné suivant :

$$C_a(k+1) = C_a(k) + \Delta t\left(\frac{q}{v}(C_{a0} - C_a(k)) - k_0 C_a(k)\, e^{-\left(\frac{E}{RT(k)}\right)}\right)$$
$$T(k+1) = T(k) + \Delta t\left(\frac{q}{v}(T_0 - T(k)) + k_1 C_a(k)\, e^{-\left(\frac{E}{RT(k)}\right)} + k_2 q_c(k)\left(1 - e^{-\left(\frac{k_3}{q_c(k)}\right)}\right)(T_0 - T(k))\right)$$

(IV.16)

Il est effectué une simulation en boucle ouverte du processus décrit par les équations (IV.16), avec comme valeurs initiales : Ca_o=0.2 [mole /l] , T_o=430 [°K].

Avec les valeurs numériques des paramètres du réacteur indiquées dans la Table 1 de l'Annexe (Cebuhar et Costanza, 1984) et à travers les courbes de simulation données dans la Figure IV.20, il est remarqué que dans la réaction chimique, une augmentation de la concentration provoque une augmentation de la température. Cette augmentation de la température est naturellement compensée par une diminution de la concentration. En effet, suivant les lois de la thermodynamique, cette augmentation favorise le déroulement de la réaction dans le sens opposé.

Ainsi, ce sont ces deux faits contradictoires qui font le problème posé de ce processus.

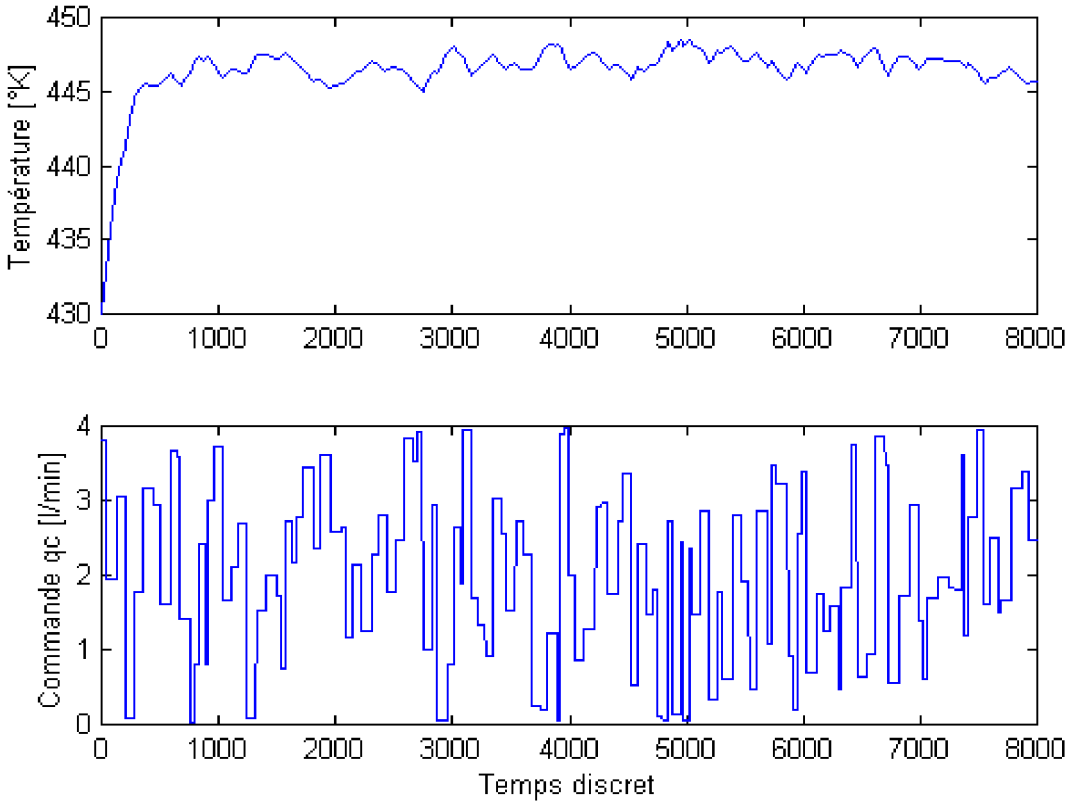


Figure IV.20 : Caractéristiques entrées/sorties du *CSTR*

IV.5.2.2. Synthèse du contrôleur neuronal

En tenant compte du comportement en boucle ouverte du processus, on a choisi comme contrôleur, le contrôleur neuronal donné dans la Figure IV.21. Ce réseau de neurones a pour but de fournir au processus, à chaque instant d'échantillonnage, la variation de la commande $\Delta u(t) = u(t)-u(t-1)$.

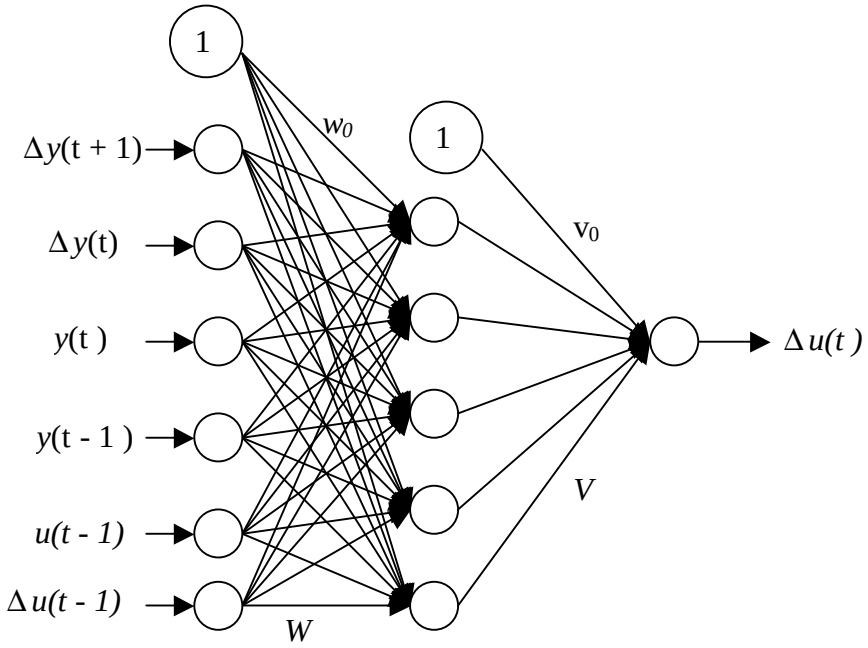


Figure IV.21 : Structure du contrôleur neuronal du *CSTR*

La variation future de la sortie *Δy(t+1)*, permettra, comme on le verra plus loin, de réaliser ainsi une commande prédictive à un pas d'échantillonnage.

L'apprentissage du réseau de la Figure IV.21 est effectué moyennant l'algorithme de Levenberg-Marquardt et un paquet entrées/sorties de 2000 exemples issu de la simulation en boucle ouverte du processus (cf. Figure IV.22).

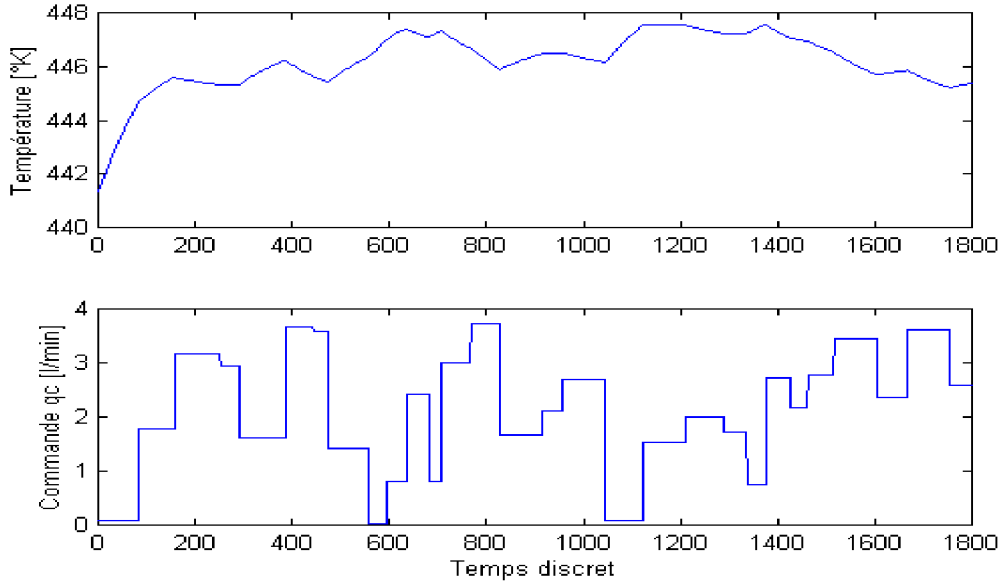


Figure IV.22 : Echantillons d'apprentissage du contrôleur neuronal

Notons que pour des raisons pratiques, les transitoires sont exclus des échantillons d'apprentissage (environ 200 échantillons). Il est montré dans les courbes de la Figure IV.23 la convergence des poids et des biais après 300 à 350 itérations.

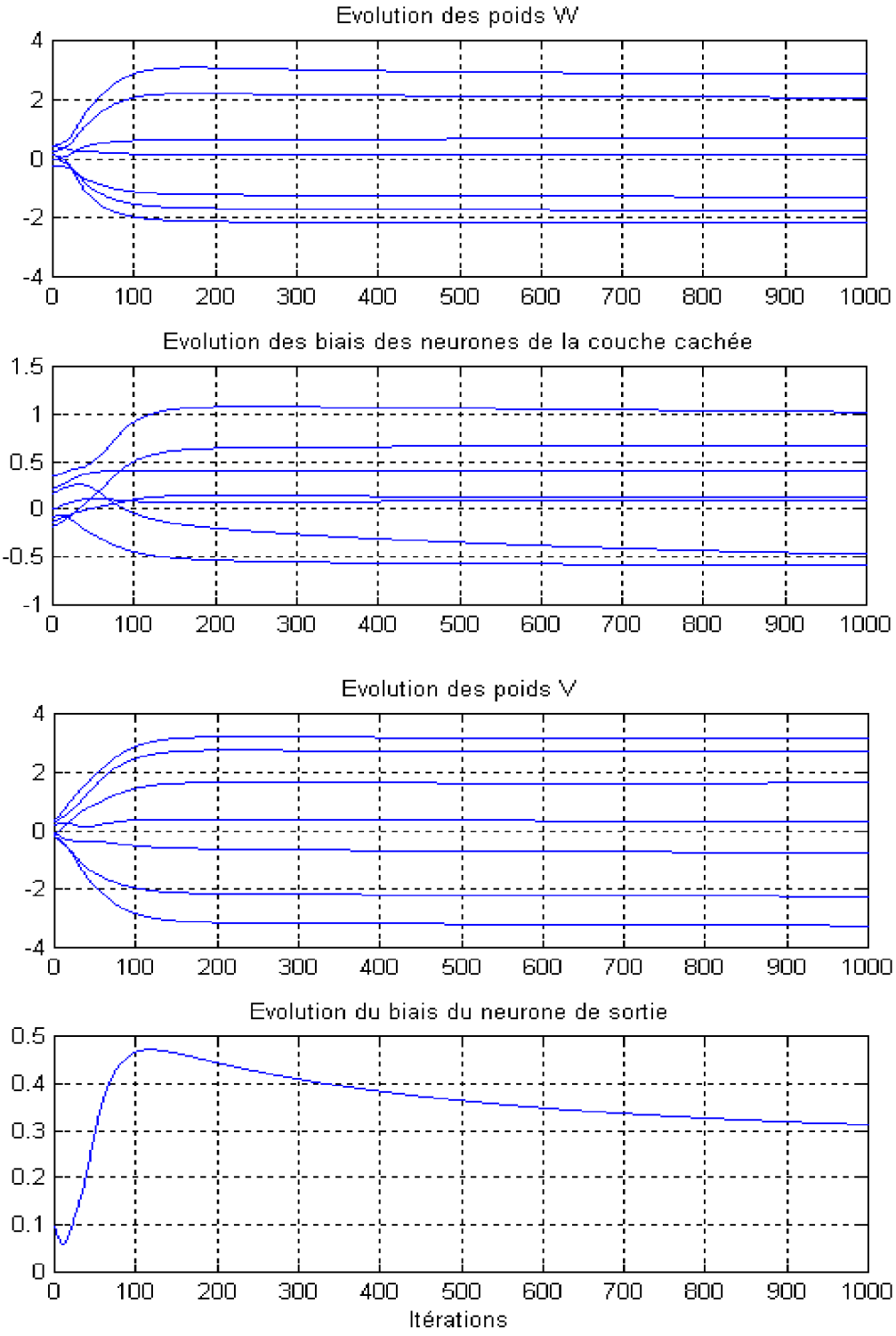


Figure IV.23 : Les courbes d'évolution des différents poids et des biais

La qualité de l'identification du modèle neuronal inverse est mise en évidence à travers la courbe d'évolution de l'erreur d'apprentissage du contrôleur neuronal donnée dans la Figure IV.24.

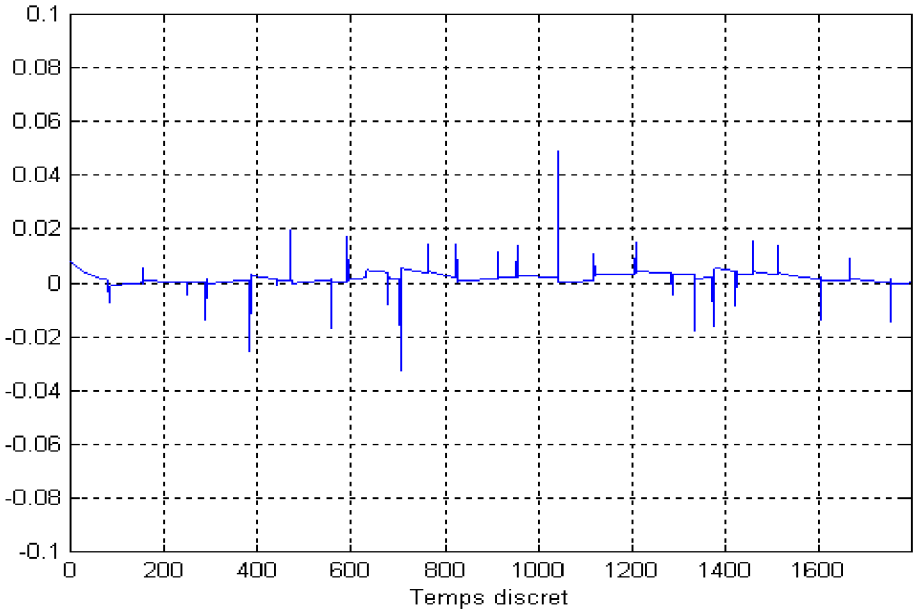


Figure IV.24 : Courbe d'évolution de l'erreur d'apprentissage

L'implémentation de cette loi de commande est mise en œuvre selon le schéma synoptique de la Figure IV.25.

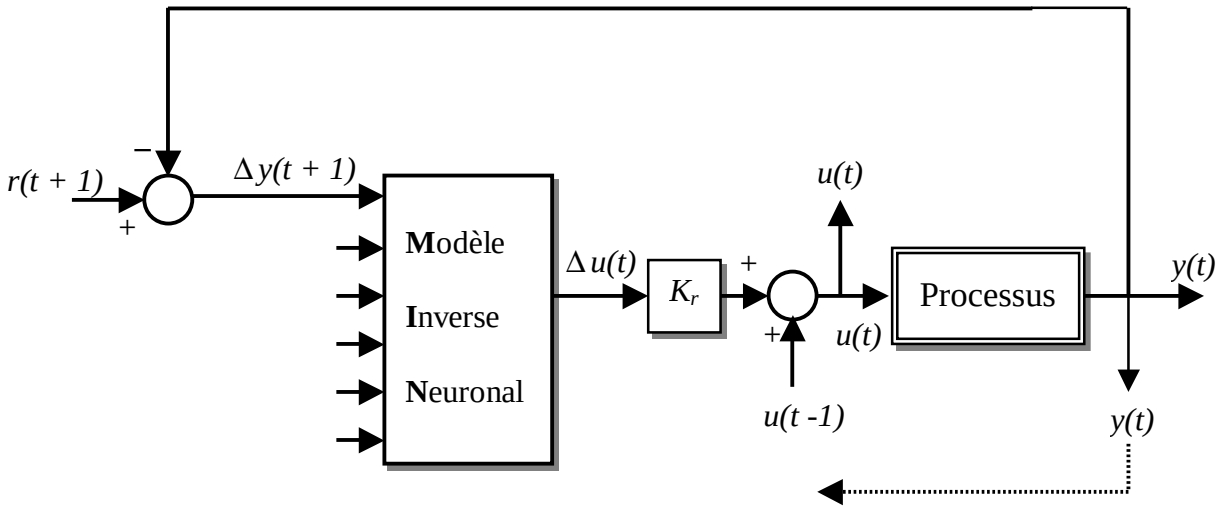


Figure IV.25 : Schéma synoptique de la loi de commande

Le signal issu du neurone de la couche de sortie du réseau, multiplié par un gain K_r associé à l'action intégrale**,** est ajouté à la commande appliquée au *CSTR* à l'instant d'échantillonnage précédent.

Il est constaté à partir des courbes de simulation de la Figure IV.26, la bonne poursuite d'un signal de référence évoluant par paliers, et ce grâce à la présence de l'action intégrale.

Figure IV.26 : Courbes d'évolution de la commande et de la sortie

De plus, et afin de tester la robustesse du contrôleur neuronal, il est effectué une simulation du processus contrôlé, avec l'insertion d'une perturbation à la sortie du processus, à l'instant $t = 1000.\Delta t$, d'une amplitude égale à $y(k)/2$.

Il est constaté dans les courbes de la Figure IV.27, que la perturbation est rejetée avec un léger dépassement en un temps raisonnable, soit après un temps relativement court par rapport à la dynamique du processus.

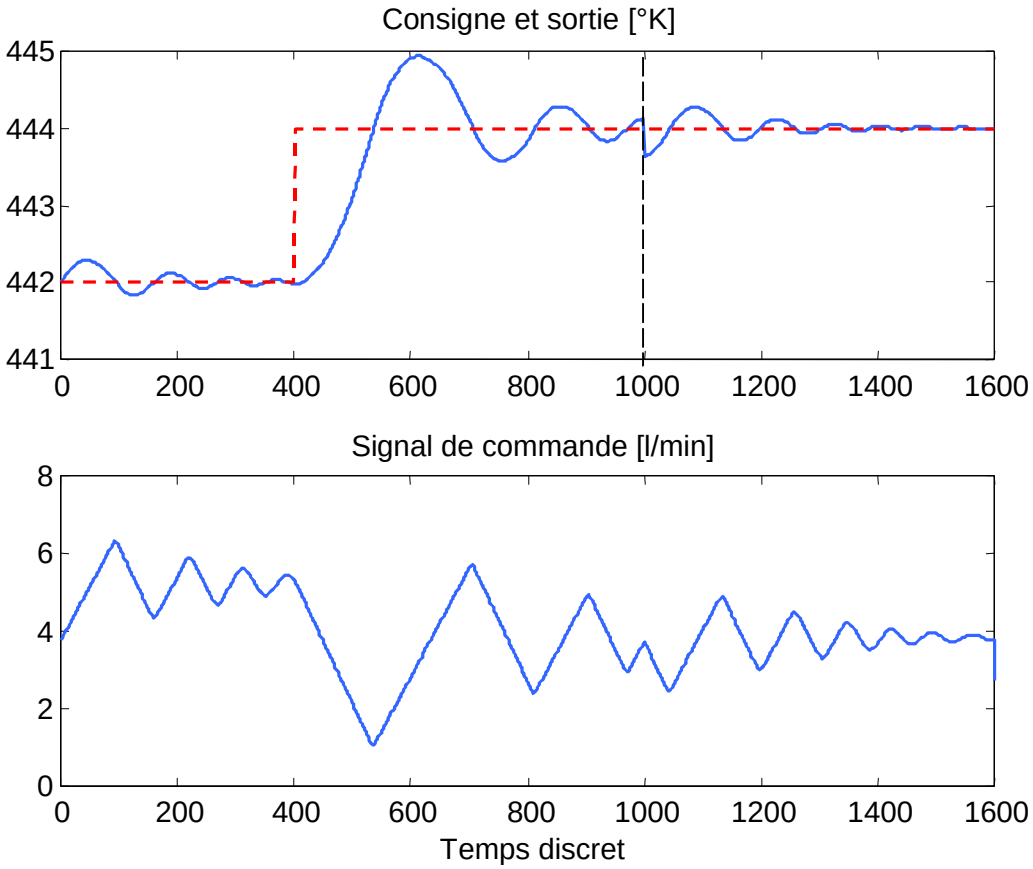


Figure IV.27 : Courbes d'évolution de la commande et de la sortie en présence d'une perturbation à la sortie, à l'instant $t=1000.\Delta t$

Notons enfin, que pour améliorer le comportement du contrôleur neuronal du point de vue rejet de perturbation et réduction des dépassements, un choix judicieux du gain K_r s'impose.

IV.6. Conclusion

Après la description des différentes stratégies de commande basées sur les réseaux de neurones et l'étude des conditions de leur mise en œuvre, les performances de ces stratégies de commande sont vérifiées à travers des exemples illustratifs.

De plus, il est développé dans ce chapitre une nouvelle stratégie de commande rapide basée sur le calcul en ligne de l'Hessien d'un critère de performance quadratique et l'utilisation du modèle de prédiction neuronal du processus.

Il est constaté à travers notre étude qu'il est souvent utile de vérifier les propriétés de robustesse des stratégies de commande neuronales vis-à-vis :

- Ø de perturbations de sortie ;
- Ø de défauts de modélisation ;
- Ø de défauts du correcteur dus à l'apprentissage, problème particulier à l'utilisation de réseaux de neurones ;

Ainsi, et afin d'éliminer l'effet de ces défauts dans le cas de la loi de commande Hessienne proposée dans ce chapitre, il est montré qu'une introduction d'une étape d'adaptation des paramètres du modèle de prédiction ou du contrôleur neuronal est nécessaire (Moussaoui et al., 2004). Dans cette étape, l'ajustement des poids du modèle neuronal ou du contrôleur neuronal est effectué en ligne à chaque période d'échantillonnage selon un mode d'apprentissage incrémental.

Cependant, il faut signaler qu'il est parfois suffisant d'ajouter tout simplement une action intégrale au schéma de contrôle pour améliorer les performances de la réponse du processus du point de vue précision (ce qui est vérifié dans le deuxième exemple illustratif de ce chapitre).

V.1. Introduction

Les trains de laminage à chaud font parties des outils sidérurgiques les plus sophistiqués. Ils transforment un produit épais (*brame*) en une tôle mince. Pour la commercialisation du produit, les problèmes de tolérance dimensionnelle sont d'un intérêt capital. Des systèmes d'automatisation et de régulation de plus en plus efficaces ont été développés au cours des années afin d'obtenir une épaisseur la plus proche possible de l'épaisseur visée en sortie du laminoir (Günter et al., 1996).

Actuellement, les performances demandées sont de plus en plus sévères. Nous avons donc entrepris une étude ayant pour but ultime la conception de nouveaux types de contrôleurs basés sur les réseaux de neurones artificiels.

V.2. Définition du laminage.

Le laminage, opération d'une grande diversité, peut se résumer par la définition suivante :

Opération *de mise en forme* par déformation plastique, destinée à réduire la section d'un produit de grande longueur, par passage entre deux ou plusieurs outils axisymétriques tournant autour de leur axe ; c'est la rotation des outils qui entraîne le produit dans l'emprise des cylindres par l'intermédiaire du frottement (Figure V.1).

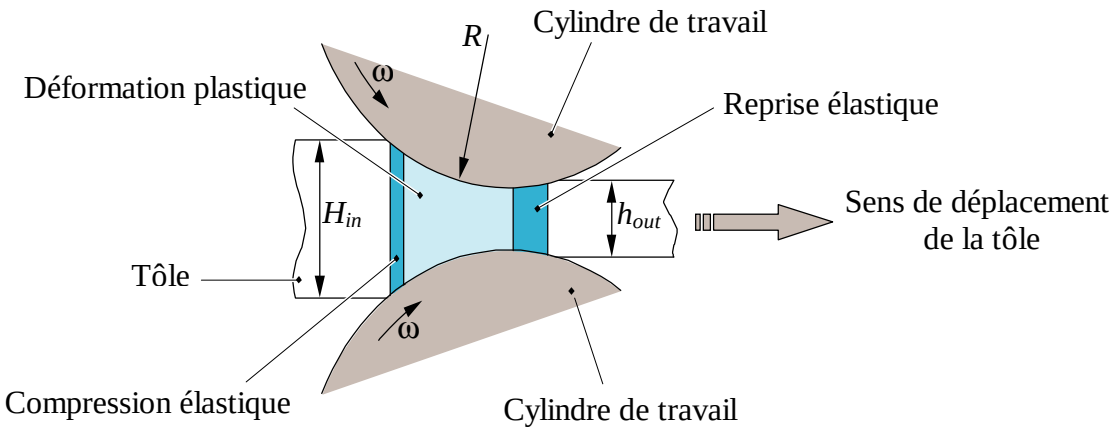


Figure V.1. : Schéma du foyer de laminage

La réduction d'épaisseur est obtenue par déformation plastique du métal entre deux cylindres de rayon *R*, appelés « cylindres de travail » qui sont en rotation inverse.

On appelle réduction le rapport de l'écart d'épaisseur avant et après déformation à l'épaisseur avant déformation :

$$r = \frac{H_{in} - h_{out}}{H_{in}} \qquad \text{(V.1)}$$

Afin de déformer la tôle (bande) de façon permanente, il faut lui appliquer une contrainte minimale, faute de quoi, après relâchement de l'effort, elle revient de façon élastique à son état d'origine.

De plus, la tôle s'écrouit, c'est-à-dire que plus on la déforme, plus elle durcit. Lors du laminage, la tôle subit d'abord une déformation élastique, une déformation plastique lors de la réduction proprement dite et enfin un retour élastique.

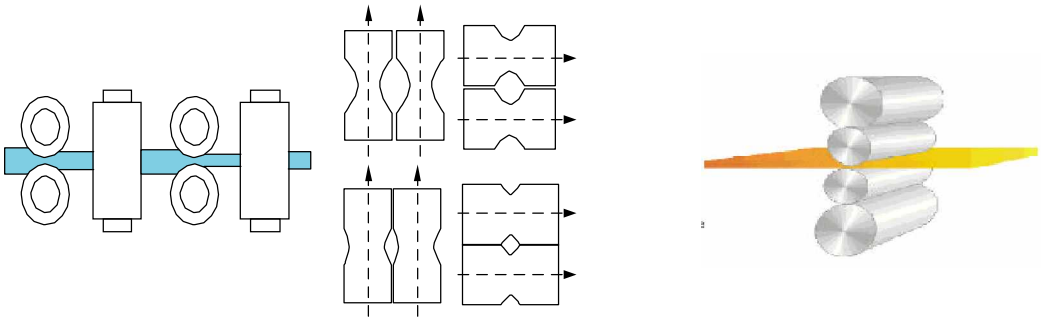
Il est à noter que *mettre en forme* un matériau, c'est lui conférer de manière contrôlée et reproductible trois types de propriétés (Monmitonnet, 2000) :

- Une forme géométrique, avec des tolérances fixées, de plus en plus sévères ;
- Des propriétés mécaniques, qui requièrent une microstructure adéquate ;
- Des propriétés de surface , au premier rang desquelles l'aspect visuel, lié à la rugosité.

V.3. Différents Types de laminages (Généralités)

Opération fondamentale en métallurgie, le laminage voit passer environ 90 % de tout le métal produit, tous métaux et alliages confondus. Le laminage vient après l'élaboration du métal dans l'aciérie, puis la coulée, le plus souvent continue. Il se scinde en deux catégories de laminage :

- laminage de *produits longs* (barres, fils, tubes, poutrelles, rails, ...), où les deux dimensions de la section, du même ordre de grandeur en général, sont petites devant la longueur ; les outils sont le plus souvent des cylindres cannelés (Figure V.2.a) ;
- laminage de *produits plats* (tôles, bandes et feuillards) où l'épaisseur est petite devant la largeur, elle-même très inférieure à la longueur. Les outils sont des objets axisymétriques presque cylindriques (au bombé de rectification près) (Figure V.2.b).



(a) (b)

Figure V.2 : Différents type de laminoirs

Le laminage fournit surtout des demi-produits, mais avec des exceptions notables : tôles fortes, poutrelles et rails, certains tubes, etc. Les autres produits longs sont destinés à l'usinage (barres), au tréfilage (fil machine), au forgeage ; quant aux produits plats, ils vont vers des opérations de mise en forme des métaux en feuilles (emboutissage, découpage fin, repoussage, etc.).

Le laminage commence *à chaud* quand on doit travailler un produit de forte section (nécessaire à la rentabilité des opérations de coulée), donc demandant de fortes puissances de déformation : chauffer le métal, donc le ramollir, revient à diminuer ces puissances et la taille des installations nécessaires, tout en lui conférant la ductilité nécessaire aux très grandes déformations à réaliser.

De nombreux produits (produits longs, tôles fortes, bandes à chaud) ne subissent qu'un laminage à chaud, puis des opérations de finition (traitements thermiques, dressage, décapage, revêtements, usinages). Le passage à des opérations *à froid* est de manière générale nécessaire pour obtenir des tolérances serrées (à quelques micromètres), des propriétés mécaniques élevées par écrouissage, et un bon état de surface. Ne sont laminées à froid pratiquement que les bandes minces.

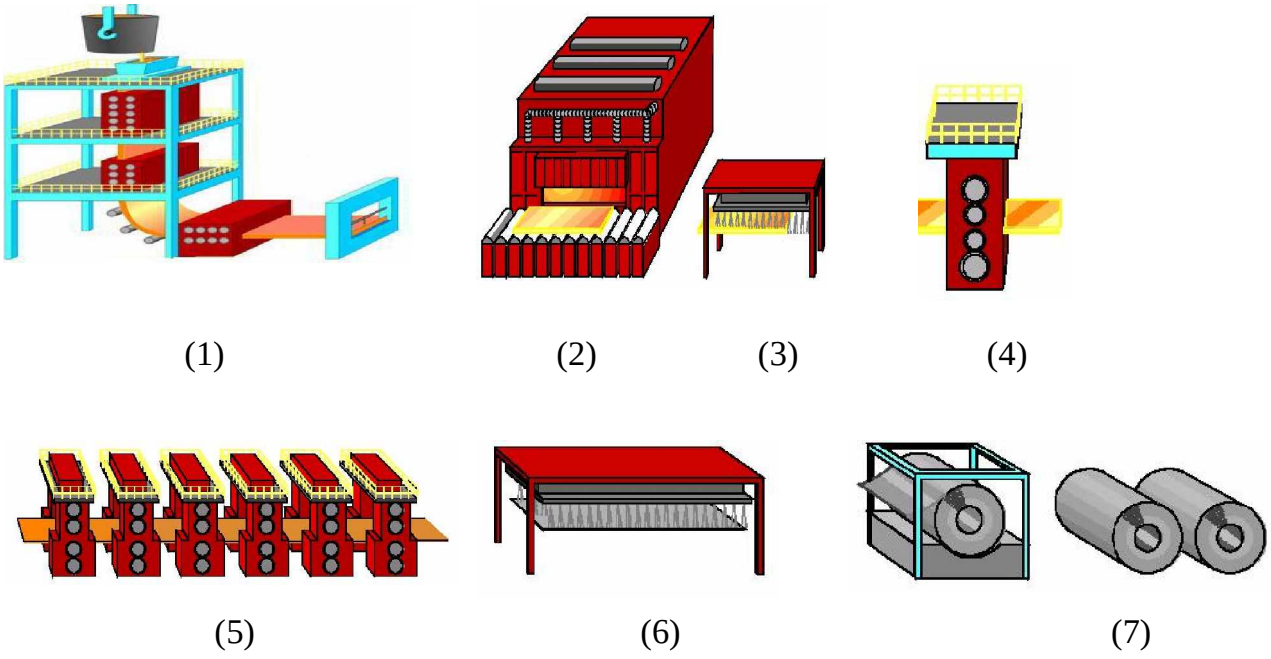
Pour les technologues, les termes « *à froid* » et « à chaud » correspondent simplement au fait que l'on préchauffe ou non le produit à laminer, à une température suffisante pour diminuer sa contrainte d'écoulement et augmenter sa ductilité. Dans notre travail, on s'intéresse exclusivement au laminage à chaud des produits plats.

V.4. Laminage à chaud des produits plat

V.4.1 Description du processus de laminage à chaud

Le schéma descriptif d'un processus de laminage à chaud de produits plat type est donné dans la Figure V.3.

La brame provenant de la machine de coulée continue de l'acier (1) est portée dans un four de réchauffement à longerons mobiles (2) à une température de 1250 °C , température dite de recristallisation. Lors de sa sortie du four, la brame passe dans la brise-oxyde (3) puis dans une cage dégrossisseuse (4). C'est dans cette dernière que la brame subit sa première diminution d'épaisseur. Les dimensions finales de la bande d'acier sont élaborées dans les cages finisseuses (5), puis la bande est bobiné dans une bobineuse (7) après son refroidissement au niveau de la table d'évacuation (6).



(1) (2) (3) (4)

(5) (6) (7)

Figure V.3 : Description d'un processus de laminage à chaud

Il existe plusieurs types de cages de laminoir (Figure V.4) :

- cage de type *duo* composée de deux cylindres ;
- cage de type *quarto* composée de quatre cylindres empilés verticalement ;
- cage de type *sexto* composée de six cylindres empilés verticalement ;
- cage de type multi-cylindre assemblés suivant un arrangement déterminé.

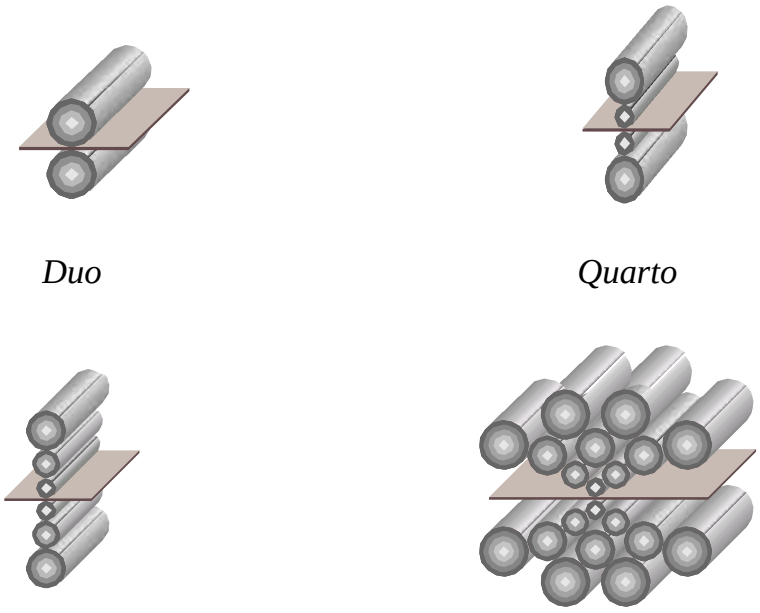


Figure V.4. : Différents types de cages de laminoirs à chaud

Les types de cage les plus couramment utilisées sur les tandems de laminoirs sont les cages de type quarto.

Les mesures disponibles pour les valeurs d'entrée de la tôle dans le train finisseur sont effectuées au niveau du laminoir dégrossisseur (mesure de force donnant l'épaisseur, mesure de température et de largeur).

A son entrée dans les cages finisseuses, la bande est à une température d'environ 900 °C à 850 °C et l'objectif est d'effectuer une réduction d'épaisseur afin de passer d'une épaisseur d'entrée (H_{in}) d'environ 25 mm à des épaisseurs de sortie (h_{out}) de quelques millimètres.

La seule mesure directe d'épaisseur de la bande au niveau du train finisseur est effectuée, à sa sortie, à l'aide d'une jauge d'épaisseur à rayons X.

Il en est de même pour la température qui est mesurée à la sortie du train finisseur à l'aide d'un pyromètre.

Sachant que l'obtention de caractéristiques métalliques tout en maîtrisant la température de laminage est indispensable dans ce processus de transformation à chaud (Vermeulen et al., 1997).

Dans le train finisseur, l'écrasement de la bande de métal est obtenu par l'intermédiaire de cylindres sur lesquels on exerce une force de laminage provenant d'un dispositif de serrage *mécanique ou hydraulique* (Hsu et al., 2000).

Pour une cage de type *quarto* (Figure V.5), l'effort d'écrasement est appliqué sur les extrémités des cylindres les plus gros que l'on appelle *cylindres d'appui*. L'effort est transmis par contact aux cylindres de travail. Généralement, les cylindres de travail sont entraînés par des moteurs qui sont régulés en vitesse.

Grâce aux efforts inter-cylindres et tôle-cylindres très importants (plusieurs centaines de tonnes), les cylindres de travail entraînent en rotation les cylindres d'appui tout en provoquant un entraînement de la tôle.

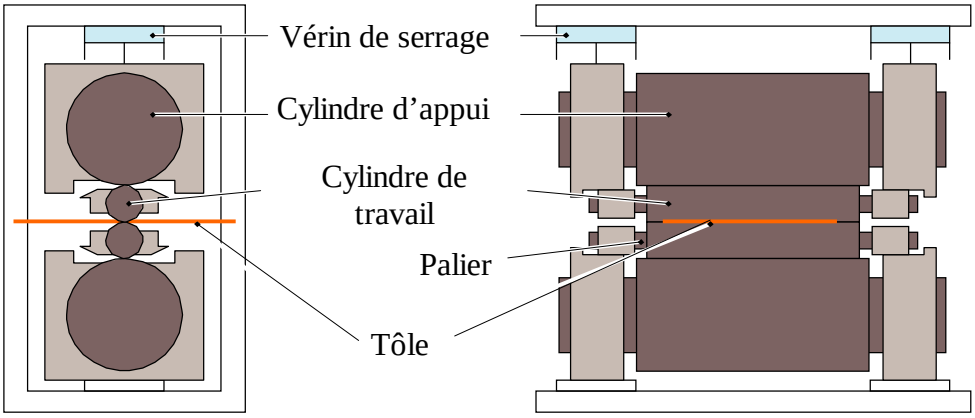


Figure V.5. : Cage de type Quarto

V.4.2. Modèle physique de laminage à chaud

Le modèle physique considéré, est le modèle correspondant au comportement de la tôle au niveau d'une cage du train finisseur sous l'effet d'écrasement. En effet, le comportement de la tôle est caractérisé par la valeur en chaque point et en chaque instant de sa largeur, de son épaisseur et de sa température qui sont des variables distribuées en espace et en temps.

Un modèle distribué rendant compte de ce comportement étant très compliqué pour l'utilisation pratique, il n'est donc considéré que des variables localisées pour la largeur et l'épaisseur (Gomez, 1980).

Le modèle classique souvent utilisé pour le comportement du métal dans une cage de laminage est un modèle qui suppose que le contact entre le métal est le cylindre est du type collant (cf. Figure V.1).

Ce modèle est élaboré à partir des des travaux de E. Orowan (Orowan, 1943). Il dépend de la résistance à la déformation moyenne du métal qui dépend en outre de la température à l'entrée de la cage ainsi que des dimensions géométrique de l'ensemble (cage+tôle).

La mise en équation du modèle est celle développée par (Sims, 1954) dont une variante simplifiée négligeant la déformation (aplatissement) des cylindres de travail, soumis aux contraintes de laminage est donnée par (Ford et Alexander, 1963).

V.4.2.1. Elaboration du modèle de contrôle d'épaisseur (HAGC)

Comme il est montré dans la Figure V.6, le modèle de contrôle d'épaisseur de la tôle, appelé aussi (HAGC : Hydraulic **A**utomatic **G**age **C**ontrol), remplie la tâche de maintien de l'épaisseur finale de la tôle à une valeur prédéterminée constante.

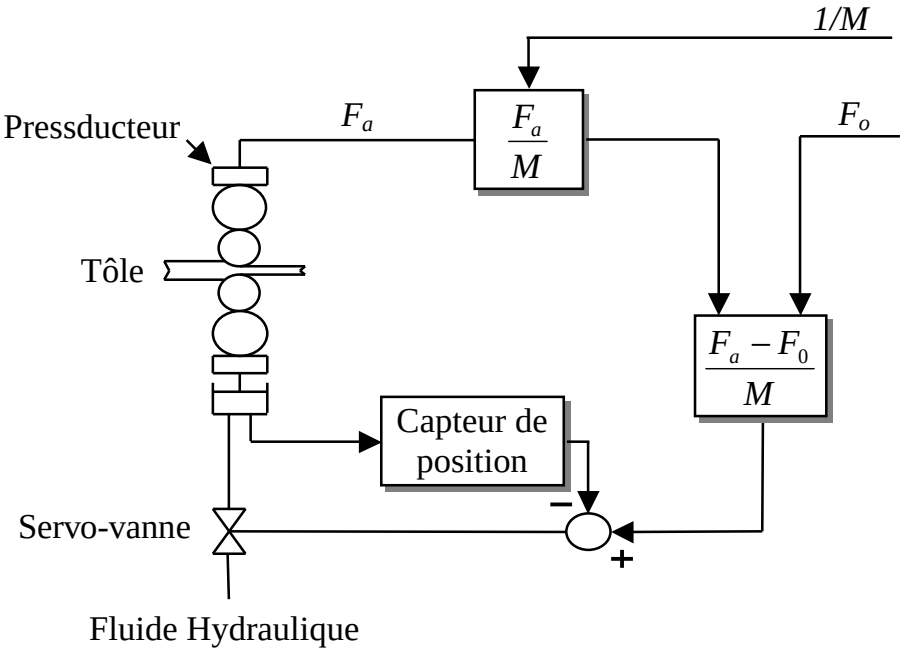


Figure V.6: Modèle de contrôle d'épaisseur (HAGC)

A partir de (Dairiki et al, 1989) et la Figure V.6, l'épaisseur finale désirée h_d et l'épaisseur finale réalisée h_a peuvent être exprimées par :

$$h_d = S_0 + \frac{F_0}{M} \tag{V.2}$$

$$h_a = S_0 + \frac{F_0}{M} + \Delta S \tag{V.3}$$

où S_o est l'écartement à vide des cylindres de travail, F_o est la force de laminage de référence, F_a est la force de la laminage actuelle, M est une constante caractérisant l'élasticité du laminoir (Module d'élasticité du laminoir) et ΔS caractérise le changement de l'écartement des cylindres par l'HAGC. La variation de l'épaisseur Δh est donnée comme suit :

$$\Delta h = h_a - h_0 = \frac{F_a - F_0}{M} + \Delta S \tag{V.4}$$

Pour aboutir à une épaisseur réalisée égale à l'épaisseur désirée, il est impérative de satisfaire la condition théorique $\Delta \boldsymbol{h} \rightarrow 0$. L'équation de contrôle correspondante est alors :

$$\Delta S = -\frac{F_a - F_0}{M} \tag{V.5}$$

La température de la tôle, la vitesse de laminage et la nuance de l'acier étant données. La force de laminage actuelle F_a peut être estimée par le modèle de laminage de Ford-Alexander (Ford et Alexander, 1963), (Yamada et al., 1991), (Gomez, 1980), (Wada et al., 1994) :

$$F_a = 1{,}15\ b\ k_m \sqrt{R(h_e - h_a)}\ Q_p \tag{V.6}$$

où $\boldsymbol{b}$ est la largeur de la tôle, $\boldsymbol{k_m}$ est la résistance à la déformation moyenne plane dans l'emprise, l'expression sous la racine carrée est la longueur de l'arc de contact, $\boldsymbol{h_e}$ est l'épaisseur d'entrée, R est le rayon du cylindre de travail et $\boldsymbol{Q_p}$ est un facteur caractérisant la géométrie de l'emprise et les conditions d'interface tôle-cylindres.

Le terme k_m peut être calculé en fonction d'un ensemble de paramètres de laminage, tels que la température de la tôle T, le coefficient de frottement visqueux μ, et la vitesse de laminage v (Oda et al., 1995).

En pratique, il n'existe pas un modèle physique décrivant d'une manière précise la relation entre les paramètres de laminage cités ci-dessus. On a souvent recours à des relations purement empiriques (Hwu et al., 1996), (Singh et al., 1998).

En effet, plusieurs auteurs ont développé des méthodes différentes pour le calcul du facteur géométrique Q_p en se basant seulement sur des relevés expérimentaux de données de laminage. Il en est de même pour la variable k_m où son évaluation est effectuée empiriquement souvent en fonction de la température et des taux de contraintes (Rigler et al., 1996).

Le fait que le calculateur du processus utilise seulement des paramètres dont les relations physiques sont déterminées empiriquement, il est évident que des problèmes de tolérance sont inévitables lors de l'estimation de l'effort de laminage.

Et dans le but d'améliorer la précision du modèle d'épaisseur pour les raisons objectives citées auparavant, on propose dans un premier temps d'approximer ce processus de contrôle par un modèle neuronal de prédiction basé seulement sur des données entrées-sortie issues d'un ensemble d'évènements de laminage réels en fonctionnement normal (ces données proviennent du laminoir à chaud « LAC » du groupe METAL-STEEL, Algérie, et nous ont été aimablement communiquées par Dr. S. Bouhouche, CERSIM/DRA).

Un échantillon de ces événements est montré dans la Figure V.7, où on peut distinguer :

- AGC_HexAct_F5 (mm) : Epaisseur d'entrée ;
- AGCHSM_HexMea (mm) : Epaisseur de sortie ;
- AGCHSM_HexDev (%) : Variation de l'épaisseur de sortie ;
- RecoIdntPosEccF6 (mm): Perturbation due à l'excentricité de la cage ;
- HGC_FrAct_F6 (kN) : Force de laminage;
- MAC_FMExitTemp (°C) : Température de sortie de la tôle ;
- LHC_ActRollSpd_F6 (m/s) : Vitesse linéaire de sortie de la bande ;

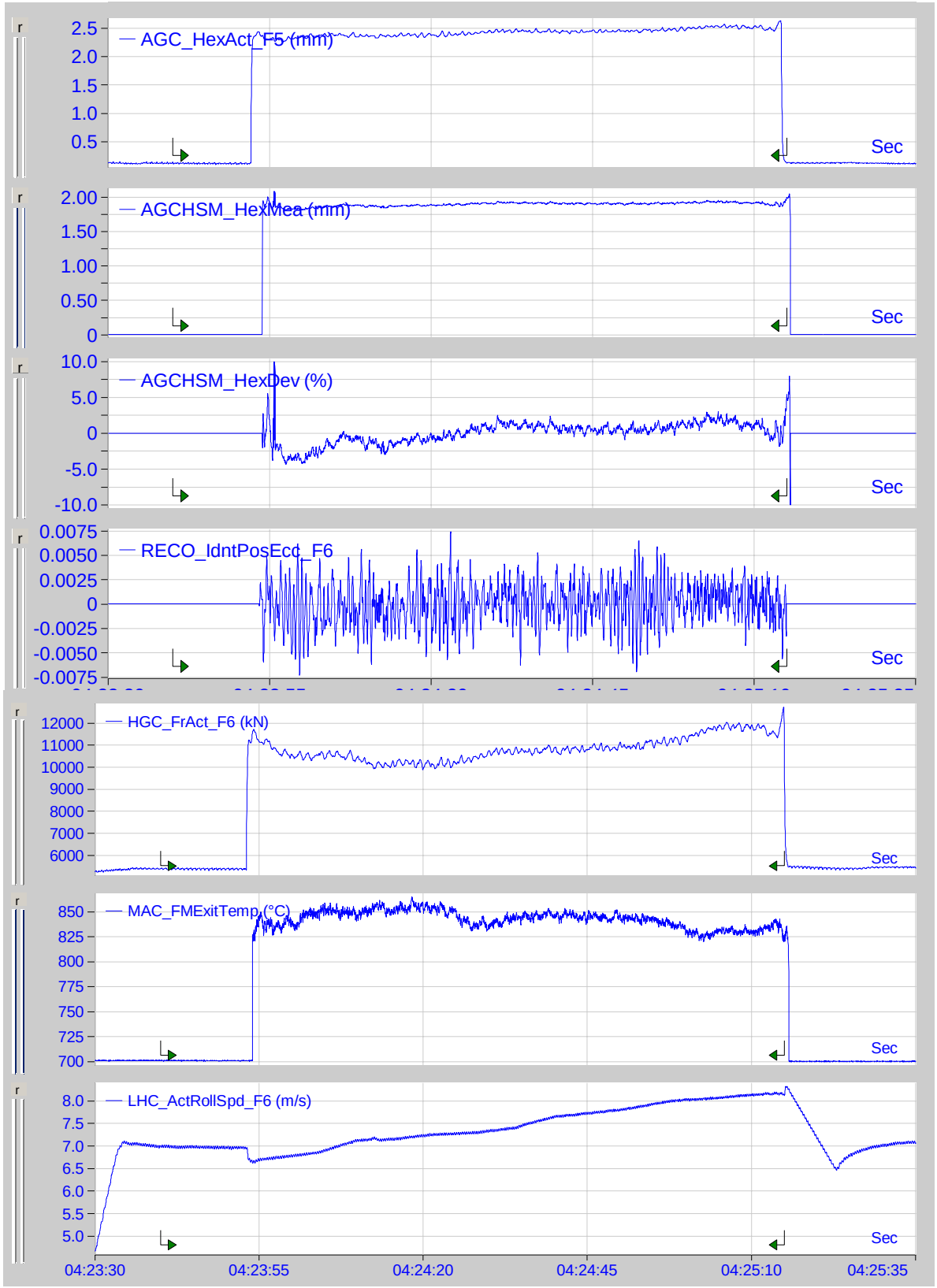


Figure V.7 : Echantillons de données de laminage entrée-sortie réelles

V.5. Etude de possibilité de l'implémentation des stratégies neuronales

Les contraintes industrielles de modernisation et d'amélioration des performances des installations sont telles qu'il n'est plus possible de stopper ou de ralentir la production pour tester une nouvelle fonction.

La mise en œuvre de nouvelles techniques ne doit pas entraîner de perturbations sur le processus et sur les produits réalisés.

Les stratégies innovantes dans le domaine de la régulation ne peuvent donc être installées sur des outils industriels qu'après avoir été testées et éprouvées en simulation (Pederson et Wittenmark, 1996).

Les stratégies de commandes basées sur les réseaux de neurones artificiels (cf. Chapitre IV) ont fait l'objet d'une étude en simulation approfondie et paraissent de très bons candidats pour leur implémentation en temps réel.

Ainsi, un simulateur du processus simulant les phénomènes mis en jeu lors de l'opération de laminage, dans notre cas, peut s'avérer très utile pour la synthèse de nouvelles loi de commande plus performantes.

Sachant que ces stratégies innovantes doivent s'intégrer totalement au processus considéré pour faciliter leur utilisation (Zietsman et al., 1998).

L'utilisation des ressources que ce type de processus industriel met à disposition en standard permet non seulement de réduire les coûts d'étude, mais offre aussi des avantages non négligeables à l'exploitant en améliorant et simplifiant les tâches d'exploitation, de maintenance et de résolution des défauts.

Ces stratégies doivent représenter un gain de temps, un allègement d'efforts et une réduction des risques d'erreurs. Sachant que dans l'environnement industriel considéré, on bénéficie des fonctions suivantes sans recours à une programmation particulière :

- l'étude graphique d'ensemble bien structurée ;
- l'analyse des mesures et des défauts ;
- l'exploitation des fonctions installées (IHM, interface homme- machine) ;
- la puissance de calcul modulable au sein du système multiprocesseur multitâche.

V.6. Synthèse d'une stratégie de commande neuronale « en ligne »

Dans le but d'affiner les régulations en place, on a jugé nécessaire de faire une étude en simulation ayant pour objectif ultime l'amélioration du système de contrôle automatique d'épaisseur (HAGC). Il est ainsi proposé de tester la stratégie de commande neuronale adaptative donnée dans les paragraphes (IV.4.5) et (IV.4.6) du chapitre IV. Le schéma synoptique de cette stratégie de contrôle en ligne est montré dans la Figure V.8.

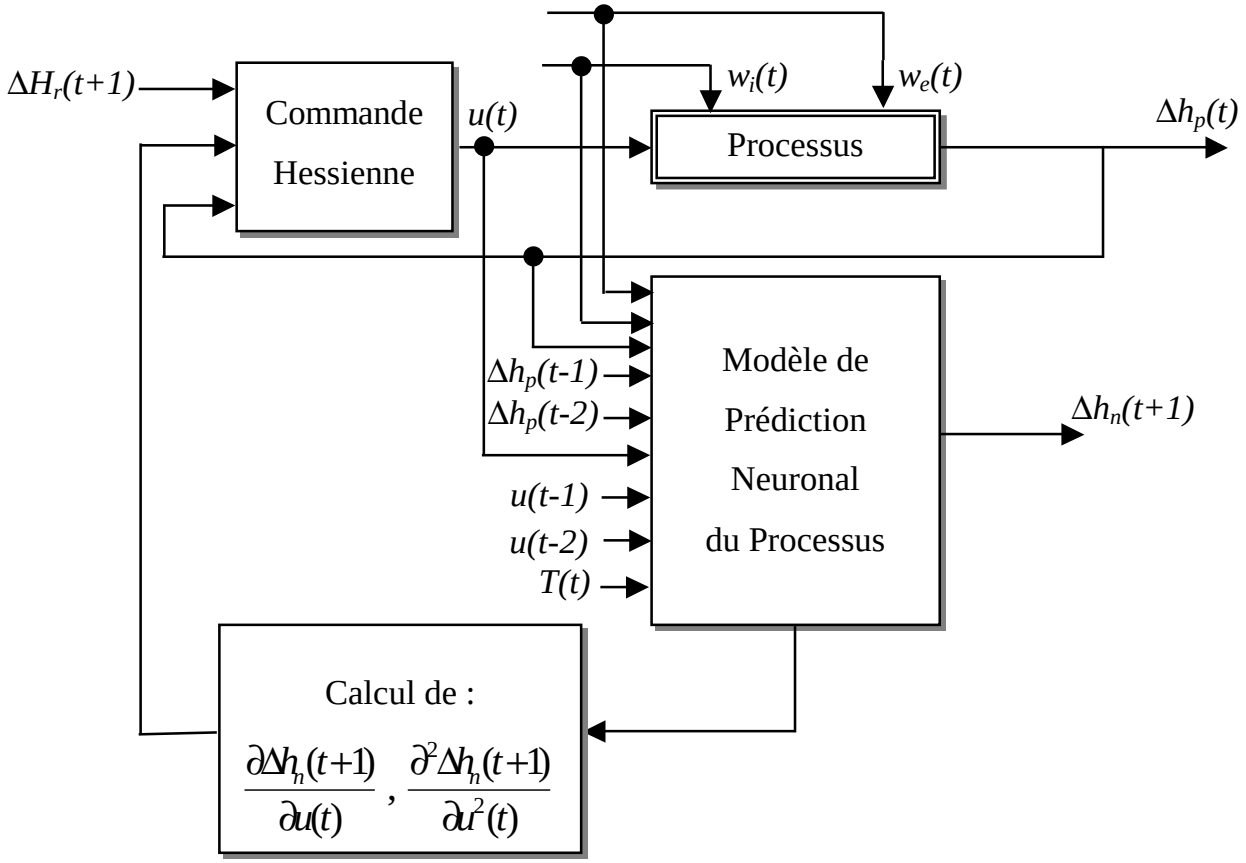


Figure V.8 : Schéma synoptique de la stratégie de contrôle

Où Δh_p est la variation de l'épaisseur de sortie, Δh_n est la variation de l'épaisseur de sortie estimée par le modèle neuronal, ΔH_r est la variation d'épaisseur de référence (désirée), w_e caractérise une perturbation due à l'excentricité des cylindres, w_i est une perturbation d'épaisseur d'entrée provoquée par les marques des glissières dans le four de recristallisation, u caractérise la variation du signal de commande (effort de laminage) et T la température de la tôle.

L'implémentation de cette loi de commande neuronale adaptative doit suivre les étapes suivantes:

V.6.1. Acquisition des données du processus

Les données utilisées pour l'apprentissage du modèle de prédiction sont celles présentées dans la Figure V.7, en tenant compte des caractéristiques techniques du laminoir à chaud données dans la Table 2 de l'Annexe.

V.6.1.1. Modélisation des perturbations extérieures

L'excentricité « faux rond » des cylindres et les marques des glissières sont les raisons principales des variations de l'épaisseur à la sortie d'une cage de laminoir à chaud.

L'effet d'excentricité des cylindres peut être décrit comme une variation *cyclique* de la zone inter-cylindre de travail qui est donc imprimée sur le produit laminé et qui est engendrée par une irrégularité des cylindres et de ses paliers.

Les excentricités peuvent être celles des cylindres de travail mais aussi celles des cylindres d'appui. Les causes d'excentricité des cylindres peuvent être classées en plusieurs familles :

- Ø les imperfections de conception telles que les non-uniformités des chemises des paliers des cylindres d'appui ;
- Ø les imperfections de rectification de la table de roulement des cylindres ;
- Ø les imperfections d'assemblage des différents éléments, chemises, paliers, empoises, cylindres ;
- Ø la déformation des cylindres et paliers suite à un usage intensif aux limites.

L'excentricité des cylindres peut être observée à partir des capteurs déjà existants sur l'installation. Ces derniers permettent :

- Ø la mesure de la force de laminage par une jauge de contrainte (pressducteur) installée dans le bâti de la cage;
- Ø la mesure de l'épaisseur à la sortie de la cage ;

Classiquement, il y a toujours une jauge d'épaisseur à la sortie d'un tandem de laminoirs. Il est donc possible, grâce à une analyse spectrale des variations d'épaisseur à la sortie du tandem de laminoirs, d'observer les excentricités de chaque cylindre (Konno et al., 1995). Sachant que la fréquence de l'effet d'excentricité est une fonction de la vitesse de rotation des cylindres.

En se basant sur des données pratiques, il est montré dans la littérature que le modèle raisonnable de ce type de perturbation peut être celui d'un filtre discret passe bande piloté par un bruit blanc. Il est exprimé comme suit :

$$H_{w_e}(z) = -g_m \frac{300(z-1)}{z^2 - 2z + 1} \qquad \text{(V.7)}$$

avec : $g_m = 1/M$, (M: module d'élasticité du laminoir).

Pour ce qui est de la perturbation associée à l'épaisseur d'entrée, il est bien établi que la perturbation est provoquée par les marques des glissières lors du séjour de la brame dans le four de recristallisation (Figure V.9).

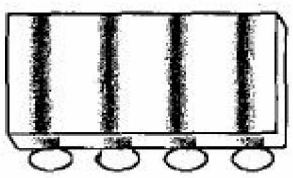


Figure V.9 : Marques de glissière dans la brame

Cette perturbation d'entrée est souvent modélisée par un filtre passe bas piloté par un bruit blanc. Pour une période d'échantillonnage de 0.02s (période choisie lors de l'acquisition des données de laminage), une représentation numérique de ce filtre discret est donnée comme suit (Lee et al., 2003):

$$H_{w_i}(z) = g_m \frac{100}{z-1} \qquad \text{(V.8)}$$

Il est à noter au passage que la période d'échantillonnage est souvent déterminée selon l'expression empirique : $T_{95} / T_e \approx 5$ à 15. Où T_{95} est l'instant où la sortie du processus est égale à 95% de sa valeur finale.

V.6.2. Apprentissage « hors ligne » du modèle de prédiction

Comme il est montré dans la Figure V.8, pour l'identification du modèle de prédiction du processus, et en tenant compte des conditions de sélection de la taille d'un modèle neuronal développées dans le chapitre III, on a choisi un réseau de neurones à 8 entrées , une couche cachée contenant 15 neurones et un neurone de sortie linéaire.

Le modèle de prédiction neuronal est entraîné hors ligne en utilisant une séquence de données de laminage pendant un fonctionnement normal.

Notons que que les données utilisées pendant l'apprentissage sont collectées dans un format non standard (non ASCII) à l'aide du logiciel d'acquisition et d'analyse en temps réel (IbaAnalyzer) implémenté au niveau du calculateur de processus du laminoir (cf. Figure V.10). Pour pouvoir exploiter ces données, nous avons procédé à leur conversion sous le format Excel de Microsoft. Sachant que la lecture, sous Matlab©, d'un fichier de données Excel s'effectue simplement par appel de la fonction « xlsread ».

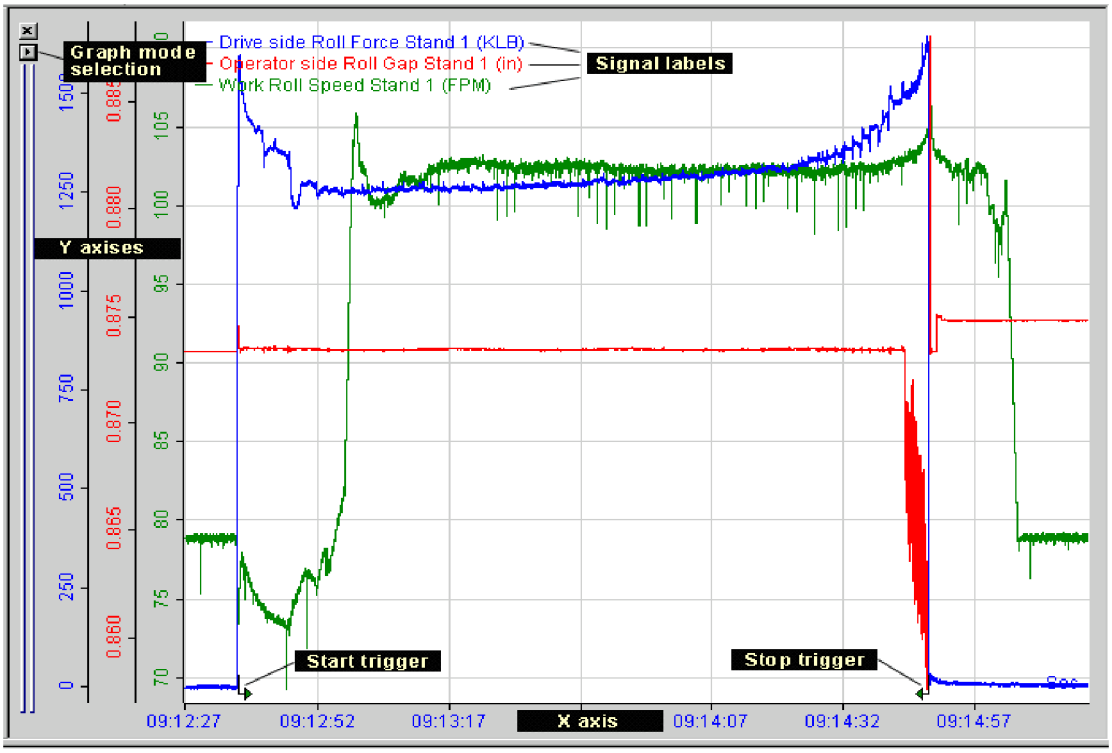


Figure V.10 : Exemple d'acquisition de données sous IbaAnalyzer

Parmi les conditions à satisfaire pour réaliser un apprentissage raisonnable du modèle neuronal de prédiction, on peut citer :

- Ø la sortie du modèle doit être proche de celle du processus réel ;
- Ø le résidu doit être situé à l'intérieur d'une plage de variation fixée par l'utilisateur ;
- Ø l'erreur de prédiction finale doit être la plus faible possible.

Après la convergence du réseau de neurones proposé vers l'erreur quadratique moyenne désirée, nous avons utilisé une nouvelle séquence de données, ayant les mêmes caractéristiques que la séquence d'apprentissage, pour effectuer un test de validation.

Les courbes d'évolution de l'erreur de prédiction (d'apprentissage) et l'erreur de validation sont montrées dans la Figure V.11.

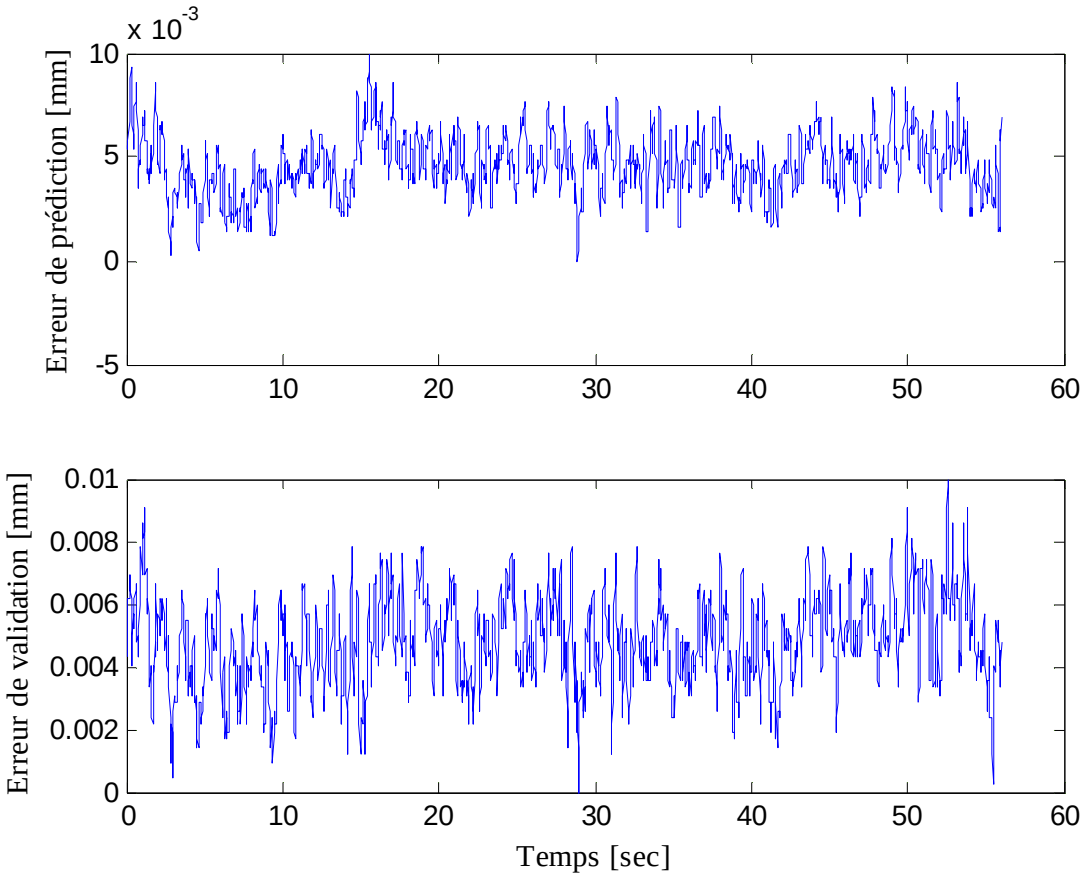


Figure V.11 : Courbe d'évolution des erreurs de prédiction et de validation

V.6.3. Elaboration de loi de commande adaptative

La loi de commande neuronale adaptative proposée est implémentée en utilisant les équations (IV.4) à (IV.12) du chapitre IV, avec un facteur de pondération approprié, soit $\rho = 10^{-8}$.

Et dans le but d'élimination des effets des erreurs de modélisation, on propose de rétro-propager l'erreur de poursuite à chaque période d'échantillonnage à travers le modèle neuronal afin de modifier les poids et les biais selon l'équation de mise à jour (IV.13) du chapitre IV, avec un gain d'adaptation approprié (α=0.8).

Les performances de cette loi de commande sont montrées dans les courbes de la Figure V.12.

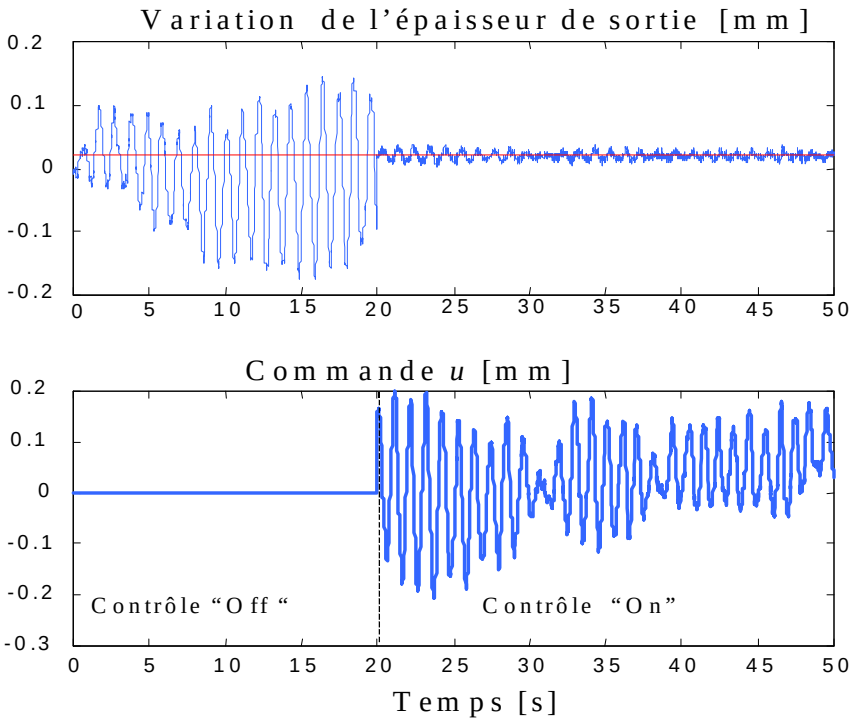


Figure V.12 : Courbes d'évolution de la commande et de la sortie

D'après les courbes de la Figure V.12, Il est bien montré que lorsque la loi de commande est activé « On », la variation de l'épaisseur de sortie poursuit pratiquement la grandeur de référence (ΔHr=0.02 mm ou 20 µm) malgré la présence de perturbations extérieures significatives.

Il est à remarquer que la variation de l'écartement des cylindres ΔS caractérisant la commande par anticipation appliquée au processus est inversement proportionnelle à la variation de l'effort de laminage (cf. équation V.5).

Et pour mieux apprécier l'algorithme de commande proposé, nous avons utilisé les grandeurs d'entrée d'une autre séquence de données de laminage que nous avons injecté dans le modèle de prédiction. Ces données sont montrées dans la Figure V.13.

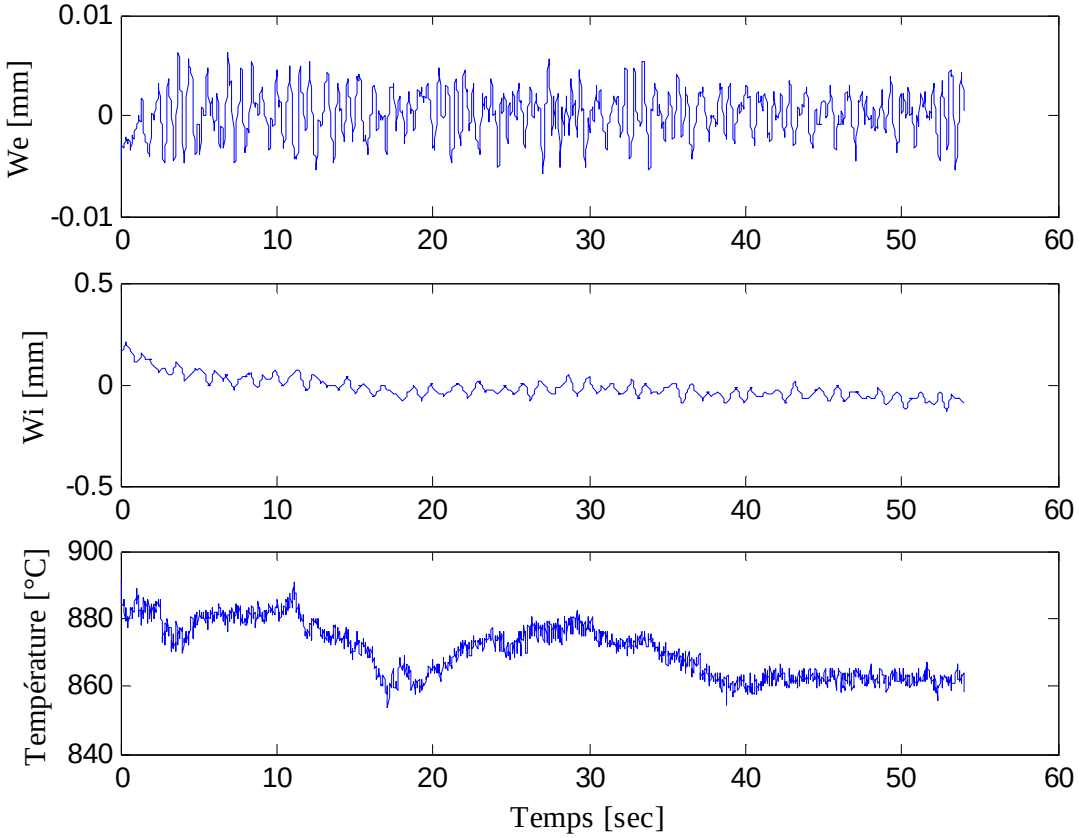


Figure V.13 : Données réelles du processus

Après l'élaboration de la loi de commande, en simulation, nous avons comparé la sortie simulée avec la sortie réelle du processus. Il est montré, à partir des courbes de la Figure V.14, que la commande développée en simulation semble avoir pratiquement des performances acceptables relativement à la sortie réelle du processus, ce qui représente un résultat encourageant. Cependant, d'autres travaux de simulation sont indispensables pour valider cette stratégie de commande, surtout lorsque des nuances et des épaisseurs de sortie différentes sont considérées.

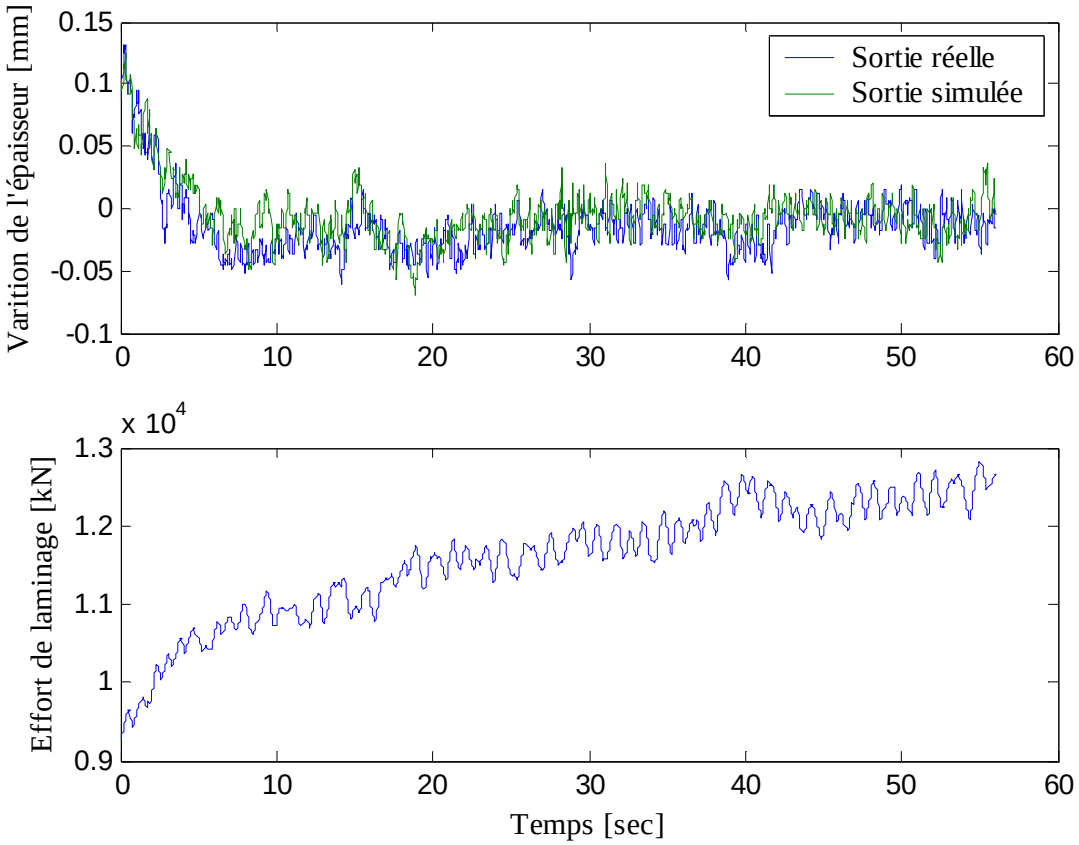


Figure V.14 : Courbes d'évolution de la commande et des sorties réelle et simulée

L'évolution de la loi de commande exprimée en fonction de la variation de l'écartement des cylindres ΔS est montrée dans la Figure V.15.

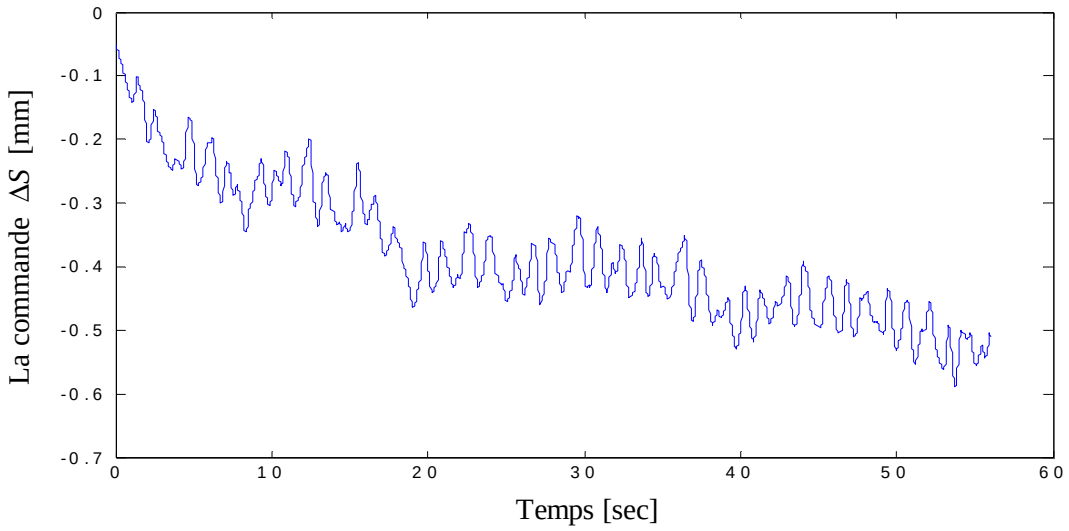


Figure V.15 : Courbe d'évolution de la variation de ΔS

V.7. Conclusion

Il est souvent exigé une efficacité opérationnelle des laminoirs à chaud, surtout vis à vis de la réduction des produits déclassés ou rejetés.

Une bande en acier est considérée déclassée si ses caractéristiques mécaniques et géométriques n'obéissent pas aux exigences du client, ce qui va entraîner sa vente à un prix plus bas. Ce qui implique de pertes énormes en énergie et en profit.

Notre objectif était justement de veiller à ce que ces exigences soient respectées, en proposant d'intervenir moyennant une stratégie de contrôle afin de diminuer les variations d'épaisseur (grandeur prise en compte comme critère de qualité).

A cet effet, un contrôle neuronal adaptative est élaboré en utilisant un algorithme d'apprentissage rapide et une loi de commande non itérative basée sur la minimisation d'un critère de performance quadratique.

La faisabilité et l'efficacité de cette loi de commande sont éprouvés moyennant une étude en simulation basée sur l'exploitation de données réelles du processus considéré.

De plus, il est encore montré que l'outil de modélisation utilisé, en l'occurrence les réseaux de neurones, semble être un outil très efficace à cause de sa robustesse par rapport aux défauts de modélisation et aux perturbations extérieures, ce qui constitue un facteur important lors de la modélisation des processus industriels.

Conclusion générale

Tout au long du présent travail nous avons tiré parti de deux caractéristiques fondamentales des réseaux de neurones :

- *la propriété d'approximation universelle parcimonieuse*, dont l'apport est manifeste pour la modélisation et la commande de processus non linéaires ;
- *l'existence d'algorithmes d'apprentissage également universels*, au sens où leur mise en œuvre ne dépend pas de l'application considérée ni de la complexité du réseau soumis à un apprentissage.

Dans un premier temps, il est effectué une étude théorique sur la modélisation et la commande de processus par réseaux de neurones dans un cadre aussi général que possible, en nous appuyant sur les deux caractéristiques fondamentales des réseaux de neurones que nous avons cités. Nous avons ainsi :

- déterminé les arguments de la fonction théorique que doit réaliser le réseau de neurones selon la tâche considérée (le prédicteur théorique associé à un modèle hypothèse donné pour une tâche de modélisation, le correcteur théorique associé au modèle d'un processus pour une tâche de commande) : la propriété d'approximation universelle affirme alors que, si cette fonction existe, elle est réalisable par un réseau de neurones ;
- construit un système d'apprentissage du réseau de neurones (éléments constitutifs, algorithme) conduisant à une réalisation de la fonction théorique, si elle existe : l'universalité des algorithmes d'apprentissage garantit que, en pratique, cette réalisation est possible, quelle que soit la complexité de la fonction.

Dans le domaine de la commande des processus non linéaires par réseaux de neurones, et après la description des différentes stratégies de commande basées sur les réseaux de neurones, il est étudié les conditions de leur mise en œuvre au sein des processus industriels.

Les performances de ces stratégies de commande sont vérifiées à travers des exemples illustratifs de processus industriels réels.

De même, et dans le souci de surmonter la contrainte du temps dans les processus industriels, il est développé une nouvelle stratégie de commande rapide basée sur le calcul en ligne de l'Hessien d'un critère de performance quadratique et l'utilisation d'un modèle neuronal de prédiction du processus.

La prise en compte par la stratégie de commande développée de l'effet des erreurs de modélisation est étudié en introduisant une étape d'adaptation des paramètres du modèle de prédiction neuronal. Il est proposé un ajustement en ligne des poids du modèle neuronal à chaque période d'échantillonnage selon un mode d'apprentissage incrémental.

Notre démarche est illustré par une application industrielle, « le contrôle d'épaisseur de bande dans un laminoir à chaud ». Il est présenté une description assez détaillée du processus. De même, il est donné son modèle empirique utilisé pour la prédiction des efforts de laminage (forces de laminage).

Une base de donnée réelle représentant le fonctionnement du laminoir est utilisée pour l'apprentissage et la validation du modèle neuronal.

La loi de commande neuronale adaptative proposée est obtenue par une nouvelle méthode utilisant le Hessien d'une fonction coût quadratique. Les performances de cette loi de commande sont évaluées par simulation, en utilisant un modèle neuronal raisonnable de prédiction estimé à partir de données entrée/sortie de laminage réelles et la faisabilité d'implémentation de ce schéma de contrôle est vérifiée en effectuant des simulations en présence de perturbations extérieures.

Cependant, l'éventuelle implémentation en temps réel de cette stratégie de commande nécessite une étude plus approfondie en tenant compte de toutes les contraintes de l'environnement industriel considéré.

Afin de palier à certains problèmes qui restent en suspens, de nombreuses études peuvent être entreprises.

En particulier, l'étude de la possibilité d'associer à la stratégie développée dans le cadre de ce travail une couche de surveillance et de diagnostic en temps réel basée sur les réseaux de neurones ou sur d'autres types de techniques avancées de diagnostic.

Ackley D., Hinton G. and Sejnowski T., A learning algorithm for Boltzmann machines, *Cognitive Science, Vol. 9,* 1985, pp.147-169.

Astrôm K.J. and Wittenmark B., Computer Controlled Systems. Theory and Design, *Prentice Hall Inc., 2nd Edition, Englewood Cliffs, New Jersey,* 1989.

Battiti R. , First and Second Order Methods for Learning: Between Steepest Descent Methods and Newton's Methods, *Neural Computation, Vol. 4, No.2,* 1992, pp.141-166.

Baum E.B. and Haussler D., What size net gives valid generalization, *Neural Computing, Vol. 1(1),* 1989, pp.151-160.

Billings S.A., H.B., Jamaluddin H.B. and Chen S., Properties of Neural Networks With Applications to Modelling non-linear Dynamical Systems, *Int. Journal of Control, Vol. 55, No 1,* 1992, pp.193-224.

Bishop C., Neural Networks for Pattern Recognitions, *Clarendon Press-Oxford New-York,* 1995.

Bruenelli R., Training nets through stochastic minimization. *Neural Networks, Vol. 7(9),* 1994, pp.1405-1412.

Bryson A.E. Jr. and Yu C.H., Applied Optimal Control. Optimization, Estimation and Control, *by Ginn and Company Waltham, Massachusetts,* 1969.

Cebuhar W.A. and Costanza V., Non Linear Control of CSTR's , *Chem. Eng. Science, 39,* 1984, pp.1715-1722.

Cybenko G., Approximation by superposition of a sigmoid function. *Math. of Control, Signals and Systems, Vol. 2(4),* 1989, pp.303-314.

Dairiki O., Mabushi H., Degawa I., and Nakamura H., Progress of AGC Utilisation Techniques at Plate Mill of Oita Works, *Nippon Steel Technical Report, No. 42,* July 1989, pp.38-47.

Dreyfus G., Idan Y., The Canonical Form of Non-linear Discrete-Time Models, *Neural Computation Vol. 10, No. 1,* 1998, pp.133-164.

Eki, Y., Hirasawa K., Murata J., and Hu J., Feed-forward Control of Thermal plants using Neural Networks, *Research Reports on Information Science and Electrical engineering of Kyushu University, Vol.3, No.1,* 1998, pp.13-21.

Fausset L., Fundamentals of Neural Networks. Architecture - Algorithms and Applications, *1st Edition, Printice Hall, Inc.,* 1994.

Fletcher R., Practical methods of optimisation, *John Wiley and Sons Ltd, second edition, New-York, USA,* 1987.

Ford H. and Alexander J.M., Simplified Hot Rolling Calculations, *Journal of the institute of Metals, Vol. 92,* 1963-64, pp.397-403.

Friedland B., Control system design. An introduction to state space methods, *McGraw-Hill Company, New York,* 1987.

Funahashi K. , On the Approximate Realization of Continuous Mappings by Neural Networks, *Neural Networks, Vol. 2,* 1989, pp.183-192.

Gallinari P., Heuristiques pour la généralisation, S. Thiria, Y. Lechevallier, O. Gascuel et S. Canu editeurs. « *Statisitiques et méthodes neuronales »,* , Dunod, Paris, 1997, Chapitre 14, pp.230-243.

Gayner R.J. and Downs T., On the properties of error functions that affect the speed of backpropagation learning, *In Marinaro M. and Morasso P. G., editors, ICANN'94, Proceeding of the International Conference on Artificial Neural Networks ; 26-29 May 1994,* Sorrento, Italy, 1994, pp.557-560.

Goldberg D.E., Genetic algorithms in search, optimization and machine learning. *Addison Wesley, New-York, NY,* 1989.

Gomez C., Modélisation et Simulation d'un Train de Laminage à Chaud, *Thèse de Doctorat, présentée à l'Université de Paris-sud, Centre d'Orsay, France,* Décembre 1980.

Goodwin G.C., Payne R.L., Dynamic System Identification : Experiment, Design and Data analysis, *Mathematics in Science and Engineering, Volume 136, Academic Press,* 1977.

Gourdin A. et Boumahrat M., Méthodes numériques appliquées, *seconde édition, Office des publications universitaires, Alger,* 1991.

Grandvalet Y., Canu S. and Boucheron S., Noise injection : theoretical prospects. *Neural Computation,* Vol. 9, No. 7, 1997, pp.1241-1256.

Grosberg S., Adaptive Pattern Classification and Universal recoding, I : Parallel Development and coding of neural features detectors, *Biological Cybernetics, Vol. 23,* 1976, pp.121-134.

Günter W.R., Heinrich R.A., Wolfgang S., Karl A. and Karl H.W., Improved Rolling Mill Automation by Means of Advanced Control, *Techniques and Dynamic Simulation, IEEE Transactions on Industry Applications, Vol. 32, No. 3,* May/June 1996, pp.599-607.

Hagan M.T., Menhaj M., Training Feed-forward Networks with the Marquardt Algorithm, *IEEE Transactions on Neural Networks, 5*(6), 1994, pp.989-993.

Hambrecht A. et Robin P., Intégration d'une régulation adaptative neuronale dans un environnement industriel d'automatismes , *REE, Vol. n° 1,* janvier 1998.

Hassibi B. and Stork D. G., Second order derivatives for network pruning : Optimal brain surgeon. *In S.J. Hanson, J.D. Cawnan, and C.L. Giles, Editors, Advances in Neural Information Processing Systems 5,* Morgan Kaumann, San Mateo, CA, 1993, pp. 164-171.

Hebb D., The Organization of Behavior, *New York: Wiley,* 1949.

Hertz J., Krogh A. and Palmer R.G., Introduction to the theory of neural computation, *Computation and neural systems series*. Addison-Wesley, New-York, NY, 1991.

Hopfield J.J., Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences, Vol. 79*, 1982, pp.2554-2558.

Hornik K., Stinchcombe M. and White H., Multilayer Feedforward Networks are Universal Approximators, *Neural Networks, Vol. 2*, 1989, pp.359-366.

Hornik K., Stinchcombe M. and White H., Universal approximation of an unknown mapping and its derivatives using multiplayer feedforward networks, *Neural Networks, Vol. 3* ., 1990, pp.551-560.

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

Hsu Y.L., Liang H.P., and Tsai S.J., An Improvement Response for CSC No.1 HSM, *IEEE Transactions on industry Applications, Vol. 36, No 3*, May/June 2000, pp.854-860.

Huang H. , Chao Y. and Cheng C., Identification and Adaptive Control for a CSTR Process, *Proc. Of the American Control Conference, Vol. N0.1*, 1982, pp.551-556.

Hwu Y.J., Lenard J.G., Application of Neural Networks in the Prediction of Roll Force in Hot Rolling, *Proc. 37th Mechanical Working and Steel Processing Conf., the Iron and Steel Society, Warrendale*, 1996, pp.549-554.

Jacobs R.A., Increased rates of convergence through learning rate adaptative, *Neural Networks, Vol. 1*, 1988, pp.295-307.

Konno Y., Shioya M., and Ueyama T., Development and Application of Dynamic System Simulator, *Nippon Steel Technical Report, No. 67*, October 1995, pp.63-68.

Lengellé R. and Denoeux T. Training MLPs layer by layer using an objective function for internal representations. *Neural Networks*, 9, 1996, pp.83-97.

Le Cun Y., Une procédure d'apprentissage pour réseau à seuil asymérique. *In Cognitiva 85: A la Frontière de l'Intelligence Artificielle des Sciences de la connaissance des Neurosciences*, Paris: CESTA, Paris, 1985, pp.599-604.

Le Cun Y., Boser B., Denker J.S., Hendersen D., Howard R.E., Hubard W., Jackel L.D., Backpropagation applied to handwritten zip code recognition. *Neural Computation, Vol. 1*, 1989, pp.541-551.

Lee Y.K., Kim S.W., and Hong S.C., Modified feed-forward automatic gauge control using adaptive line enhancer in hot strip finishing rolling, *Industrial Electronics Society, IECON '03. The 29th Annual Conference of the IEEE, Vol. 3*, 2-6 Nov. 2003, pp.2233-2238.

Leontaritis I.J. and Billings S.A., Input-output parametric models for non-linear systems–Part 1: Deterministic non-linear systems; Part 2: Stochastic non-linear systems, *International Journal of Control, Vol. 41,* 1985, pp.164-168.

Levenberg K., A Method for the Solution of Certain Non-linear Problems in Least Squares, *Quarterly Journal of Applied Mathematics II (2),* 1944, pp.164-168.

Linden A. and Kindermann K., Inversion of multilayered nets, *Int. Joint. Conf. On Neural Networks (Wachington D.C.),* June 1989, pp.425-430.

Ljung L., System Identification ; Theory for the User, *Prentice Hall, Englewood Cliffs, New Jersey,* 1987.

Marie M. et Mokhtari M. , Application de MATLAB© Ver. 5 et SIMULINK Ver. 2, Chapitre IV, *Springer-Verlag, France,* 1998.

Marquardt D., An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM J. Appl. Math. 11,* 1963, pp.164-168.

McCulloch W. and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics, Vol. 5,* 1943, pp.115-133.

Minoux M., Programmation Mathématique : Théorie et Algorithmes, *Ed. Dunod,* 1983.

Minsky M., et Papert S., Perceptrons, *MIT Press, Cambridge, MA,* 1969.

Montmitonnet P., Laminage, Objectifs et Modélisation, *Techniques de l'Ingénieur, Traité Matériaux Métalliques, M 3 065,* 2000, pp.1-12.

Moussaoui A.K., Bouazza S.E. and Abbassi H.A., Feedforward Control of Hot Strip thickness Process Using Neural Networks, *Proceedings du Séminaire National sur l'Automatique et les Signaux, SNAS'99, U.B.M. Annaba, Algérie,* 1999, pp.237-241.

Moussaoui A.K., Abbassi H.A., Kambhampati C., and Goraine H., Neural Networks Feed-forward Controller for Hot Strip Rolling Process, *Proc. 2nd ICSC Symp. on Engineering of Intelligent Systems, University of Paisley, Scotland, U.K.*, (Editor: C. Fyfe, ISBN 3-906454-23-1), 2000, pp.661-665.

Moussaoui A.K., Abbassi H.A., Bouazza S.E. and Kondratas A., Neural-Networks-Based Adaptive Nonlinear Controller for Hot Strip Rolling Mill, Mechanika, ISSN:1392-1207, 2003, Nr.6(44). p.55-61.

Moussaoui A.K., Abbassi H.A. and Bouazza S.E., Neural-Networks-Based Nonlinear Optimal Controller For Hot Strip Rolling Process, 3ème Conférence de Génie Electrique, *In Proceedings of CGE'03, Ecole Militaire Polytechnique,* 15-16 Février 2004, Alger.

Narendra K.S. and Parthasarathy K., Identification and Control of Dynamical Systems using Neural Networks, *IEEE Trans. on Neural Networks, Vol. NO. 1,* 1990, pp.4-27.

Nash J.C., Compact Numerical Methods for Computers : Linear Algebra and Function Minimization, *Adam-Hilger Ltd, Bristol,* 1980.

Neelakantan R. and Guiver J., Applying Neural Networks, *Hydrocarbon Processing , Gulf Publishing Company, Houston,* September 1998, pp.91-96.

Nerrand O., Réseaux de neurones pour le filtrage adaptatif, l'identification et la commande de processus, *Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris VI*, 1992.

Oda T., Satou N., and Yabuta T., Adaptive Technology For Thickness Control of Finisher Set-up on Hot Strip Mill, *ISIJ International, 35*(1), 1995, pp.42-49.

Orowan O., The Calculation of Roll Pressure in Hot and Cold Flat Rolling, *Proc. Inst. Mech. Eng., Vol. 150*, 1943, pp.140-167.

Oussar Y., Réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus, *Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris VI*, 1998.

Park J. and Sandberg I.W., Universal Approximation Using Radial-Basis-Function Networks, *Neural Computation Vol. 3, No. 2,* 1991, pp.246-257.

Parker D.B., Learning-logic, *Technical Report TR-47, Center for Computational Research in Economics and Management Sci., MIT,* April, 1985.

Pedersen L.M. and Wittenmark B., Multivariable Controller design for hot rolling mill, *In Proceedings of the 13th trennial IFAC World Congress, Vol. M*, IFAC, 1996, pp.475-480.

Plaut D.C., Nowlan S.J. and Hinton G.E., Experiments on learning by back-propagation, *Technical Report CMU-CS-86-126, Carnegie Mellon University - Pittsburgh, PA 15213,* 1990.

Psaltis D., Siders A. and Yamamura A., A Multilayered Neural Network Controller, *IEEE Control System Magazine,* April 1988, pp.17-21.

Rigler G.W., Abert H.R., Staufer W., Aistleitner K., and Weinberger K.K., Improved Rolling Mill Automation by Means of Advanced Control Techniques and Dynamic Simulation, *IEEE Transactions on industry Applications, Vol. 32, No 3,* May/June 1996, pp.599-607.

Rivals I., Personnaz L., Dreyfus G., Ploix J.L., Modélisation, classification et commande par réseaux de neurones : principes fondamentaux, méthodologie de conception et applications industrielles, *Les réseaux de neurones pour la modélisation et la commande de processus, J.P. Corriou éd., Lavoisier Tec et Doc,* 1995.

Rivals I., Modélisation et commande par réseaux de neurones : application au pilotage d'un véhicule autonome, *Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris VI,* 1995.

Robbins H. and Monro S., A stochastic approximation method, *Annals of Math. Stat., Vol. 22,* 1951, pp.400-407.

Rosenblatt F., The Perceptron: a Perceiving and Recognizing Automaton, *Project PARA, Report 85-460-1, Cornell Aeronautical Lab.,* 1957.

Rosenblatt F., The Perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review, Vol. 65,* 1958, pp.386-408.

Rumelhart D.E., Hinton G.E., and Williams R.J., Learning Internal Representations by Error Propagation. *In Parallel Distributed Processing : Explorations in the Microstructures of Cognition, Vol. 1, D. Rumelhart and J. McClelland Eds. Cambridge: MIT Press,* 1986, pp.318-362.

Silva F.M., Almeida L.B., Speeding up backpropagation. In R. Eckmiller, editor, *Advanced Neural Computers, Elsevier Science Publishers B.V., North-Holland,* 1990, pp. 151-158.

Sims R.B., Calculation of Roll force and Torque in Hot Rolling Mills, *Proc. Inst. Mech. Eng., Vol. 168,* 1954, pp.191-219.

Singh S.B., Bhadeshia H.K., Mackay D.J., Carey H., and Martin I., Neural Network Analysis of Steel Plate Processing», *Iron and steel Making, 25*(5), 1998, pp.355-365.

Sjöberg J. , Hjalmerson H. and Ljung L., Neural Networks in System Identification, *Preprints 10th IFAC symposium on SYSID, Copenhagen, Denmark. Vol.2,* 1994, pp.49-71.

Sorensen O., Neural Networks in Control Applications, *Ph.D. Thesis. Aalborg University, Department of Control Engineering,* 1994.

Thomas P. and Bloch G., Robust Pruning for Multilayer Perceptrons. *In Proceedings of IMACS/IEEE Multiconference on Computational Engineering in Systems Applications CESA'98, Vol. 4, Nabeul-Hammamet, Tunisia,* April 1-4, 1998, pp.17-22.

Tollenaere T., SuperSAB: fast adaptive back propagation with good scaling properties, *Neural Networks, Vol. 3,* 1990, pp.561-573.

Urbani D., Méthodes Statistiques de Sélection d'Architectures Neuronales: Application à la Conception de Modèles de Processus Dynamiques, *Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris VI,* 1995.

Vermeulen W., Bodin A., and Van Der Zwaag S., Prediction of the Measured Temperature after the last Finishing Stand Using Artificial Neural Networks, *Steel Research 68*(1), 1997, pp.20-26.

Wada B., Sumi H., Ueda I., and Suziki K., Automatic Gauge Control For Heavy Plate Mill, *La Revue de Métallurgie - CIT,* Mars 1994, pp.465-470.

Walter E. et Pronzato L., Identification de modèles paramétriques à partir de données expérimentales, *Editions Masson, Paris,* 1994.

Warwick K., An overview of Neural Networks in Control Applications, *In "Neural Networks for Robotic Control, Theory and Applications ", Edited by Zalzala A.M.S. and Morris A.S., First published by Ellis Horwood Ltd.,* 1996, pp.1-25.

Weerasooriya S., Sharkawi M.A., Identification and Control of a DC Motor Using Back-Propagation Neural Networks , *IEEE Trans. On Energy Conversion*, *6*(4), 1991, pp.363-369.

Werbos P., Beyond regression: new tools for prediction and analysis in the behavioural sciences, *PhD thesis, Harvard University, Cambridge, MA.*, 1974.

Widrow B. and Hoff M., Adaptive switching circuits, *WESCON Convention Record, New York: IRE, Vol. 4*, 1960, pp.96-104.

Wu J. K., Neural Networks and Simulation Methods, *1st Edition, Marcel Dekker, Inc.*, 1994.

Yamada F., Sekiguchi K., Tsugeno M., Anbe Y., Andoh Y., Forse C., Guernier M. and Coleman T., Hot Strip Mill Mathematical Models ans Set-Up Calculation, *IEEE Transactions on industry Applications, Vol. 27, No. 1*, January/February 1991, pp.131-139.

Zietsman J.H., Kumar S., Meech J.A., Samarasekera I.V., and Brimacombe J.K., Taper Design In Continuous Billet Casting Using Artificial Neural Networks, *Ironmaking and Steelmaking, 25*(6), 1998, pp.476-483.

Table 1: Données numériques du Réacteur Chimique à Agitation Continue

Concentration de l'alimentation en entrée	C_{a0}	*0.2*
Température du produit en entrée	T_0	*430*
Taux d'écoulement du processus	q	*60*
Volume du réacteur	v	*100*
Constante définissant le taux (la vitesse) de réaction	k_0	$7.2*10^{10}$
Chaleur de la réaction.	ΔH	$-2*10^5$
Chaleurs spécifiques.	$\boldsymbol{C_p = C_{pc}}$	*1*
Densités des liquides	$\rho = \rho_c$	*1000*
Coefficient de transfert de chaleur	h_a	$7*10^5$
Constantes caractéristiques du réacteur	$k_1 = k_0 \frac{\Delta H}{\rho C_p}$ $k_2 = \rho_c \frac{C_{pc}}{\rho C_p v}$ $k_3 = \frac{h_a}{\rho_c C_{pc}}$	

Table 2 : Caractéristiques techniques du train finisseur du laminoir à chaud MITTAL-STEEL, ALGERIE, El-Hadjar

Type du train de laminage finisseur	6 cages quarto équipées avec un système de serrage hydraulique
Puissance nominale de moteurs d'entraînement	2 x 3000 HP, C.C.
Intervalle d'épaisseur de la bande	1.5 à 15 mm
Intervalle de largeur de la bande	600 à 1350 mm
AGC	Hydraulique
Gauge d'épaisseur	Rayon X
Vitesse de laminage	49 à 407 rpm (tours/min)
Module d'élasticité du laminoir	10000 tonnes/mm
Température de laminage	860 to 880 °C

Propriétés physiques de l'acier

- Specific Gravity: 7,9
- Density: 7850 kg/m³
- Melting Point: 1300-1450°C
- Specific Heat: 0.12 cal/g.°C
- Linear Coefficient of Thermal Expansion: 11.7 x 10^{-6} 1/°C
- Volumetric Coefficient of Thermal Expansion: 35.1 x 10^{-6} 1/°C
- Thermal Conductivity at 15,6°C: 58.9 W/m.K
- Electrical Resistivity at 15,6°C: 17 x 10^{-8} ohm.m
- Speed of Sound through Steel: 5490 m/s
- Young's Modulus of Elasticity: 207,000 MPa
- Poisson's Ratio:
  - Elastic Range: 0,3
  - Plastic Range: 0,5
- Bulk Modulus: 159,000 MPa
- Shear Modulus: 83,000 MPa
- Emissivity of Polished Metal Surface:
  - 0.07 @ 38°C
  - 0,10 @ 260°C
  - 0,14 @ 540°C
- Emissivity of Oxidized Steel Plate at 15,6°C: 0,80

Ø Formule de Tselikov pour le calcul du Module de Young de l'acier

. Tselikov

$$E = 308250 + 42924\,C - 144000\,C^2 + 20525\,Si - 5289\,Mn - 12000\,P + 174000\,S - 225{,}6\,T + 0{,}01379\,T^2$$

Notation:

E: Young Modulus [kgf/cm²]
C: C content [weight %]
Mn: Mn content [weight %]
Si: Si content [weight %]
P: P content [weight %]
S: S content [weight %]
T: Temperature [°C]

Note:

- Valid for carbon, alloy and stainless steels between 20 and 900°C.

Source: ROYZMAN, S.E. *Thermal Stresses in Slab Solidification*. **Asia Steel**, 1996, 158-162.

Ø Formule de Misaka pour le calcul de la contrainte de déformation de l'acier

$$\sigma = \exp\left[0.126 - 1.75\,C + 0.594\,C^2 + \frac{\left(2851 + 2968\,C - 1120\,C^2\right)}{T}\right] \varepsilon^{0{,}21} \left(\frac{d\varepsilon}{dt}\right)^{0{,}13}$$

Notation:

σ: Steel Hot Strength [kgf/mm²]
C: C content [weight %]
T: Absolute Temperature [K]
ε: True Strain
t: Time [s]

Source: MISAKA, Y. et al. *Formulatization of Mean Resistance to Deformation of Plain C Steels at Elevated Temperature*. **Journal of the Japan Society for the Technology of Plasticity**, 8, 79, 1967-1968, 414-422.