Contribution à la surveillance des systèmes de production à l'aide des réseaux de neurones dynamiques : Application à la e-maintenance

Ryad Zemouri

Année : **2003** N° ordre : 986

THESE

présentée à

L'UFR des Sciences et Techniques
de l'Université de Franche-Comté

pour obtenir le

GRADE DE DOCTEUR DE L'UNIVERSITE DE FRANCHE-COMTE

en Automatique et Informatique
(Ecole Doctorale Sciences Physiques pour l'Ingénieur et Microtechniques)

Contribution à la surveillance des systèmes de production à l'aide des réseaux de neurones dynamiques : Application à la e-maintenance.

par

Mohamed Ryad ZEMOURI

Soutenue le *28 Novembre 2003* devant la Commission d'examen :

Alain	**BOURJAULT**	**Professeur à l'ENSMM de Besançon**	**Président de jury**
Daniel	**NOYES**	**Professeur à l'ENI de Tarbes**	**Rapporteur**
Denis	**HAMAD**	**Professeur à Université du Littoral Côte d'Opale, Calais**	**Rapporteur**
Jean-Pierre	**THOMESSE**	**Professeur à l'ENSEM-INPL de Nancy,**	**Rapporteur**
Jean-Marc	**FAURE**	**Professeur à l'ISMCM-CESTI de Paris**	**Examinateur**
Noureddine	**ZERHOUNI**	**Professeur à l'ENSMM de Besançon**	**Directeur de thèse**
Daniel	**RACOCEANU**	**Maître de Conférences à l'Université de Franche-Comté**	**Co-Directeur de thèse**
Raphaël	**LABOURIER**	**PDG Sté. AVENSY Ingénierie, Besançon**	**Invité**

Remerciements

Je tiens avant tout à remercier Noureddine ZERHOUNI, professeur à l'Ecole Nationale Supérieure de Mécaniques et Microtechniques de Besançon (ENSMM) de m'avoir accueilli au sein de sa jeune équipe de maintenance et sûreté de fonctionnement. Je lui suis sincèrement reconnaissant pour son encadrement, ses conseils et ses orientations.

Je souhaite remercier très vivement la personne sans laquelle tout ceci n'existerait pas. J'exprime en effet toute ma profonde gratitude à Daniel RACOCEANU, maître de conférences à l'Université de Franche-Comté, non seulement pour son encadrement de très haut niveau, ses précieux conseils et orientations, mais également pour sa disponibilité et son dévouement. Je le remercie de m'avoir toujours poussé vers l'avant, pour toute la confiance qu'il a porté en moi et qui m'a permis d'acquérir une précieuse expérience du métier d'enseignant chercheur. Je le remercie tout simplement pour sa sincère amitié et ses précieuses qualités humaines.

Je remercie messieurs les membres du jury pour la caution qu'ils ont bien voulu apporter à ce travail. J'adresse mes remerciements aux :

- *professeur Alain BOURJAULT de l'ENSMM de Besançon et directeur du Laboratoire d'Automatique de Besançon (LAB). Je le remercie particulièrement de m'avoir accueilli dans son laboratoire et d'avoir accepté de présider le jury de soutenance,*
- *professeur Denis HAMAD de l'Université du Littoral Côte d'Opale, professeur Daniel NOYES de l'ENI de Tarbes et professeur Jean-Pierre THOMESSE de l'ENSEM-INPL de Nancy pour leur travail de lecture critique du manuscrit,*
- *professeur Jean-Marc FAURE de l'ISMCM-CEST de Paris de m'avoir honoré en acceptant d'être examinateur,*
- *monsieur Raphaël LABOURIER Président Directeur Général de la société AVENSY Ingénierie de Besançon pour toute la confiance qu'il a accordée à nos travaux de recherche. Je le remercie personnellement et au nom de l'équipe de maintenance et sûreté de fonctionnement pour sa collaboration, et pour l'ensemble des ressources humaines et matérielles mises à notre disposition durant la phase d'exploitation industrielle.*

Je suis très reconnaissant aux membres de l'équipe ISA du LURPA et particulièrement à Bruno, Olivier et au professeur J.M Faure pour leurs précieux conseils qui m'ont permis de bien préparer mon exposé de soutenance.

Je remercie très sincèrement et sans exception, l'ensemble du personnel du Laboratoire d'Automatique de Besançon pour l'excellente ambiance qui y règne.

Ces années de thèse ont été pour moi l'occasion de connaître des personnes exceptionnelles qui m'ont tout simplement offert leur sincère amitié, et avec qui j'ai partagé d'agréables moments. Je remercie très chaleureusement ***Ivana****,* ***Elena****,* ***Mikky****,* ***Floriana, Lumi, Roberto****,* ***Slava, Alexei*** *et sa femme* ***Natacha****,* ***Nico et*** *enfin* ***Vadime*** *pour tout ce qu'on a vécu ensemble.*

Je remercie bien particulièrement ma petite ***Magda*** *pour tout ce qu'elle fait pour moi.*

Enfin, merci à mes parents pour toute l'éducation qu'ils m'ont inculquée et surtout de m'avoir encouragé et permis de réaliser une thèse de doctorat.

Table des matières

Introduction Générale

Introduction Générale

Le cadre de cette étude

Le rôle premier de la surveillance industrielle est d'augmenter la disponibilité des installations industrielles afin de réduire les coûts directs et indirects de la maintenance des équipements de production. Les coûts directs de cette maintenance sont ceux relatifs aux diverses pièces de rechange, main d'œuvre, etc. Par contre, les coûts indirects sont essentiellement dus au manque à gagner engendré par un arrêt de production. On comprend alors que l'enjeu d'une bonne politique de surveillance est très important pour les entreprises soucieuses d'avoir une meilleure maîtrise des coûts de maintenance.

Actuellement, le domaine de la maintenance a tendance à devenir un marché à part entière et beaucoup d'entreprises se spécialisant dans ce domaine ont été créées ces dernières années. La maintenance est devenue un vrai métier avec ses propres méthodologies et concepts. Parmi les facteurs qui ont favorisé cette nouvelle tendance, le développement des Sciences et Technologies de l'Information et de la Communication (STIC) prend une place très importante. L'essor de ces moyens de communication a provoqué une évolution importante de l'organisation des entreprises. Celles-ci ont tendance de plus en plus à *externaliser* la fonction maintenance pour mieux se concentrer sur leur activité principale. Par ailleurs, les composantes du système de surveillance se caractérisent par une autonomie de plus en plus importante, en travaillant dans des systèmes distribués et en intégrant souvent une intelligence embarquée.

Les méthodologies de surveillance peuvent être divisées en deux grandes catégories : les méthodologies qui se basent sur l'existence d'un modèle formel de l'équipement à surveiller, et les méthodologies qui se basent uniquement sur l'analyse des variables de surveillance ainsi que sur les connaissances a priori des experts humains. Les méthodes qui se basent sur une modélisation de l'équipement sont naturellement tributaires de l'existence ainsi que de la qualité d'une modélisation physique de l'équipement. Ce modèle servira de référence pour un fonctionnement nominal et tout écart par rapport au point de fonctionnement nominal sera synonyme de défaillance. L'inconvénient de ces techniques est l'existence d'incertitudes de modélisation qui sont dues au fait que la modélisation physique ne prend pas en considération tous les paramètres et les aléas qui peuvent influer sur une information d'un paramètre de

surveillance. Ces incertitudes de modélisation sont généralement prises en compte par le modèle d'une manière explicite (additive ou multiplicative).

Si la modélisation de composants est souvent réalisable, un problème délicat concerne la modélisation de toute une machine complexe ou d'un procédé entier. Ceci nous conduit à la remarque suivante : lorsque l'on veut surveiller un équipement (ou un système) sur lequel on ne dispose que de très peu d'informations physiques, on peut se poser la question de savoir s'il serait intéressant de prendre le risque d'investir dans l'élaboration d'un modèle de l'équipement, ou bien d'utiliser les deuxièmes méthodologies qui ne se basent pas sur l'existence d'une modélisation physique.

Les techniques de surveillance sans modèle sont divisées en deux parties. La première partie correspond aux outils statistiques et de traitement du signal qui sont généralement qualifiés d'outils de traitement de bas niveau, parce qu'ils sont en contact direct avec le signal capteur, et ne servent généralement que pour la génération d'alarmes brutes, sans aucune information concernant leur signification. La deuxième partie est celle des techniques dites de haut niveau et qui sont plutôt orientées vers la communication avec l'expert. Celles-ci représentent les techniques de l'*Intelligence Artificielle* (***IA***) et servent comme outil de base pour l'aide à la décision. Leur réponse est donc plus élaborée que celle des techniques de bas niveau. Cette réponse peut être obtenue soit à partir des données brutes venant directement des variables de surveillance, soit à partir des données traitées venant des sorties des traitements de bas niveau. Le rôle que peut jouer un expert humain reste tout de même indispensable si l'on veut concevoir un outil de surveillance avec les techniques de l'Intelligence Artificielle. Parmi les techniques de l'IA utilisées pour la surveillance, nos travaux concernent les *Réseaux de Neurones Artificiels* (***RNA***), qui se démarquent des autres outils par leur capacité d'apprentissage et de généralisation.

Les réseaux de neurones artificiels peuvent être exploités en surveillance selon deux manières différentes : comme outil principal de surveillance ou comme outil qui permet de reconstruire une quantité donnée, par exemple une sortie de capteur. Cette estimation peut servir par exemple à évaluer une variable difficilement mesurable ou inaccessible dont la valeur est importante pour la prise de décision ou bien prédire sa future évolution dans un horizon temporel donné. En surveillance, les réseaux de neurones sont utilisés comme outil de reconnaissance des formes. En effet, le problème de la surveillance peut être vu comme un problème de reconnaissance des formes où les classes correspondent aux différents modes de défaillance du système et les formes représentent l'ensemble des observations ou mesures du système (données qualitatives ou quantitatives).

La surveillance dynamique et la prédiction d'une variable pour la maintenance prédictive nécessitent la prise en compte par le réseau de neurones de la dimension temporelle. Cette prise en compte du passé du signal n'est possible que par des architectures de réseaux de neurones dites temporelles. Les réseaux de neurones statiques sont incapables d'assurer ce genre de traitement. Une détection précoce d'un palier de dégradation ou même l'apprentissage d'une séquence d'évolution d'un système à événements discrets nécessite par

conséquent, une architecture neuronale temporelle (dynamique). C'est dans ce contexte que se positionnent nos travaux de recherches.

Initiée par la mise à notre disposition par IBM (par son intermédiaire Silicon Recognition - France) d'une carte neuronale statique ZISC (Zero Instruction Set Computer), notre problématique de recherche a débouché finalement sur un réseau neuronal dynamique (temporel) appelé *RRFR* (Réseau de neurones Récurrent à Fonctions de base Radiales) qui se trouve au cœur d'un partenariat entre le LAB et une entreprise (PME) bisontine, présente sur le marché de la maintenance industrielle. Cette société dénommée *AVENSY Ingénierie,* développe essentiellement des produits de gestion de production et de suivi de maintenance temps réel accessibles à distance via Internet. Le traitement temps réel est obtenu grâce aux architectures des Automates Programmables Industriels (***API***). L'accessibilité distante via le protocole TCP/IP est obtenue grâce au *Coupleur Ethernet serveur Web* des automates de nouvelle génération.

La collaboration entre le LAB et AVENSY a donné naissance à un outil neuronal de surveillance temps réel entièrement paramétrable (apprentissage) à distance par le serveur Web de l'automate. L'idée d'une éventuelle commercialisation du produit nous a incité à déposer un brevet d'invention.

L'organisation du rapport

Le rapport est organisé en six chapitres qui peuvent être résumés comme suit :

Le premier chapitre est dédié à la surveillance des équipements de production. Les méthodologies de surveillance sont généralement divisées en deux groupes : méthodologies de surveillance *avec modèle* et *sans modèle*. Les premières se basent sur l'existence d'un modèle formel de l'équipement et utilisent généralement les techniques de l'automatique. La deuxième catégorie de méthodologies est plus intéressante dès lors qu'un modèle de l'équipement est inexistant ou difficile à obtenir. Dans ce cas, on utilise les outils de la statistique et de l'Intelligence Artificielle. La fonction surveillance est alors vue comme une application de reconnaissance des formes. Les formes représentent le vecteur d'entrée composé par les différentes données de l'équipement (données mesurables et qualifiables) et les classes correspondent aux différents modes de fonctionnement.

L'objet du deuxième chapitre est la présentation d'un état de l'art sur l'application des réseaux de neurones à la surveillance des systèmes de production. Les avantages les plus importants que l'on peut attribuer à une application de surveillance par réseaux de neurones sont : la modélisation et l'estimation de fonctions non linéaires par apprentissage, la fusion de données, la généralisation et la reconstruction des signaux capteurs. Deux architectures neuronales sont généralement utilisées pour des tâches de surveillance : le Perceptron Multi Couches (PMC) et les Réseaux à base de Fonctions Radiales (RFR). Des différences majeures existent entres ces deux architectures qui ont une représentation globale pour le *PMC* et locale

pour les *RFR*. La représentation locale est plus avantageuse pour la surveillance que la représentation globale. L'un de ces avantages est que contrairement au *PMC*, les *RFR* sont capables de dire « *je ne sais pas* ». Cette caractéristique propre aux approches locales, est indispensable afin d'assurer la sûreté de fonctionnement d'une surveillance industrielle.

La représentation du temps dans les réseaux de neurones représente une caractéristique essentielle dans la perspective d'une surveillance industrielle dynamique. Les réseaux de neurones statiques présentés au deuxième chapitre peuvent offrir des solutions très intéressantes dans des applications de reconnaissance des formes ou approximation de fonctions, mais ne peuvent en aucun cas être appliqués sur des données où le temps joue un rôle déterminant dans la résolution du problème. Le troisième chapitre présente ainsi une synthèse des architectures des réseaux de neurones temporels qui existent en littérature, avec les techniques d'apprentissage appropriées.

Parmi toutes les architectures neuronales étudiées au troisième chapitre, celle qui semble la mieux adaptée aux problématiques de la surveillance dynamique est l'architecture des réseaux de neurones récurrents. Seuls les réseaux de neurones récurrents possèdent une mémoire dynamique interne. L'inconvénient majeur de ces réseaux récurrents est la lourdeur ainsi que la complexité de leur phase d'apprentissage. Pour contourner cet obstacle, une façon simple est de prendre en compte l'aspect temporel d'une manière interne au réseau de neurones par des récurrences locales. Ces récurrence locales sont tolérées uniquement au niveau du neurone lui même, sans trop compliquer l'architecture globale du réseau de neurones. Le quatrième chapitre est donc consacré à l'étude de ces réseaux à représentation locale du temps qui sont appelés architectures *LRGF* : *Locally Recurrent Globally Feedforward* ou architectures *Localement Récurrente Globalement Feedforward*. Le but de ce quatrième chapitre est de démontrer, grâce à des développements mathématiques appuyés par des simulations informatiques, que les architectures *LRGF* possèdent des caractéristiques dynamiques très intéressantes. Deux types d'architectures *LRGF* existent en littérature : une architecture à retour local de la sortie du neurone et une architecture à retour local de l'activation du neurone. Cette étude nous permet de justifier le choix du type de mémoire dynamique afin de proposer une nouvelle architecture d'un réseau *RFR* récurrent. Le Réseau Récurrent à Fonctions de base Radiales que nous proposons profite donc des performances des réseaux *RFR* avec l'aspect dynamique à la fois performant et simple des architectures *LRGF*.

Après avoir présenté au chapitre IV une étude sur les réseaux *LRGF*, nous allons dans le cinquième chapitre évaluer les performances du réseau *RRFR* avec sa mémoire dynamique sur trois types d'applications dynamiques : *la reconnaissance de séquences booléennes* et *réelles*, *la prédiction de séries temporelles* et, enfin, *la reproduction de séries temporelles*. Ces trois types d'applications sont très importants en surveillance dynamique. Nous verrons que l'architecture *LRGF* adoptée permet au réseau *RRFR* d'acquérir des propriétés dynamiques tout en gardant la simplicité et l'efficacité des réseaux *RFR*. La phase de paramétrage du réseau *RRFR* est effectuée par une version améliorée de l'algorithme d'apprentissage des *k-moyennes* que nous proposons. En effet, la version simple présente quelques faiblesses qui

seront mises en évidence à travers quelques tests. La version proposée procure à la phase d'apprentissage une stabilité du résultat avec une plus grande robustesse par rapport à la phase de paramétrage de l'algorithme.

Le dernier chapitre est dédié à la présentation d'une exploitation industrielle innovante sur laquelle a débouché notre étude. L'externalisation de la maintenance commence à prendre une certaine ampleur au sein des entreprises soucieuses de réduire les coûts de la maintenance. Les avantages de cette externalisation sont entre autres : d'un côté une meilleure connaissance du budget de la maintenance donc une meilleure maîtrise des coûts, et d'un autre côté la capacité de se recentrer sur son véritable métier, en confiant cette fonction à des professionnels pouvant ainsi assurer une maintenance distante des moyens de production. La solution que nous développons dans ce sixième chapitre s'encadre tout à fait dans ce contexte de *e-maintenance*. Le réseau *RRFR* développé dans nos travaux de recherche est ainsi structuré en un programme évolutif en langage automate assurant ainsi une surveillance dynamique en temps réel. Le choix d'une architecture *LRGF* comme mémoire dynamique d'un réseau *RFR* est très avantageux pour une telle implémentation. L'apprentissage du réseau *RRFR* est entièrement géré à distance, par connexion TCP/IP. Cette exploitation a fait l'objet d'un prototypage et d'un dépôt de brevet et se trouve actuellement, en phase d'étude de marché dans le cadre d'un projet de collaboration entre le *LAB* et notre partenaire industriel *AVENSY*.

Notations et Abréviations

Notations et Abréviations

RFR	Réseau de neurones à Fonctions de base Radiales
RBF	Radial Basis Function neural network (terme anglophone de RFR)
PMC	Perceptron Multi Couches
MLP	Milti Layer Perceptron (terme anglophone de PMC)
TDRBF	Time Delay Radial Basis Function
TDNN	Time Delay Neural Networks
RRFR	Réseau Récurrent à Fonctions de base Radiales
RCE	Restricted Coulomb Energy *ou* Reilly Cooper Elbaum
DDA	Dynamic Decay Adjustment
Feedforward	Réseau de neurone à propagation avant
GRGF	Globalement Récurrent Globalement Feedforward
LRGF	Localement Récurrent Globalement Feedforward

$f(.)$	Fonction d'activation du neurone, généralement la sigmoïde ayant l'expression suivante : $\frac{1-\exp^{-bx}}{1+\exp^{-bx}}$
ξ_i	Entrée Externe du $i^{ème}$ neurone d'entrée
w_{ij}	Valeur du poids de la connexion du $j^{ème}$ neurone vers le $i^{ème}$ neurone
a_i	L'activation du $i^{ème}$ neurone ($a_i = \sum_j w_{ij}\xi_j$)
o_i	Réponse obtenue du $i^{ème}$ neurone de la couche de sortie
v_i	Réponse obtenue du $i^{ème}$ neurone des couches cachées dans le cas d'un réseau PMC, et représente la réponse obtenue du $i^{ème}$ neurone de tout le réseau dans le cas d'un réseau récurent.
ζ_i	Sortie désirée du réseau de neurone

$E(.)$	Erreur quadratique de sortie ($E = \frac{1}{2}\sum_i (\zeta_i - o_i)^2$)
η	Paramètre de l'apprentissage
θ	Seuil pour l'apprentissage
Ψ	Fonction d'énergie du réseau de Hopfield
$\boldsymbol{\mu}_i$	Vecteur prototype (centre) du neurone gaussien
σ_i	Rayon d'influence (écart type) du neurone gaussien
$\mathbf{x}$	Variable d'entrée du réseau RFR (ou variable aléatoire)
$\phi(.)$	Fonction d'activation des neurones gaussiens ($\phi_i(\mathbf{x}) = e^{-\frac{\|\mathbf{x}-\boldsymbol{\mu}_i\|^2}{\sigma^2{}_i}}$)
k	Représente le $k^{ème}$ prototype (centre) d'un réseau RFR
τ	Retard unitaire
S	Séquence (continue ou discrète)
Z_i	$i^{ème}$ événement d'une séquence S discrète.

Chapitre I

Surveillance des équipements de production

Résumé : *Un système de surveillance comporte deux fonctions de base : détection et diagnostic. Les méthodologies de surveillance sont généralement divisées en deux groupes : méthodologies de surveillance avec modèle et sans modèle. Les premières se basent sur l'existence d'un modèle formel de l'équipement et utilisent généralement les techniques de l'automatique. La deuxième catégorie de méthodologies est plus intéressante dès lors qu'un modèle de l'équipement est inexistant ou difficile à obtenir. Dans ce cas, on utilise les outils de la statistique et de l'Intelligence Artificielle. La fonction surveillance est alors vue comme une application de reconnaissance des formes. Les formes représentent le vecteur d'entrée composé par les différentes données de l'équipement (données mesurables et qualifiables) et les classes représentent les différents modes de fonctionnement.*

Abstract : *Monitoring systems have two basic functions: detection and diagnosis. The monitoring methodologies are generally divided into two groups: monitoring methodologies with and without equipment model. The first ones are based on the existence of equipment's formal model and use generally control system techniques. The second category is more interesting since the formal model is non-existent or difficult to obtain. In that case, we use the statistics tools and the Artificial Intelligence techniques. The monitoring is then seen as a pattern recognition application. The pattern is represented by the input vector which represents the various equipment data and the classes are represented by the various operating modes.*

Chapitre I

Surveillance des équipements de production

I.1. Introduction

Dans un grand nombre d'applications industrielles, une demande croissante est apparue en matière de remplacement des politiques de maintenance curative par des stratégies de maintenance préventive. Cette mutation d'une situation où on « *subit les pannes* » à une situation où on « *maîtrise les pannes* », nécessite quelques moyens technologiques ainsi que la connaissance de techniques d'analyse appropriées. La fonction *surveillance* en continu de l'évolution de l'équipement à travers des données quantifiables et qualifiables permet ainsi de prévenir un dysfonctionnement avant qu'il n'arrive et d'écarter les fausses alarmes qui peuvent ralentir la production (Basseville, 1996). De nombreux auteurs ont abordé le domaine de la surveillance industrielle mettant ainsi en évidence l'intérêt croissant manifesté par la communauté scientifique et les industriels par rapport à cette problématique. Nous pouvons citer sans souci d'exhaustivité les travaux suivants : (Combacau, 1991), (Devauchelle, 1991), (Toguyeni, 1992), (Poulard, 1996), (Cussenot, 1996), (Evsukoff, 1998), (Weber, 1999), (Zhang, 1999), (Combastel, 2000), (Lefebvre, 2000).

L'objectif de ce chapitre est de présenter les techniques les plus courantes en surveillance d'équipements industriels. Dans la littérature associée à ce domaine, on peut trouver plusieurs définitions quelquefois divergentes. C'est pourquoi nous nous positionnons dans la première partie de ce chapitre, en donnant des définitions des mots clés qui sont utiles pour la compréhension de ce rapport.

Les méthodologies de surveillance sont généralement divisées en deux groupes : méthodologies de surveillance *avec modèle* et *sans modèle* (Dash *et al.*, 2000). Les premières se basent sur l'existence d'un modèle formel de l'équipement et utilisent généralement les techniques de l'automatique (Combacau, 1991). La deuxième catégorie de méthodologies est plus intéressante dès lors qu'un modèle de l'équipement est inexistant ou difficile à obtenir. Dans ce cas, on utilise les outils de la statistique et de l'Intelligence Artificielle. La fonction surveillance est alors vue comme une application de reconnaissance des formes. Les formes représentent le vecteur d'entrée composé par les différentes données de l'équipement (données mesurables et qualifiables), et les classes représentent les différents modes de fonctionnement.

Notons que les deux approches Automatiques / Intelligence Artificielle peuvent être combinées pour profiter de certains avantages de chacune et avoir ainsi une certaine complémentarité (Dubuisson, 2001). Les méthodes de l'Automatique[1] sont par nature proches du système surveillé puisqu'elles travaillent directement à partir des données issues des capteurs ; elles sont ainsi principalement utilisées pour la génération d'alarmes. Les méthodes de l'Intelligence Artificielle sont, elles, plus tournées vers la communication avec l'opérateur et se focalisent plus sur la transformation d'un ensemble d'informations brutes et non reliées entre elles en une information interprétable directement par l'opérateur chargé de la conduite ; elles sont donc utilisées pour l'interprétation des alarmes et l'aide à la décision (Basseville *et al.,* 1996). D'autres réflexions sur la complémentarité entre ces deux domaines peuvent être trouvées dans (Dubois *et al.,* 1994). On peut citer également quelques travaux où les deux techniques Automatique/Intelligence Artificielle ont été conjointement utilisées : (Katsillis *et al.,* 1997), (Loiez, 1997), (Washio *et al.,* 1998), (Hines *et al.,* 1995), (Vemuri, 1997), (Vemuri *et al.,* 1998).

I.2. Définitions

La diversité des définitions trouvées dans différents travaux fait que nous avons jugé important d'établir un lexique sur les termes qui seront utiles pour la compréhension du présent rapport. Ces définitions ont été extraites pour certaines à partir des références suivantes : (Villemeur, 1988), (Dubuisson, 1990), (Combacau, 1991), (Toguyeni, 1992), (Zwingelstein, 1995), (Basseville *et al.,* 1996), (Lefebvre, 2000). On peut toutefois trouver en littérature des définitions qui sont complètement différentes de celles que nous proposons, mais ceci nous permet de présenter notre point de vue de la surveillance industrielle.

Dégradation

Une dégradation représente une perte de performances d'une des fonctions assurées par un équipement.

□

Si les performances sont au-dessous du seuil d'arrêt défini dans les spécifications fonctionnelles de cet équipement, il n'y a plus dégradation mais défaillance.

[1] *En particulier les méthodes statistiques du traitement du signal*

Défaillance

Une défaillance est l'altération ou la cessation de l'aptitude d'un ensemble à accomplir sa ou ses fonctions requises avec les performances définies dans les spécifications techniques.

□

On peut classer les défaillances selon leur degré de sévérité par :

- *Défaillance critique* : nécessite une intervention d'urgence,
- *Défaillance significative* : nécessite un processus de traitement,
- *Défaillance absorbable* : pouvant être ignorée dans un premier temps.

Panne

Une panne est l'inaptitude d'une entité (composant ou système) à assurer une fonction requise.

□

Si nous écartons la possibilité d'erreurs de conception, la définition précédente implique que toute défaillance entraîne une panne. La défaillance correspond à un événement et la panne à un état. Sur le plan temporel, la défaillance correspond à une date et la panne à une durée comprise entre la date d'occurrence de la défaillance et la date de fin de réparation.

Mode de fonctionnement

Un système présente généralement plusieurs modes de fonctionnement. On peut observer des modes de plusieurs types parmi lesquels :

- *Mode de fonctionnement nominal* : c'est le mode où l'équipement ou le système industriel remplit sa mission dans les conditions de fonctionnement requises par le constructeur et avec les exigences attendues de l'exploitant.
- *Mode de fonctionnement dégradé* : qui correspond soit à l'accomplissement partiel de la mission, soit à l'accomplissement de celle-ci avec des performances moindre. En d'autres termes, il y a eu dégradation dans l'équipement ou le système mais pas de défaillance.
- *Mode de défaillance* : qui correspond à des mauvais fonctionnements du système, c'est-à-dire qu'il y a eu défaillance soit après dégradation soit défaillance brusque. Un mode de défaillance est caractérisé par les effets causés par cette défaillance. Ces effets peuvent être mesurables ou qualifiables. En faisant une analyse de cause à effet de la défaillance, on peut associer le mode de défaillance à toute cette analyse faite par un expert. En d'autres termes, à chaque mode de défaillance, on associe une décision et une interprétation possible. Chaque équipement ou système peut posséder qu'un seul mode nominal ; par contre, il possède plusieurs modes de défaillance.

Surveillance

*La surveillance est un dispositif **passif**, **informationnel** qui analyse l'état du système et fournit des indicateurs. La surveillance consiste notamment à **détecter** et **classer** les défaillances en observant l'évolution du système puis à les **diagnostiquer** en **localisant** les éléments défaillants et en **identifiant** les causes premières.*

□

La surveillance se compose donc de deux fonctions principales qui sont la *détection* et le *diagnostic*. La Figure 1 montre une architecture générale d'un système de surveillance. Les principales raisons qui conduisent à surveiller un système sont :

- *La conduite :* qu'il s'agit d'optimiser et qui est une tâche en ligne (production maximale, sécurité, non dégradation des équipements). Ceci passe par la surveillance du procédé afin de détecter toute anomalie de fonctionnement et de l'identifier aussi bien que possible. Ce type d'action est aussi appelé supervision : surveillance + conduite.

- *La maintenance :* qui a pour objet d'optimiser le remplacement ou la réparation d'équipements usés ou défectueux. On peut citer trois types de maintenance :

  - *Maintenance corrective :* intervient après la détection et la localisation d'un défaut.

  - *Maintenance préventive :* Maintenance effectuée dans l'intention de réduire la probabilité de défaillance d'un bien ou la dégradation d'un service rendu. C'est une intervention de maintenance prévue, préparée et programmée avant la date probable d'apparition d'une défaillance. Le plus souvent elle est systématique, c'est-à-dire une maintenance préventive effectuée selon un échéancier établi suivant le temps ou le nombre d'unités d'usage.

  - *Maintenance conditionnelle :* alternative à la maintenance systématique, fait l'objet d'une demande croissante dans un grand nombre d'applications industrielles. Cette maintenance est basée sur la surveillance en continu de l'évolution du système, afin de prévenir un dysfonctionnement avant qu'il n'arrive. Elle n'implique pas la connaissance de la loi de dégradation. La décision d'intervention préventive est prise lorsqu'il y a évidence expérimentale de défaut imminent, ou approche d'un seuil de dégradation prédéterminé. Elle impose donc des traitements en ligne, au moins en partie.

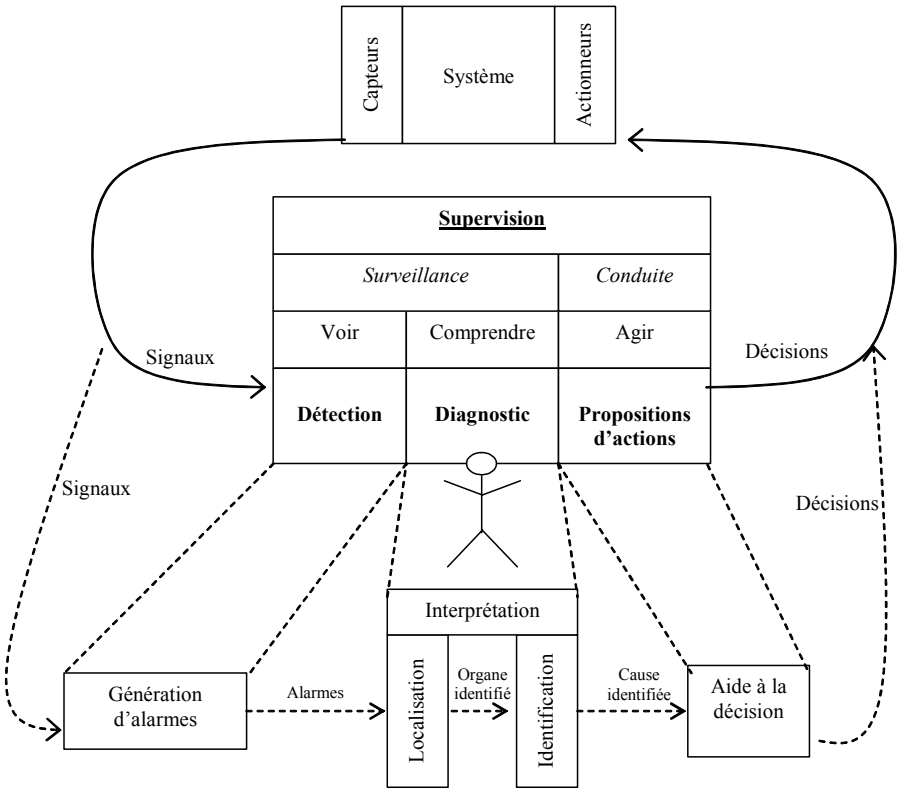


Figure 1. *Architecture générale d'un système de supervision en ligne* (Basseville *et al.,* 1996)

Détection

*Pour **détecter** les défaillances du système, il faut être capable de **classer** les situations observables comme étant **normales** ou **anormales**.*

□

Cette classification n'est pas triviale, étant donné le manque d'information qui caractérise généralement les situations anormales. Une simplification communément adoptée consiste à considérer comme anormale toute situation qui n'est pas normale.

Diagnostic

L'objectif de la fonction diagnostic est de rechercher les causes et de localiser les organes qui ont entraîné une observation particulière.

□

Cette fonction se décompose en deux fonctions élémentaires : *localisation* et *identification*. A partir de l'observation d'un état de panne, la fonction diagnostic est chargée de retrouver la faute qui en est à l'origine. Ce problème est difficile à résoudre. En effet si, pour une faute donnée, il est facile de prédire la panne résultante, la démarche inverse qui consiste à identifier la faute à partir de ses effets, est beaucoup plus ardue. Une défaillance peut

généralement être expliquée par plusieurs fautes. Il s'agit alors de confronter les observations pour fournir la bonne explication (Figure 2).

Localisation

La localisation permet de déterminer le sous-ensemble fonctionnel défaillant.

□

Identification de la cause

Cette dernière étape consiste à déterminer les causes qui ont mené à une situation anormale.

□

Ces causes peuvent être internes (sous-ensembles défaillants faisant partie de l'équipement), ou bien externes à l'équipement.

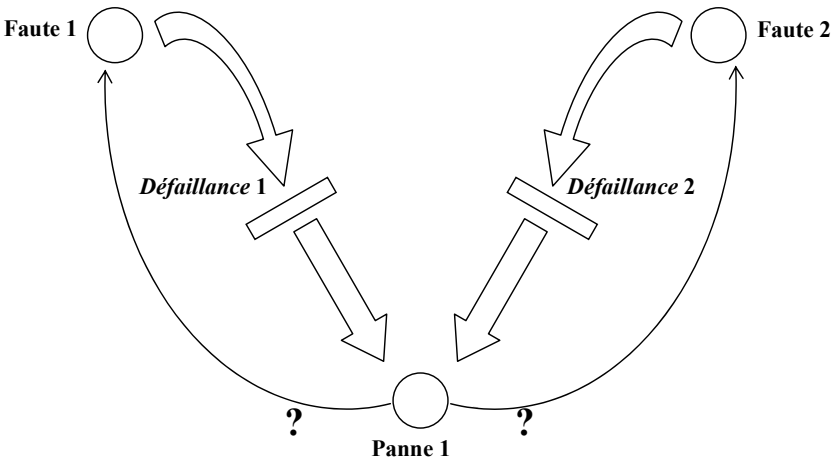


Figure 2. *Difficulté du diagnostic. Deux fautes conduisent à la même panne ce qui complique l'opération inverse, en l'occurrence le diagnostic.*

Surveillance prédictive (dynamique)

Comme pour la surveillance classique, la surveillance prédictive est un dispositif ***passif****,* ***informationnel*** *qui analyse l'état présent et passé du système et fournit des indicateurs sur les tendances d'évolution futur du système. La surveillance prédictive se compose de : la* ***Détection prédictive*** et *du* ***Diagnostic prédictif***

□

Détection prédictive

La détection prédictive consiste à prédire une défaillance future. En d'autres termes, le but de la détection prédictive est de détecter une dégradation (Figure 3) au lieu d'une défaillance, pour le cas de la détection classique.

□

Diagnostic prédictif (Pronostic)

L'objectif du diagnostic prédictif est d'identifier les causes et de localiser les organes qui ont entraîné une dégradation particulière.

□

Pour résumer toutes les définitions que nous avons présentées précédemment, nous illustrons sur la Figure 3 un exemple de *surveillance* d'un équipement industriel. La *dégradation* de l'équipement est caractérisée par toute la période où l'amplitude des vibrations croit sans que le signal n'atteigne le seuil d'alarme. La *détection* du franchissement de ce seuil provoque une génération d'alarme synonyme d'un événement de *défaillance*. L'équipement se trouve alors dans une situation de *panne*. Un *diagnostic* permet de *localiser* l'organe de l'équipement qui est à l'origine de ces vibrations (Tige A) et *d'identifier* la cause qui a provoquée ces vibrations (un desserrement d'un boulon). Mettre en place un système de surveillance prédictive consiste donc à pouvoir détecter la dégradation avant l'événement défaillance par une génération de pré-alarme. Le diagnostic prédictif devra prédire que c'est la tige A qui vibre à cause du desserrement d'un boulon.

Après avoir présenté les éléments de base du domaine concerné par nos contributions, nous considérons utile d'approfondir les techniques les plus importantes liées à la surveillance industrielle.

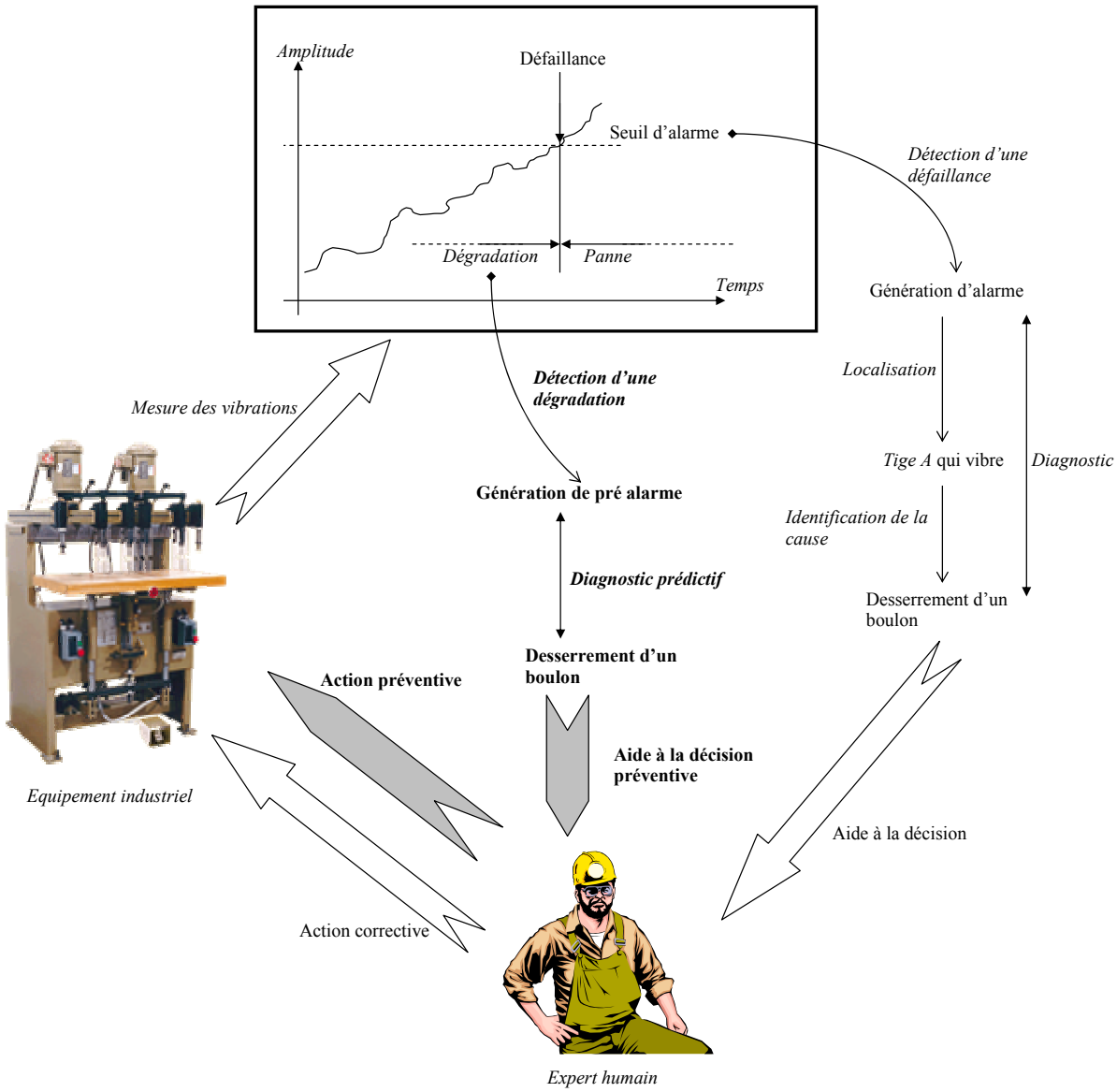


Figure 3. *Exemple d'une boucle de supervision (surveillance+action) d'un équipement industriel avec toutes les notions que nous avons définit précédemment. La surveillance préventive permet d'anticiper sur tout le traitement de détection et diagnostic avant que le signal n'atteigne le seuil d'alarme.*

I.3. Méthodes de surveillance

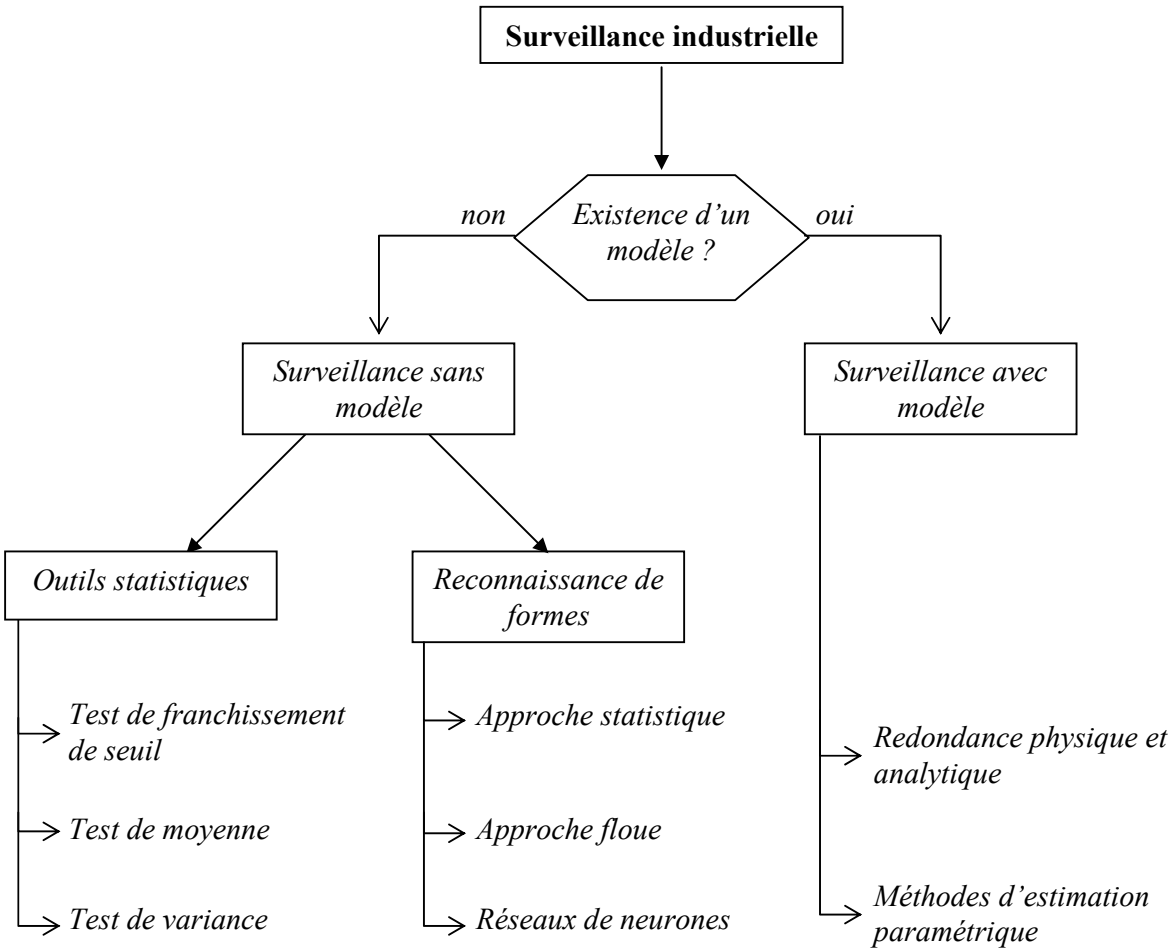


Figure 4. *Classifications des méthodologies de surveillance industrielle.*

Les méthodes de surveillance industrielle telle qu'elles sont présentées dans ce paragraphe sont illustrées sur la Figure 4. L'existence d'un modèle formel ou mathématique de l'équipement détermine la méthode de surveillance utilisée. La surveillance avec modèle se compose essentiellement de deux techniques : méthodes de redondance physique et analytique et méthodes d'estimation paramétrique. D'un autre côté, les méthodes qui ne se basent pas sur l'existence du modèle se divisent en deux catégories : méthodes utilisant des outils statistiques et méthodes de reconnaissance des formes. Les outils statistiques établissent des tests sur les signaux d'acquisition. Ces tests ne sont capables d'assurer que la fonction détection de défaillances. Par contre, les techniques de surveillance par reconnaissance des formes sont plus élaborées par rapport aux simples tests statistiques et sont capables de détecter et de diagnostiquer les défaillances.

I.3.1. Méthodes de surveillance avec modèles

Les méthodes de surveillance avec modèle ont pour principe de comparer les mesures effectuées sur le système aux informations fournies par le modèle (Frank, 1990). Tout écart est alors synonyme d'une défaillance. Les outils de la théorie de la décision sont ensuite utilisés pour déterminer si cet écart est dû à des aléas normaux comme, par exemple, le bruit de mesure ou s'il traduit une défaillance du système. Ces méthodes peuvent être séparées en deux techniques : techniques de redondance physique et analytique et techniques d'estimation paramétrique. Ces deux techniques seront présentées brièvement. Toutefois, pour plus de détails, nous renvoyons le lecteur aux références suivantes : (Willsky, 1976), (Isermann, 1984), (Basseville, 1988), (Gertler, 1988), (Patton et al., 1989), (Frank, 1990), (Combacau, 1991), (Basseville et al., 1993), (Cussenot, 1996), (Gertler, 1998), (Weber, 1999), (Tromp, 2000), (Combastel, 2000).

I.3.1.1. Redondances physiques et analytiques

a) Redondances physiques

Afin de fiabiliser la détection des défaillances à partir des signaux mesurés, il faut un moyen pour distinguer les défaillances capteurs des défaillances système. La méthode la plus simple consiste à utiliser la redondance physique. Il s'agit de doubler ou tripler des composantes de mesure du système. Si ces composantes identiques placées dans le même environnement émettent des signaux identiques, on considère que ces composants sont dans un état de fonctionnement nominal et, dans le cas contraire, on considère qu'une défaillance capteur s'est produite dans au moins une des composantes (Zhang, 1999). Cette méthode par redondance physique a l'avantage d'être conceptuellement simple mais est coûteuse à être mise en œuvre et conduit à des installations encombrantes. Elle est, par conséquent, utilisée uniquement pour la surveillance des sous-ensembles critiques d'un système. Un autre inconvénient est que les composantes identiques fabriquées dans la même série peuvent se dégrader de la même façon et tomber en panne en même temps. Pour pallier ce dernier inconvénient, on peut utiliser des composantes différentes qui remplissent la même fonction.

b) Redondances analytiques

Les méthodes de redondance analytique nécessitent un modèle du système à surveiller. Ce modèle comprend un certain nombre de paramètres dont les valeurs sont supposées connues lors du fonctionnement nominal. Dans la mesure où la surveillance est établie à partir des mesures échantillonnées des grandeurs observables du système, la modélisation de ce dernier sous forme discrète semble être raisonnable. De plus, dans le cas où le système présente un

caractère non linéaire, il est possible, afin de disposer d'un modèle plus simple, d'opérer une linéarisation autour d'un point de fonctionnement.

Le modèle d'espace d'état discret échantillonné relie le vecteur d'état *x(k)* au vecteur d'entrée *u(k)* et au vecteur de sortie *y(k)* du système à surveiller par l'intermédiaire des matrices *A*, *B* et *C* (indépendante du temps) sous la forme :

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \qquad [1]$$

Les défaillances et les perturbations qui peuvent survenir dans le procédé peuvent être alors modélisées à partir de ces équations. Les fautes $F_p(k)$ et le bruit $b_p(k)$ du procédé sont représentés de façon additive :

$$x(k+1) = Ax(k) + Bu(k) + PF_p(k) + Qb_p(k) \qquad [2]$$

Les erreurs de mesures $F_u(k)$ et $F_y(k)$ des entrées et des sorties ainsi que les bruits $b_u(k)$ et $b_y(k)$ sont modélisés par les relations :

$$\begin{aligned} \tilde{u}(k) &= u(k) + F_u(k) + b_u(k) \\ \tilde{y}(k) &= y(k) + F_y(k) + b_y(k) \end{aligned} \qquad [3]$$

où $\tilde{u}(k)$ représente en fait la réalisation du signal de commande $u(k)$ par les actionneurs et $\tilde{y}(k)$ la mesure du signal $y(k)$.

Les différences entre les matrices de paramètres du modèle et celles du système réel se traduisent par :

$$\tilde{A} = A + \Delta A(k), \tilde{B} = B + \Delta B(k), \tilde{C} = C + \Delta C(k) \qquad [4]$$

Le but des méthodes de redondance analytique est d'estimer l'état du système afin de le comparer à son état réel. L'estimation de l'état du système peut être réalisée soit à l'aide de techniques d'estimation d'état, soit par obtention de relations de redondance analytique.

Le but des techniques d'estimation d'état est de reconstruire au moyen d'observateur, les états et les sorties du système, à partir des entrées et des sorties mesurées (Frank, 1990). On disposera donc d'une estimation du vecteur d'état et du vecteur de sortie du système, vecteur qui correspond généralement aux grandeurs mesurables. Ces sorties estimées sont alors comparées aux sorties réelles et tout écart est révélateur d'une défaillance. La théorie de la

décision est ensuite utilisée pour déterminer si l'écart observé est dû à des aléas normaux du fonctionnement ou à des défaillances.

Les relations de redondance analytique sont utilisées lorsque le modèle fait intervenir des grandeurs mesurables. Les relations de redondance analytique sont des relations entre les variables disponibles du système, prises dans une fenêtre temporelle. La redondance directe est la méthode la plus simple pour éliminer le vecteur d'état *x(k)*. Elle se produit parmi les capteurs qui mesurent les grandeurs qui sont reliées par les relations algébriques du modèle. C'est-à-dire que ces grandeurs sont reliées de façon à ce que la grandeur que mesure un capteur puisse être déterminée par les valeurs instantanées délivrées pas les autres capteurs.

Dans les deux méthodes (estimation d'état et relations de redondance analytique) on dispose d'une estimation de l'état du système. La comparaison avec son état réel fournit alors une quantité appelée *résidu* qui va servir à déterminer si le système est dans un état défaillant ou non. Un résidu idéal doit rester à zéro en absence de panne et s'éloigner de zéro en présence de panne. A cause des erreurs de modélisation et des bruits de mesures, un résidu réel est souvent différent de zéro. Pour les pannes additives dans les systèmes d'états linéaires à paramètres constants dans le temps, la génération et l'évaluation de résidus ont été largement étudiées, tant du point de vue déterministe que stochastique (Willsky, 1976), (Patton *et al.,* 1989), (Frank, 1990), (Basseville *et al.,* 1993), (Basseville, 1997), (Gertler, 1998), (Chen *et al.,* 1999). En revanche, en ce qui concerne les pannes non additives, même pour les systèmes d'état linéaires, les résultats connus sont moins abondants (Zhang, 1999). La situation est encore moins florissante pour les systèmes non linéaires. Toutefois, pour surveiller les pannes d'amplitude faible, une démarche générale a été développée à l'IRISA[2] (Zhang, 1999) qui, s'appuyant sur une approche locale, permet de concevoir des algorithmes pour la génération de résidus à partir des fonctions d'estimation et pour leur évaluation. Elle s'applique à une large classe de systèmes non linéaires avec des pannes additives ou non.

I.3.1.2. Méthodes d'estimation paramétrique

Les méthodes d'estimation paramétrique supposent l'existence d'un modèle paramétrique décrivant le comportement du système et que les valeurs de ces paramètres en fonctionnement nominal soient connues. Elles consistent alors à identifier les paramètres caractérisant le fonctionnement réel, à partir de mesures des entrées et des sorties du système (Willsky, 1976). On dispose ainsi d'une estimation des paramètres du modèle, effectuée à partir des mesures prises sur le système et de leurs valeurs théoriques. Pour détecter l'apparition de défaillances dans le système, il faut effectuer la comparaison entre les paramètres estimés et les paramètres théoriques. Comme pour les méthodes de redondance analytique, la théorie de la décision sert alors à déterminer si l'écart observé est dû à des aléas normaux du fonctionnement ou à des défaillances. La différence entre les méthodes de redondance analytique et les méthodes

[2] *Institut de Recherche en Informatique et Systèmes Aléatoires.*

d'estimation paramétrique est qu'on effectue, pour les premières, la comparaison entre l'état estimé et l'état théorique du système, alors que pour les secondes, on compare les paramètres estimés aux paramètres théoriques du système.

La procédure générale d'estimation paramétrique pour la surveillance peut être décrite en 5 étapes (Isermann, 1984) :

- établissement du modèle mathématique du procédé dans les conditions normales de fonctionnement, à partir de considérations théoriques :

$$y(t) = f(u(t); \boldsymbol{\theta}) \qquad [5]$$

dans lequel *u(t)* et *y(t)* désignent respectivement les entrées et les sorties du système et $\boldsymbol{\theta}$ représente le vecteur des paramètres du modèle.

- Détermination des relations entre les paramètres physiques du modèle $\boldsymbol{\theta}$ et les paramètres physiques du procédé $\mathbf{p}$:

$$\boldsymbol{\theta} = g(\mathbf{p}) \qquad [6]$$

dans lequel $\mathbf{p}$ désigne les constantes physiques du système, supposées connues, qui sont modifiées lorsqu'une défaillance survient.

- Estimation $\hat{\boldsymbol{\theta}}$ des paramètres $\boldsymbol{\theta}$ du modèle à l'aide de l'équation [5] et à partir des mesures des entrées *u(t)* et des sorties *y(t)* du système en fonctionnement réel

$$\hat{\boldsymbol{\theta}}(t) = h(y(1)...y(t); u(1)...u(t)) \qquad [7]$$

- Estimation $\hat{\mathbf{p}}$ des paramètres physiques $\mathbf{p}$ du système à partir de la relation [6]

$$\hat{\mathbf{p}}(t) = g^{-1}(\hat{\boldsymbol{\theta}}(t)) \qquad [8]$$

- Détermination de la présence d'une défaillance ou pas. Elle se fait soit à partir de la comparaison entre les paramètres théoriques connus $\boldsymbol{\theta}$ du modèle et ceux estimés lors du fonctionnement réel $\hat{\boldsymbol{\theta}}$, soit en comparant les paramètres physiques $\mathbf{p}$ connus du système à ceux estimés lors du fonctionnement réel $\hat{\mathbf{p}}$. Cette détermination fait également appel à la théorie de la décision.

Les méthodes d'estimation paramétrique requièrent donc l'élaboration d'un modèle dynamique précis du système à surveiller. Ceci restreint leur utilisation à des procédés bien

définis. Nous pouvons citer à titre d'exemple les travaux réalisés par *Desforges* (Desforges, 1999) qui se basent exactement sur la technique d'estimation paramétrique. Un réseau de neurones sert à estimer les paramètres physiques d'une machine outil à partir de la mesure du courant, tension et vitesse de rotation de l'axe de la machine outil. Les valeurs estimées sont utilisées comme base pour la détection et le diagnostic des défaillances.

I.3.2. Méthodes de surveillance sans modèles

Nombreuses sont les applications industrielles dont le modèle est difficile, voire impossible à obtenir suite à une complexité accrue ou à de nombreuses reconfigurations intervenants durant le processus de production. Pour ce type d'applications industrielles, les seules méthodes de surveillance opérationnelles sont celles sans modèle. Deux solutions existent dans ce cas : surveillance avec des tests statistiques et surveillance par reconnaissance des formes. La première technique est moins élaborée que la deuxième dans le sens où elle ne remplit qu'une partie de la surveillance, à savoir la détection des défaillances. Nous détaillerons donc un peu plus la partie surveillance par reconnaissance des formes. Trois approches sont alors utilisées : approche probabiliste, approche floue et approche neuronale (Dubuisson *et al.,* 2001).

I.3.2.1. Surveillance avec outils statistiques

Les outils statistiques de détection de défaillances consistent à supposer que les signaux fournis par les capteurs possèdent certaines propriétés statistiques. On effectue alors quelques tests qui permettent de vérifier si ces propriétés sont présentes dans un échantillon des signaux mesurés de taille *n* (appelé fenêtre d'observation glissante). On considère que le signal mesuré est une variable aléatoire notée par γ. Nous ne présentons que trois tests statistiques, mais une grande variété de tests, applicables sur un échantillon de mesures, peut être trouvée dans (Basseville, 1988).

a) Test de franchissement de seuils

Le test le plus simple est de comparer ponctuellement les signaux avec des seuils préétablis. Le franchissement de ce seuil par un des signaux capteurs génère une alarme. On peut trouver dans l'industrie deux types de seuils. Un premier type est dit seuil de pré-alarme qui permet d'entreprendre une action de maintenance préventive ; le second type est le seuil d'alarme qui impose l'arrêt de la production et l'engagement d'une action de maintenance corrective. Ce type de méthode est très simple à mettre en œuvre mais ne permet pas d'établir

un diagnostic des défaillances. Cette méthode est aussi très sensible aux fausses alarmes (Figure 5).

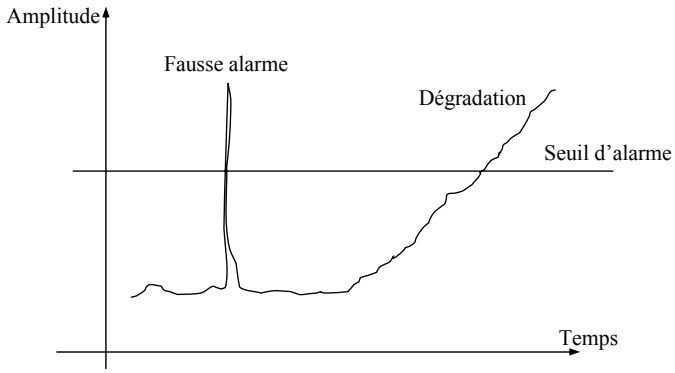


Figure 5. *Sensibilité de la méthode à franchissement de seuils aux fausses alarmes.*

b) Test de moyenne

Contrairement à la méthode précédente, le test de comparaison est effectué sur la moyenne $\hat{y}$ du signal contenu dans une fenêtre de *n* valeurs plutôt que sur une valeur ponctuelle :

$$\hat{y} = \frac{1}{n} \sum_{i=t-n+1}^{t} y_i \qquad [9]$$

Ceci rejoint le principe du calcul des tendances, une des techniques les plus simples de la maintenance prédictive.

c) Test de variance

On peut également calculer la variance d'un signal. Tant que cette variance se situe dans une bande située autour de sa valeur nominale, l'évolution du système est supposée normale. La variance de l'échantillon est définie par :

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=t-n+1}^{t} (y_i - \hat{y})^2 \qquad [10]$$

I.3.2.2. Surveillance par reconnaissance des formes

L'approche de surveillance par reconnaissance des formes permet d'associer un ensemble de mesures (continues ou discrètes) effectuées sur le système à des états de fonctionnement connus. Cette fonction permet d'avoir une relation d'un espace caractéristique vers un espace de décision, de façon à minimiser le risque de mauvaise classification. Trois techniques de reconnaissance des formes sont présentées. La première technique présentée est une technique classique de discrimination basée sur les outils de la probabilité. Cette technique peut se montrer insuffisante car elle suppose une connaissance *a priori* de tous les états de fonctionnement et ne prend pas en compte l'évolution du système (Denoeux *et al.,* 1998). Les deux autres techniques de discrimination qui seront présentées reposent sur la théorie de l'intelligence artificielle. Ces techniques ont l'avantage de ne pas se baser sur les connaissances *a priori* des états de fonctionnement mais plutôt sur une phase d'apprentissage. Ces deux techniques sont la reconnaissance des formes par la logique floue et la reconnaissance des formes par réseaux de neurones.

a) Reconnaissance des formes par outils statistiques

Le formalisme général de ces techniques de reconnaissance des formes est de devoir décider parmi *M* classes pour tout vecteur d'entrée **x** (vecteur forme). Les classes sont désignées par $\alpha_1, \alpha_2, ..., \alpha_M$. On utilise alors une application *d* qui associe un scalaire *i* à chaque vecteur d'entrée **x**:

d(**x**) = *i*, où **x** est associé à la classe α_i avec $i = 1,..,M$

Les probabilités *a priori* $\Pr(\alpha_i)$ des classes $\alpha_1, \alpha_2, ..., \alpha_M$ sont connues. Deux cas sont possibles :

- Toutes les classes sont connues et dans ce cas on obtient la somme de toutes les probabilités égale à un (équation [11]). Cette situation est appelée cas d'un *monde fermé*.

$$\sum_{i=1}^{M} \Pr(\alpha_i) = 1 \qquad [11]$$

- Dans le deuxième cas, toutes les classes ne sont pas connues. On utilise alors une classe α_0 appelée classe de rejet en distance pour combler le manque d'information sur le problème. Cette classe représente donc le mélange de toutes les autres classes non identifiées par l'utilisateur. C'est ce dernier qui lui associera une probabilité, en fonction de son degré de connaissance ou d'ignorance du système. Ce cas est qualifié de *monde ouvert*. On obtient donc la relation suivante :

$$\sum_{i=0}^{M} \Pr(\alpha_i) = 1 \qquad [12]$$

La loi de probabilité de **x** dans chaque classe $\alpha_1, \alpha_2, ..., \alpha_M$ est supposée connue et caractérisée par les lois conditionnelles : $\varphi(\mathbf{x}/\alpha_i)$ i = 1,..*M*. La loi du vecteur **x,** quand on ne connaît pas sa classe d'appartenance, est donnée par la loi mélange $\varphi(\mathbf{x})$:

$$\varphi(\mathbf{x}) = \sum_{i=0}^{M} \Pr(\alpha_i)\varphi(\mathbf{x}/\alpha_i) \qquad [13]$$

La qualité de la décision des classes d'affectation de **x** est quantifiée par un coût de décision, $C(i,\alpha_j)$ (Duda, 1973), coût de décider α_i quand α_j est la vraie classe de **x** (*i,j=0,..,M*). La décision peut être aussi qualifiée par un autre coût qui peut jouer un rôle particulier : c'est le coût entraîné par le fait de ne pas prendre de décision. Ce coût est appelé rejet d'ambiguïté. Sa valeur est égale à (-1) quelle que soit la vraie classe α_i de **x** : $C(-1,\alpha_i)$. La décision finale est jugée par un indicateur de performance, appelé *risque moyen de décision R*, dont on cherchera à obtenir la valeur minimal (Fukunaga, 1990).

Pour un vecteur donné **x**, le risque *R* est associé à la décision d(**x**) :

$$R(\mathbf{x}) = \sum_{j=0}^{M} C(\mathrm{d}(\mathbf{x}),\alpha_j)\Pr(\alpha_j/\mathbf{x}) \qquad [14]$$

où $\Pr(\alpha_j/\mathbf{x})$ représente la probabilité *a posteriori* de la classe α_j :

$$\Pr(\alpha_j/\mathbf{x}) = \frac{\Pr(\alpha_j)\varphi(\mathbf{x}/\alpha_j)}{\varphi(\mathbf{x})} \qquad [15]$$

R peut être moyenné pour tous les vecteurs **x** : On obtient ainsi le risque moyen $\hat{R}$

$$\hat{R} = \int R(\mathbf{x})\varphi(\mathbf{x})d\mathbf{x} \qquad [16]$$

La règle minimisant ce critère est appelée *règle de Bayes* ou *règle du risque minimum* (Fukunaga, 1990). Cette règle consiste à choisir la décision d(**x**) qui minimise [14] et [16].

Pour fixer les coûts des décisions en surveillance, on adopte souvent la procédure suivante :

$$C(i,\alpha_j)=\begin{cases}1 & i\neq j\\ 0 & i=j\end{cases} \qquad i,j=0,M$$
$$C(-1,\alpha_j)=a \qquad [17]$$

L'équation [14] devient alors :

- si d(**x**) = -1

$$R(\mathbf{x})=R_a(\mathbf{x})=a \qquad [18]$$

- si d(**x**) = i

$$R(\mathbf{x})=R_i(\mathbf{x})=1-\Pr(\alpha_i/\mathbf{x}) \qquad [19]$$

La règle de décision est alors :

- soit le vecteur **x** est associé à une des classes connues α_i

$$\Pr(\alpha_i/\mathbf{x})=\max_{j=0,M}\Pr(\alpha_j/\mathbf{x}) \quad \text{et} \quad \Pr(\alpha_i/\mathbf{x})\geq 1-a \qquad [20]$$

- soit le vecteur **x** est rejeté en ambiguïté entre deux ou plusieurs classes connues

$$\Pr(\alpha_i/\mathbf{x})=\max_{j=0,M}\Pr(\alpha_j/\mathbf{x}) \quad \text{et} \quad \Pr(\alpha_i/\mathbf{x})<1-a \qquad [21]$$

On préfère dans ce cas ne pas le classer plutôt que de risquer de commettre une erreur.

- soit le vecteur **x** est rejeté en distance comme n'appartenant à aucune des classes connues

$$\Pr(\alpha_0/\mathbf{x})=\max_{j=0,M}\Pr(\alpha_j/\mathbf{x}) \quad \text{et} \quad \Pr(\alpha_0/\mathbf{x})\geq 1-a \qquad [22]$$

Le vecteur **x** est considéré comme membre d'une nouvelle classe.

- soit le vecteur **x** est rejeté en ambiguïté entre une ou plusieurs classes connues et la classe de rejet en distance

$$\Pr(\alpha_0/\mathbf{x})=\max_{j=0,M}\Pr(\alpha_j/\mathbf{x}) \quad \text{et} \quad \Pr(\alpha_0/\mathbf{x})<1-a \qquad [23]$$

On préfère dans ce cas ne pas prendre de décision plutôt que d'associer le vecteur **x** à une classe ou de le rejeter en distance comme membre d'une nouvelle classe.

Cas gaussien

La plupart du temps, on ne dispose pas de loi de probabilité des vecteurs formes dans chaque classe. On peut toutefois considérer que chaque vecteur **x** obéit à une loi de Gauss dans chaque classe, loi dont on ignore les paramètres. Les paramètres de ces lois doivent être estimés. On doit alors disposer d'échantillons de vecteurs pour chaque classe que l'on prend en considération dans le système de décision.

La densité de probabilité d'une loi gaussienne s'écrit :

$$\varphi(\mathbf{x}/\alpha_i) = \frac{1}{\sigma^2 (2\pi)^{d/2}} \exp\left(-\frac{1}{2\sigma^2} (\mathbf{x}-\boldsymbol{\mu}_i)(\mathbf{x}-\boldsymbol{\mu}_i)^t \right) \qquad [24]$$

où $\boldsymbol{\mu}_i$ est le vecteur espérance mathématique de la classe α_i :

$$\boldsymbol{\mu}_i = E(\mathbf{x}) \quad \text{avec} \quad \mathbf{x} \in \alpha_i \qquad [25]$$

et $\sigma^2{}_i$ représente la variance de la classe α_i :

$$\sigma^2{}_i = E\left((\mathbf{x}-\boldsymbol{\mu}_i)(\mathbf{x}-\boldsymbol{\mu}_i)^t \right) \quad \text{avec} \quad \mathbf{x} \in \alpha_i \qquad [26]$$

Si l'on prend un cas monodimensionnel, on peut représenter les quatre règles de décision citées précédemment (équations [20], [21], [22], [23]) sur la Figure 6.

Comme nous l'avons dit précédemment, les paramètres des distributions gaussiennes ne sont pas a priori connus. Ces paramètres sont le vecteur espérance mathématique $\boldsymbol{\mu}_i$ et la variance $\sigma^2{}_i$. Il faut donc disposer d'un échantillon de vecteurs indépendant pour chaque classe. Soient donc $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ cet échantillon appartenant à la classe α_i. Un des moyens de déterminer ces paramètres est l'estimateur du maximum de vraisemblance. On obtient alors :

$$\hat{\boldsymbol{\mu}}_i = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}_j \qquad [27]$$

ainsi que la variance avec l'estimateur sans biais :

$$\widehat{\sigma^2{}_i} = \frac{1}{N-1} \sum_{j=1}^{N} (\mathbf{x}_j - \widehat{\boldsymbol{\mu}_i})(\mathbf{x}_j - \widehat{\boldsymbol{\mu}_i})^t \qquad [28]$$

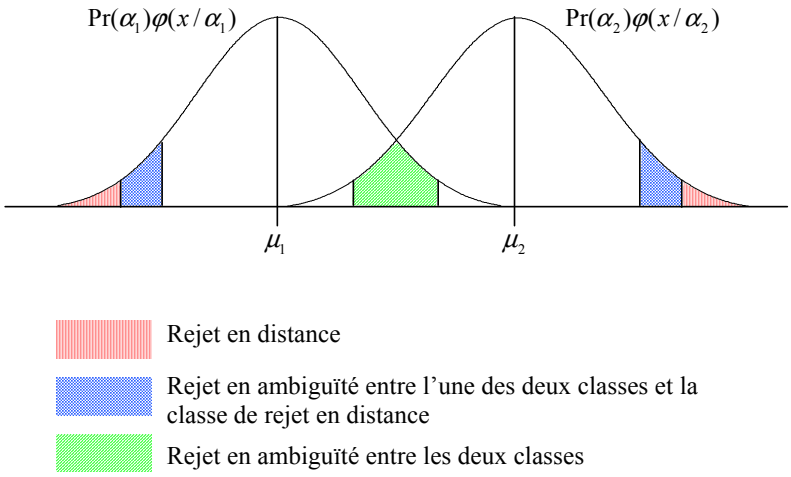


Figure 6. Représentation des différentes zones de deux distributions gaussiennes. Le rejet en distance aux deux extrémités des gaussiennes traduit l'équation [22]. *La zone d'intersection des deux gaussiennes représente une situation de rejet d'ambiguïté, traduite par l'équation* [21]. *Un autre rejet d'ambiguïté entre l'une des deux classes et la classe de rejet en distance est traduit par l'équation* [23]. *Le reste de la zone représente l'appartenance de x à l'une des deux classes.*

b) Reconnaissance des formes par une approche floue

En reconnaissance des formes par approche floue, les classes sont représentées par des sous-ensembles flous. Une fonction d'appartenance quantifie le degré d'appartenance $\lambda_i(\mathbf{x})$ de chaque vecteur **x** à la classe α_i. Généralement, on donne pour chaque vecteur **x** l'ensemble des degrés d'appartenance à toutes les classes $(\lambda_1(\mathbf{x}),....\lambda_M(\mathbf{x}))$. La mise en œuvre d'une méthode de classification floue implique deux étapes : la construction des fonctions d'appartenance et la définition des règles de décision.

Définition des fonctions d'appartenance (fuzzification)

L'un des premiers algorithmes proposés pour construire automatiquement des fonctions d'appartenance dites aussi partition floue, est l'algorithme (*Fuzzy k-Means*) ou algorithme des centres mobiles flou, introduit par (Dunn, 1974) et (Bezdek, 1974). Cet algorithme non supervisé consiste à minimiser itérativement un critère en fonction d'une matrice de partition floue $U = \left[\lambda_k(\mathbf{x}_i)\right]_{(k=1,M;i=1,N)}$ et $V = (\boldsymbol{\mu}_1,....\boldsymbol{\mu}_M)$ de la forme :

$$J_m(U,V) = \sum_{i=1}^{N}\sum_{k=1}^{M} \lambda_k(\mathbf{x}_i)^m d_k(\mathbf{x}_i)^2 \qquad [29]$$

avec les conditions suivantes :

$$\lambda_k(\mathbf{x}_i) \in [0,1] \quad \forall i,k, \qquad [30]$$

$$\sum_{k=1}^{M} \lambda_k(\mathbf{x}_i) = 1 \quad \forall i, \qquad [31]$$

$$0 < \sum_{i=1}^{N} \lambda_k(\mathbf{x}_i) < N \quad \forall k, \qquad [32]$$

où $d_k(\mathbf{x}_i)^2 = \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$ représente la distance euclidienne entre le vecteur d'entrée $\mathbf{x}_i$ et le prototype $\boldsymbol{\mu}_k$ (ou noyau) de la classe α_k, m est un paramètre appelé *fuzzyfier* ($m \geq 1$). L'ensemble des vecteurs d'apprentissage est constitué de N vecteurs $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ susceptibles d'appartenir à M classes $\{\alpha_1, \alpha_2, ..., \alpha_M\}$.

La solution qui minimise J_m ([29]) est donnée par les deux conditions suivantes :

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} \lambda_k(\mathbf{x}_i)^m \mathbf{x}_i}{\sum_{i=1}^{N} \lambda_k(\mathbf{x}_i)^m} \quad \forall k \qquad [33]$$

$$\lambda_k(\mathbf{x}_i) = \frac{1}{\sum_{j=1}^{M} \left(d_k(\mathbf{x}_i) / d_j(\mathbf{x}_i) \right)^{2/(m-1)}} \qquad [34]$$

Les prototypes ainsi que les fonctions d'appartenance sont calculés d'une manière itérative :

- Initialisation de la matrice de partition floue U^0, $t = 0$;
- Faire
  - $t \leftarrow t+1$
  - Calcul de la matrice des prototypes V^t avec l'équation [33]
  - Mise à jour de la matrice de partition floue U^t avec l'équation [34]
- Jusqu'à $\|U^t - U^{t-1}\| \leq \varepsilon$

Après la convergence de l'algorithme, pour chaque vecteur d'entrée **x**, on calcule la matrice de partition floue $U = [\lambda_k(\mathbf{x})]_{(k=1,M)}$ avec l'équation [34].

Décision à partir des degrés d'appartenance (défuzzification)

Ayant calculé la matrice de partition floue $U = [\lambda_k(\mathbf{x})]_{(k=1,M)}$ qui représente donc les degrés d'appartenance du vecteur **x** aux différentes classes, il reste à en déduire le choix d'une action $\gamma(\mathbf{x})$. Une étude comparative entre plusieurs règles de décision a été réalisée par (Masson *et al.,* 1996). Le cas le plus simple serait d'avoir chaque action $\gamma_k(\mathbf{x})$ qui représente l'affectation du vecteur **x** à la classe α_k. Dans ce cas, on pourrait appliquer le principe du *maximum d'appartenance* (Pal, 77), qui consiste à choisir la classe ayant le plus haut degré d'appartenance :

$$\gamma(\mathbf{x}) = \gamma_k(\mathbf{x}) \qquad si \qquad \lambda_k(\mathbf{x}) = \max_{j=1,M} \lambda_j(\mathbf{x}) \qquad [35]$$

Afin de prendre en compte les notions de rejet en distance et d'ambiguïté présentées précédemment, il est possible d'utiliser un seuil θ_k pour chaque classe. Ce seuil est soit défini a priori, soit déterminé à partir de l'ensemble d'apprentissage :

$$\theta_k = \min_{\mathbf{x}_i \in \alpha_k} \lambda_k(\mathbf{x}_i) \qquad [36]$$

Si l'on considère donc $A = \{\gamma_0, \gamma_d, \gamma_1, ...\gamma_M\}$ l'ensemble des actions possibles incluant l'affectation à un rejet d'ambiguïté γ_0 et l'action de rejet en distance γ_d, pour chaque vecteur d'entrée **x**, on obtient un ensemble $\Re(\mathbf{x})$ des résultats des actions obtenues :

$$\Re(\mathbf{x}) = \{k \in \{1,...M\} \mid \lambda_k(\mathbf{x}) > \theta_k\} \qquad [37]$$

Le résultat est exprimé de la manière suivante :

$$\gamma(\mathbf{x}) = \begin{cases} \gamma_k & si \quad \Re(\mathbf{x}) = \{k\} \\ \gamma_d & si \quad \Re(\mathbf{x}) = \varnothing \\ \gamma_0 & si \quad |\Re(\mathbf{x})| > 1 \end{cases} \qquad [38]$$

L'inconvénient de la règle précédente est le fait que les deux options de rejet sont contrôlées par le même paramètre θ_k. La règle dite *du rapport d'appartenance*, proposée par Frélicot (Frélicot, 1992) se base sur le rapport :

$$\upsilon = \frac{\lambda_m(\mathbf{x})}{\lambda_p(\mathbf{x})} \qquad [39]$$

avec

$$\begin{aligned} \lambda_p(\mathbf{x}) &= \max_{k \in \Re(\mathbf{x})} \lambda_k(\mathbf{x}) \\ \lambda_m(\mathbf{x}) &= \max_{k \in \Re(\mathbf{x}) \setminus \{p\}} \lambda_k(\mathbf{x}) \end{aligned} \qquad [40]$$

On a évidemment $0 \le \upsilon \le 1$, et son interprétation est la suivante :

- Si υ est proche de zéro, alors le degré d'appartenance λ_p est très grand par rapport aux autres degrés d'appartenance. L'action $\gamma_p(\mathbf{x})$ sera la plus favorable,
- Si υ est proche de un, alors au moins deux actions différentes ($\gamma_p(\mathbf{x})$ et $\gamma_m(\mathbf{x})$) ont des degrés d'appartenance presque identiques ($\lambda_p(\mathbf{x}) \approx \lambda_m(\mathbf{x})$). Cette situation correspond à un rejet d'ambiguïté.

Pour prendre la décision finale, ce rapport d'appartenance υ est alors comparé à un seuil *S*. Si $\upsilon > S$, alors **x** est rejeté en ambiguïté, sinon, l'action $\gamma_p(\mathbf{x})$ est sélectionnée.

c) Reconnaissance des formes par réseaux de neurones

Les réseaux de neurones sont des outils de l'intelligence artificielle, capables d'effectuer des opérations de classification. Leur fonctionnement est basé sur les principes de fonctionnement des neurones biologiques. Leur principal avantage par rapport aux autres outils est leur capacité d'apprentissage et de généralisation de leurs connaissances à des entrées inconnues. Les réseaux de neurones peuvent être également implémentés en circuits électroniques, offrant ainsi la possibilité d'un traitement temps réel. Le processus d'apprentissage est donc une phase très importante pour la réussite d'une telle opération. Plusieurs types de réseaux de neurones et plusieurs algorithmes d'apprentissage existent en littérature.

Une des qualités de ce type d'outil est son adéquation pour la mise au point de systèmes de surveillance modernes, capables de s'adapter à un système complexe avec reconfigurations multiples. Nous leur dédions l'ensemble du chapitre suivant. Nous présentons ainsi les deux architectures les plus utilisées en surveillance industrielle, à savoir le Perceptron Multi Couches (*PMC*) et les Réseaux à Fonctions de base Radiales (*RFR*). La Figure 7 montre l'architecture générale d'une application de surveillance par reconnaissance des formes avec réseaux de neurones. L'expert humain joue un rôle très important dans ce type d'application. Toute la phase d'apprentissage supervisé du réseau de neurones dépend de son analyse des modes de fonctionnement du système. Chaque mode est caractérisé par un ensemble de données recueillies sur le système. A chaque mode, on associe une expertise faite par l'expert.

Cette association (ensemble de données - modes de fonctionnement) sera apprise par le réseau de neurones. Après cette phase d'apprentissage, le réseau de neurones associera les classes représentant les modes de fonctionnement aux formes d'entrée caractérisées par les données du système.

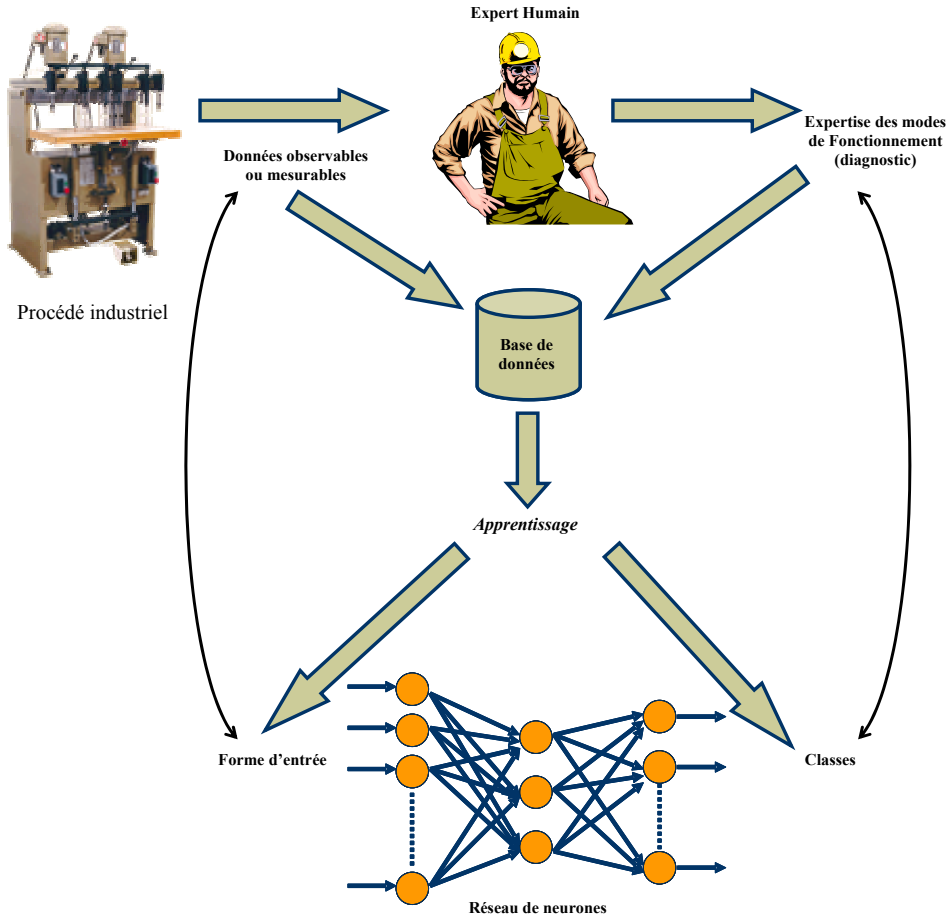


Figure 7. Reconnaissance des formes par réseau de neurones.

I.4. Conclusion

L'objectif de ce chapitre a été de donner un aperçu des techniques habituellement utilisées pour résoudre des problématiques de surveillance. La première partie de ce chapitre a été dédiée à la présentation des mots clés les plus importants en surveillance. La surveillance d'un équipement industriel se fait au travers de deux fonctions de base : la *détection* et le *diagnostic* des défaillances. La détection des défaillances a pour rôle de signaler toute situation autre qu'une situation nominale. En d'autres termes, tout ce qui n'est pas normal doit

être classé comme anormal. C'est alors que la fonction diagnostic doit *localiser* l'organe défaillant et *identifier* les causes ayant provoqué ces situations de défaillance. Cette opération est souvent menée par un expert et dans certains cas exige des connaissances poussées sur l'équipement.

Le classement des techniques de surveillance est fonction de l'existence ou non d'un modèle formel de l'équipement à surveiller. Nous avons donc présenté, d'une part, les méthodes qui ne se basent pas sur l'existence de ce modèle, c'est-à-dire les outils statistiques et les techniques de reconnaissance des formes et, d'autre part, celles qui l'utilisent, à savoir les méthodes d'estimation d'état et d'estimation paramétrique (techniques de *l'Automatique*). Ces dernières techniques ont pour principe de comparer l'état théorique du système fourni par le modèle avec son état courant donné par les observations. Souvent, pour des équipements complexes, ces modèles sont difficiles à mettre en œuvre et, quand ces modèles existent, leur réponse est souvent entachée d'incertitudes de modélisation. Ces incertitudes sont dues au fait qu'on ne peut pas cerner tous les paramètres physiques d'un équipement. Les incertitudes de modélisation ainsi que les bruits de mesures sont pris en compte d'une manière explicite par le modèle. Généralement, les paramètres inconnus, appelés perturbations, sont pris en compte dans le modèle d'une manière additive, les erreurs de modélisation d'une manière multiplicative et les bruits de mesure par la conception de résidus robustes généralement obtenus grâce à des techniques de découplage et des techniques de filtrage du signal. Les techniques de l'Intelligence Artificielle ne se basent pas sur le modèle de l'équipement et prennent en compte les perturbations ainsi que les bruits de mesure, d'une manière implicite.

La surveillance à base de modèle est souvent opérée hors ligne, empêchant ainsi des traitements temps réel. En revanche, l'Intelligence Artificielle offre des outils totalement découplés de la structure du système, permettant un suivi temps réel de l'évolution de celui-ci. Le raisonnement en ligne fait que l'approche de l'Intelligence Artificielle est plus robuste à des changements de modes opératoires comme pour les systèmes ayant plusieurs configurations. Elle est donc évolutive.

L'approche de surveillance par reconnaissance des formes, munie des notions de rejet (d'ambiguïté et de distance) et de la possibilité d'adaptation s'est donc montrée performante pour résoudre des problèmes de surveillance. En effet, la connaissance existante sur les différents modes de fonctionnement d'un système est toujours incomplète.

Le rejet d'ambiguïté permet de ne pas prendre une décision trop hâtive par rapport aux modes de fonctionnement identifiés. L'expert devra alors décider parmi deux ou plusieurs solutions proposées.

Le rejet de distance permet de tenir compte du caractère incomplet de la connaissance et l'adaptation périodiquement réalisée met en évidence de nouvelles classes. Il convient aussi d'interpréter physiquement ces classes en termes de causes des modes de fonctionnement. Cela ne peut être réalisé que par un expert.

Les systèmes de surveillance par outils de l'Intelligence Artificielle peuvent donc représenter d'excellents systèmes d'aide à la décision pour l'expert humain. Dans ce sens, le chapitre suivant sera consacré à la présentation des notions de base des réseaux de neurones, et de leur application en surveillance d'équipements de production.

Chapitre II

Application des réseaux de neurones à la surveillance des systèmes de production

Résumé : *Les avantages les plus importants que l'on peut attribuer à une application de surveillance par réseaux de neurones sont : la modélisation et estimation de fonctions non linéaires par apprentissage, la fusion de données et la généralisation et reconstruction des signaux capteurs. Deux architectures neuronales sont généralement utilisées pour des tâches de surveillance : le Perceptron Multi Couches et les Réseaux à base de Fonctions Radiales. Des différences majeures existent entres ces deux architectures qui ont une représentation* ***globale*** *pour le PMC et* ***locale*** *pour les RFR. La représentation locale est plus avantageuse pour la surveillance que la représentation globale. L'un des avantages est que contrairement au PMC, les RFR sont capables de dire « je ne sais pas ».*

Abstract : *The most important advantages of a neural network monitoring application are : the modeling and the estimation of non linear functions by learning, the data fusion and finally the generalization and reconstruction of sensors signals. Two neural networks are generally used in the monitoring application: Multi Layer Perceptron and Radial Basis Function network. Several differences exist between these two architectures which have a* ***global*** *representation for the MLP and a* ***local*** *one for the RBF. The local representation is more advantageous for the monitoring than the global one. One of the advantages is that contrary to the MLP, the RBF is able to say " I do not know ".*

Chapitre II

Application des réseaux de neurones à la surveillance des systèmes de production

II.1. Introduction

Dans la plupart des modélisations des systèmes industriels, des incertitudes persistent entre le comportement du système réel et l'évolution du modèle. Ces incertitudes sont dues, d'un côté, aux manques de connaissances exhaustives sur le fonctionnement de l'équipement et, d'un autre côté, le modèle ne prend en compte qu'une partie des paramètres qui influent sur l'évolution de la sortie. Par ailleurs, dans certains cas, ce modèle est quasiment impossible à obtenir.

Les réseaux de neurones peuvent fournir une solution intéressante pour des problématiques de surveillance d'équipements industriels. En effet, le chapitre précédent montre que leur utilisation ne nécessite pas l'existence d'une modélisation formelle de cet équipement. Par ailleurs, leurs capacités de mémorisation, d'apprentissage et d'adaptation représentent des fonctions très utiles à tout système de surveillance autonome.

Ce chapitre est structuré en trois parties. Une première partie est consacrée à la présentation des réseaux de neurones artificiels. Nous commençons par donner une brève présentation de l'évolution historique de cet axe de recherche dont la première inspiration biologique remonte à 1890. Avant de présenter le principe de fonctionnement des neurones artificiels, nous décrivons les bases essentielles des neurones biologiques. Nous concluons cette partie en présentant les propriétés les plus importantes des réseaux de neurones artificiels.

La deuxième partie de ce chapitre est consacrée aux architectures neuronales les plus utilisées en surveillance industrielle qui sont : le *Perceptron Multi Couches* et les *Réseaux à base de Fonctions Radiales* et, avec un degré moindre, le modèle de *Hopfield* et celui de *Kohonen.* Nous présentons ainsi leur principe de fonctionnement avec les différents algorithmes d'apprentissage.

Dans la troisième partie du chapitre, nous verrons comment les réseaux de neurones artificiels sont appliqués en surveillance. Deux types d'applications sont ainsi présentées : *la reconnaissance des formes* et *l'approximation de fonctions*.

Dans le premier type d'application, les réseaux de neurones servent à reconnaître le mode de fonctionnement ou de dysfonctionnement à partir des paramètres de surveillance[3]. Ces données représentent le vecteur forme qui caractérise chaque mode. L'identification d'un mode de défaillance à partir du vecteur forme est vue comme la détection d'une défaillance, puisque le système est sorti de la classe qui représente le mode nominal.

Dans le deuxième type d'application, les réseaux de neurones sont utilisés comme un approximateur universel. Ils offrent ainsi une identification de l'équipement (pour le pronostic) sous la forme d'une boîte noire en utilisant les techniques d'apprentissage.

II.2. Eléments de base des réseaux de neurones

II.2.1. Historique

L'origine de l'inspiration des réseaux de neurones artificiels remonte à *1890* où W. James, célèbre psychologue américain, introduit le concept de mémoire associative. Il propose ce qui deviendra une loi de fonctionnement pour l'apprentissage des réseaux de neurones, connue plus tard sous le nom de loi de Hebb. Quelques années plus tard, en *1949,* J. Mc Culloch et W. Pitts donnent leurs noms à une modélisation du neurone biologique (un neurone au comportement binaire). Ce sont les premiers à montrer que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes. C'est ensuite que D. Hebb, physiologiste américain, présente en *1949* les propriétés des neurones par le conditionnement chez l'animal. Ainsi, un conditionnement de type pavlovien tel que, nourrir tous les jours à la même heure un chien, entraîne chez cet animal la sécrétion de salive à cette heure précise même en l'absence de nourriture. La loi de modification des propriétés des connexions entre neurones qu'il propose, explique en partie ce type de résultats expérimentaux.

Les premiers succès de cette discipline remontent à *1957,* lorsque F. Rosenblatt développe le modèle du Perceptron. Il construit le premier neuro-ordinateur basé sur ce modèle et l'applique au domaine de la reconnaissance des formes. Notons qu'à cette époque les moyens à sa disposition étaient limités et c'était une prouesse technologique que de réussir à faire fonctionner correctement cette machine plus de quelques minutes. C'est alors qu'en *1960,* l'automaticien Widrow développe le modèle Adaline (Adaptative Linear Element). Dans sa structure, le modèle ressemble au Perceptron, cependant la loi d'apprentissage est différente. Celle-ci est à l'origine de l'algorithme de rétropropagation de gradient très utilisé aujourd'hui

[3] *Directement issus de l'équipement ou après traitement. Nous tenons à préciser que la phase de traitement et filtrage des données capteurs dépasse le cadre de notre étude.*

avec les Perceptrons Multi Couches. M. Minsky et S. Papert publient ensuite en *1969* un ouvrage qui met en évidence les limitations théoriques du Perceptron. Ces limitations concernent l'impossibilité de traiter des problèmes non linéaires en utilisant ce modèle.

Quelques années d'ombre se sont ensuite succédées de *1967* à *1982*. Le renouveau de cette discipline reprend en *1982* grâce à J. J. Hopfield, un physicien reconnu. Au travers d'un article court, clair et bien écrit, il présente une théorie du fonctionnement et des possibilités des réseaux de neurones. 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, J. J. Hopfield fixe préalablement le comportement à atteindre par 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. On peut citer encore la Machine de Boltzmann en *1983* qui était 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). C'est ensuite qu'en *1985* la rétro-propagation de gradient apparaît. C'est un algorithme d'apprentissage adapté au Perceptron Multi Couches. Sa découverte est réalisée par trois groupes de chercheurs indépendants. Dès cette découverte, nous avons la possibilité de réaliser une fonction non linéaire d'entrée/sortie sur un réseau, en décomposant cette fonction en une suite d'étapes linéairement séparables. Enfin, en *1989* Moody et Darken exploitent quelques résultats de l'interpolation multi variables pour proposer le Réseau à Fonctions de base Radiales (*RFR*), connu sous l'appellation anglophone *Radial Basis Function network* (*RBF*). Ce type de réseau se distingue des autres types de réseaux de neurones par sa représentation locale.

II.2.2. Le modèle neurophysiologique

L'élément de base du système nerveux central est le neurone. Le cerveau se compose d'environ mille milliards de neurones, avec 1000 à 10000 synapses (connexions) par neurone. Le neurone est une cellule composée d'un corps cellulaire et d'un noyau (Figure 8). Le corps cellulaire se ramifie pour former ce que l'on nomme les dendrites. Celles-ci sont parfois si nombreuses que l'on parle alors de chevelure dendritique ou d'arborisation dendritique. C'est par les dendrites que l'information est acheminée de l'extérieur vers le soma (corps du neurone). L'information est traitée alors par le corps cellulaire. Si le potentiel d'action dépasse un certain seuil, le corps cellulaire répond par un stimuli (Figure 9 –a–). Le signal transmis par le neurone chemine ensuite le long de l'axone (unique) pour être transmis aux autres neurones. La transmission entre deux neurones n'est pas directe. En fait, il existe un espace intercellulaire de quelques dizaines d'Angströms entre l'axone du neurone afférent et les dendrites du neurone efférent. La jonction entre deux neurones est appelée synapse (Figure 9 –b–).

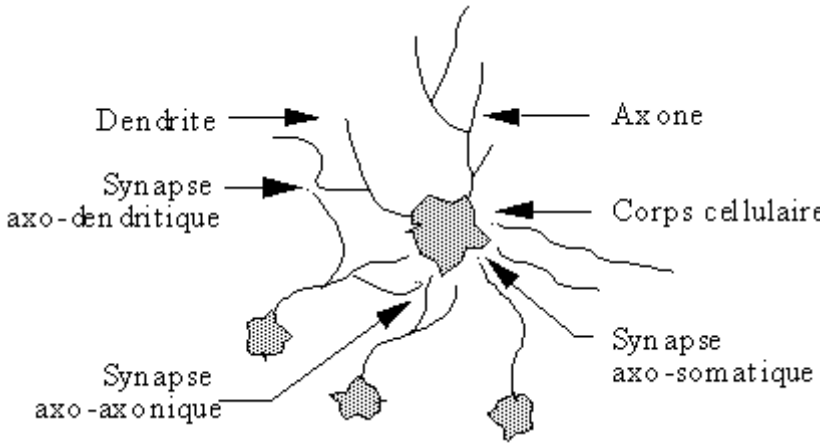


Figure 8. *Un neurone avec son arborisation dendritique*

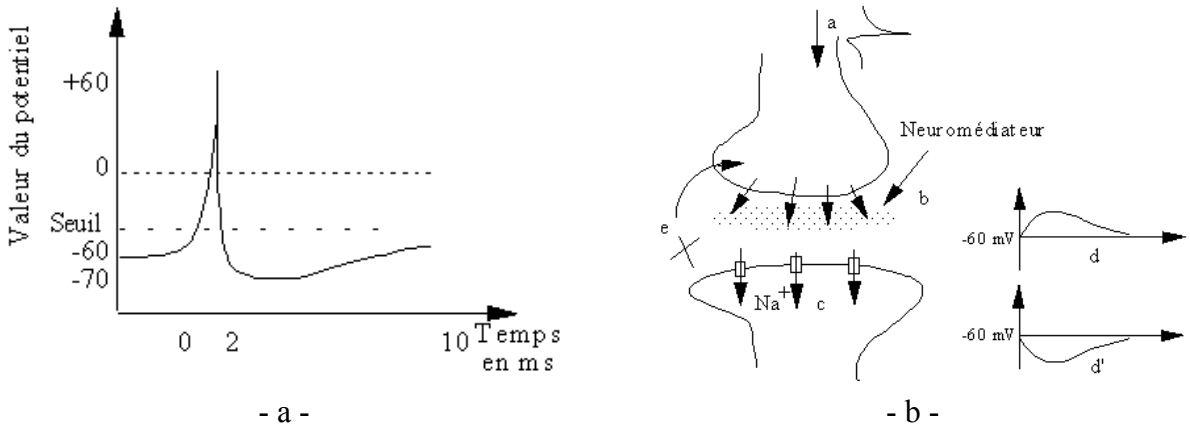


Figure 9. *–a– Un potentiel d'action. –b– Fonctionnement au niveau synaptique : a) Arrivée d'un potentiel d'action. b) Libération du neuromédiateur dans l'espace synaptique. c) Ouvertures des canaux ioniques dues au neuromédiateur. d) Génération d'un potentiel évoqué excitateur. d') Génération d'un potentiel évoqué inhibiteur. Les synapses inhibitrices empêchent la génération de potentiel d'action. e) Fermeture des canaux, élimination du neuromédiateur.*

II.2.3. Le modèle mathématique

Les réseaux de neurones biologiques réalisent facilement un certain nombre d'applications telles que la reconnaissance des formes, le traitement du signal, l'apprentissage par l'exemple, la mémorisation et la généralisation. C'est à partir de l'hypothèse que le comportement intelligent émerge de la structure et du comportement des éléments de base du cerveau, que les réseaux de neurones artificiels se sont développés. La Figure 10 montre la structure d'un

neurone artificiel. Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones amont. A chacune de ces entrées est associée un poids *w,* abréviation de *weight* (poids en anglais), représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones aval. A chaque connexion est associée un poids.

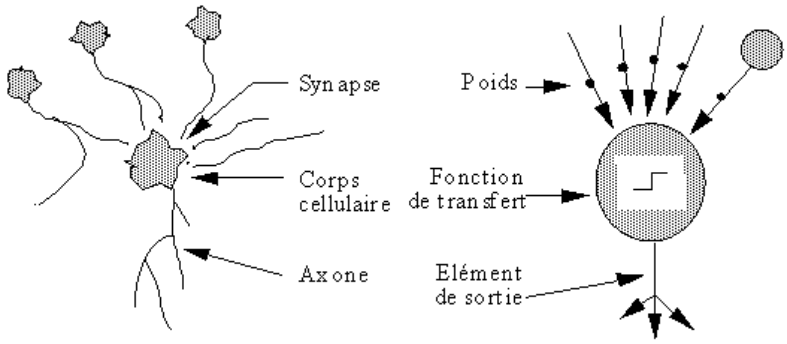


Figure 10. *Mise en correspondance du neurone biologique /et du neurone artificiel*

Par analogie avec le neurone biologique, le comportement du neurone artificiel se compose de deux phases (Figure 11).

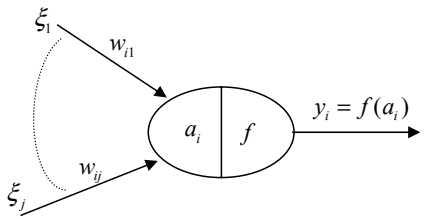


Figure 11. *Neurone formel.*

La première phase, appelée *activation*, représente le calcul de la somme pondérée des entrées selon l'expression suivante :

$$a_i = \sum_j w_{ij} \xi_j \qquad [41]$$

A partir de cette valeur, une fonction de transfert calcule la valeur de l'état du neurone. C'est cette valeur qui sera transmise aux neurones aval. Il existe de nombreuses formes possibles pour la fonction de transfert. Les plus courantes sont présentées sur la Figure 12. On remarquera qu'à la différence des neurones biologiques dont l'état est binaire, la plupart des

fonctions de transfert sont continues, offrant une infinité de valeurs possibles comprises dans l'intervalle [0, +1] ou [-1, +1].

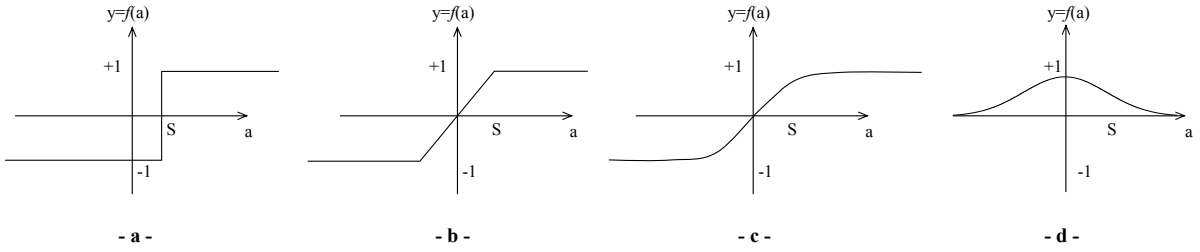


Figure 12. *Différents types de fonctions de transfert pour le neurone artificiel. a) fonction à seuil du neurone de Mc Culloch et W. Pitts (1949), b) linéaire par morceaux du modèle Adaline de Widrow et Hoff (1960), c) sigmoïde d'un réseau Perceptron Multi Couches de Rosenblatt (1962), d) gaussienne du réseau RFR de Moody et Darken (1989)*[4].

II.2.4. Propriétés des Réseaux de neurones artificiels

II.2.4.1. Apprentissage et mémoire

L'une des caractéristiques les plus complexes du fonctionnement de notre cerveau est bien la phase d'apprentissage. C'est une phase au bout de laquelle certaines modifications s'opèrent entre les connexions des neurones : certaines sont renforcées et d'autres affaiblies ou carrément inhibitrices. Le cerveau converge alors vers un comportement souhaité : par exemple l'apprentissage d'une langue, ou encore l'apprentissage par un enfant à reconnaître son environnement. Ceci nous emmène à la notion de mémoire qui donne au cerveau la capacité de retrouver des expériences passées. Le cerveau possède plusieurs types de mémoires. Nous ne nous attarderons pas sur ces différents types de mémoires mais tout ce que nous pouvons retenir c'est que le cerveau humain procède par association. Cela permet par exemple de retrouver une information à partir d'éléments incomplets ou imprécis (bruités). Par exemple, le fait de voir un bout d'une photographie qu'on connaît déjà est suffisant pour que notre cerveau soit capable de la reconnaître. Dans le paragraphe suivant, nous détaillerons d'avantage cette importante caractéristique des réseaux de neurones artificiels, plus connue comme *capacité de généralisation*. Le mécanisme de l'association permet aussi au cerveau de converger vers un état à partir d'un autre état. Par exemple, le fait de passer devant une boulangerie nous fait rappeler qu'on devait acheter du pain. Cette deuxième importante caractéristique est aussi connue sous le nom de mémoire adressée par le contenu, dont le

[4] Pour le cas de la fonction gaussienne, l'activation *a* n'est pas la même que celle représentée par l'expression [41], mais représente un calcul de distance euclidienne. Nous détaillerons plus loin cette fonction.

modèle de Hopfield s'en inspire. Par analogie avec les réseaux de neurones biologiques, les réseaux de neurones artificiels tentent de reproduire les caractéristiques les plus importantes du comportement biologique, à savoir *l'apprentissage*, *la généralisation* et *l'association*.

L'apprentissage des réseaux de neurones artificiels est une phase qui permet de déterminer ou de modifier les paramètres du réseau, afin d'adopter un comportement désiré. Plusieurs algorithmes d'apprentissage ont été développés depuis la première règle d'apprentissage de Hebb en 1949. Nous présentons au paragraphe 3 une partie de ces algorithmes d'apprentissage qui sont classés en deux catégories : *supervisé* et *non supervisé*.

Dans l'apprentissage *supervisé*, un superviseur (ou expert humain) fournit une valeur ou un vecteur ζ de sortie (appelé cible ou sortie désirée) que le réseau de neurones doit associer au vecteur d'entrée **x**. L'apprentissage consiste dans ce cas à modifier les paramètres du réseau de neurones afin de minimiser l'erreur entre la sortie cible et la sortie réelle du réseau de neurones.

Dans l'apprentissage *non supervisé*, les données ne contiennent pas d'informations sur une sortie désirée. Il n'y a pas de superviseur. Il s'agit de déterminer les paramètres du réseau de neurones suivant un critère à définir.

II.2.4.2. Sous-apprentissage, généralisation et sur-apprentissage

La capacité de généralisation est l'une des raisons qui motivent l'étude et le développement des réseaux de neurones artificiels. Elle peut être définie par la capacité d'élargir les connaissances acquises après apprentissage à des données nouvellement rencontrées par le réseau de neurones. C'est de cette façon que les réseaux de neurones sont capables d'approximer une fonction uniquement à partir d'une partie des données, ou encore d'associer un vecteur d'entrée **x** qui n'a pas fait l'objet d'un apprentissage, à une classe ***c***. On peut distinguer deux types de généralisation : *locale* et *globale*.

Dans l'approche *locale*, chaque neurone est associé à une région d'activation (région d'influence) localisée dans l'espace des données. Seule une partie des neurones participe donc à la réponse du réseau. Deux types d'architectures neuronales possèdent cette particularité : les Réseaux à base de Fonctions Radiales et la carte de Kohonen. Par contre, dans l'approche *globale,* l'ensemble des neurones du réseau participe à l'élaboration de la sortie du réseau. L'information est donc distribuée dans le réseau tout entier. C'est le cas des réseaux de neurones de type Perceptron Multi Couches ou le modèle de Hopfield.

L'approche globale est supposée plus robuste aux *pannes* éventuelles de quelques neurones isolés. Par ailleurs, lors d'un apprentissage incrémental, des problèmes d'*interférences catastrophiques* peuvent apparaître : la modification des paramètres d'un neurone a des répercussions sur l'ensemble de la fonction modélisée par le réseau. Des régions de l'espace des données éloignées de la région de la donnée à mémoriser risquent d'être perturbées. En

d'autres termes, tandis que le réseau apprend dans une région de l'espace des données, le modèle peut *oublier* ce qu'il a appris dans d'autres régions.

La figure ci-dessous illustre la différence de généralisation entre les deux architectures neuronales caractérisées par les deux fonctions d'activation (locale pour le RFR et globale pour le PMC) :

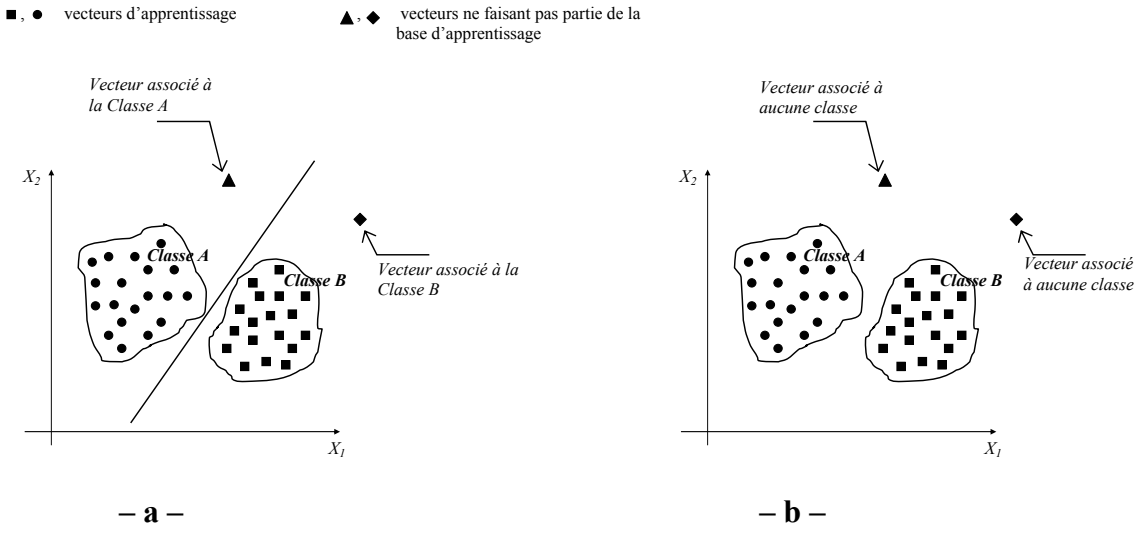


Figure 13. *Différence des capacités de généralisation entre le PMC (a) et les RFR (b).*

Cette capacité de généralisation est très liée à la notion de sur-apprentissage[5]. Ces deux caractéristiques sont complètement antagonistes. On parle de sur-apprentissage quand le réseau a trop parfaitement appris les exemples proposés. Il sera donc incapable de généraliser. Ceci est appelé calcul de la *complexité* du réseau de neurones. En pratique, on effectue un apprentissage sur un sous ensemble *S* de l'espace de données *D*. Le réseau est alors testé sur un ensemble de test *T* ne faisant pas partie de l'apprentissage. On calcule alors la moyenne des erreurs quadratiques sur l'ensemble *S* appelée '*erreur base apprentissage*' et sur l'ensemble de test *T* appelée '*erreur base test*'. Plus on agrandit l'ensemble *S*, plus l'*erreur base apprentissage* diminue, plus l'*erreur base test* augmente. Le réseau perd dans ce cas là ses capacités de généralisation. La Figure 14 illustre clairement ce compromis entre sur-apprentissage, sous-apprentissage et bonne généralisation.

[5] *Over-fitting* en anglais.

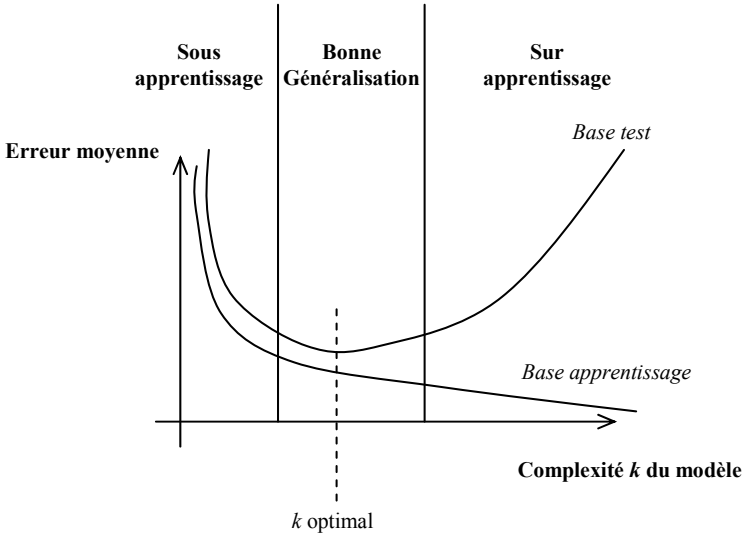


Figure 14. *Complexité d'un réseau de neurones : pour trouver un compromis entre erreur apprentissage et erreur test, on compare pour différentes complexités k du modèle, l'erreur moyenne sur la base d'apprentissage et sur une base de test indépendante. La complexité optimale du modèle correspond à la valeur de k pour laquelle l'erreur sur la base de test est minimale.*

II.3. Architectures neuronales les plus utilisées en surveillance

Les techniques de surveillance par réseaux de neurones sont fondées sur l'existence d'une base de données d'apprentissage et non sur l'existence d'un modèle formel ou fonctionnel de l'équipement. Le principe d'une telle application est de trouver une relation entre une (des) variable(s) d'entrée et une (des) variable(s) de sortie. Les variables d'entrée peuvent être quantifiables (sorties capteurs) ou qualifiables (observations faites par l'opérateur). A partir de ces variables d'entrée, le réseau de neurones donne une réponse caractérisée par deux types de variables de sortie. Des variables de sortie réelles qui peuvent représenter une sortie estimée d'un paramètre de surveillance ou des variables de sortie catégorielles qui représentent l'état de fonctionnement de l'équipement. Selon la nature de ces données en sortie, il existe deux types d'applications. Le premier type est une application *d'approximation de fonctions*, qui consiste à estimer une sortie mesurée de l'équipement. Dans ce cas, les réseaux de neurones sont utilisés en tant qu'approximateur universel et fournissent un modèle sous la forme d'une boîte noire du système. Ceci n'est autre que de l'identification des processus industriels. Comme pour les méthodes à base de modèle, décrites au chapitre précédent, la comparaison de la sortie du réseau de neurones avec celle du système réel donne un résidu qui servira à déterminer si le système est dans un état défaillant ou pas. Le deuxième type d'application considère le problème de la surveillance comme un

problème de reconnaissance des formes[6]. La forme à reconnaître est caractérisée par l'ensemble des données (quantifiables et – ou – qualifiables) et les classes d'appartenance représentant les différents modes (de fonctionnement ou de dysfonctionnement qui ont été définis au chapitre précédent). Le réseau de neurones doit nous fournir une réponse qui nous renseigne sur l'état de fonctionnement de l'équipement. Il assure la fonction de *détection* (fonctionnement normal ou pas) et la fonction de *diagnostic* (reconnaître un mode de défaillance). La figure ci-dessous illustre les deux types d'applications des réseaux de neurones en surveillance :

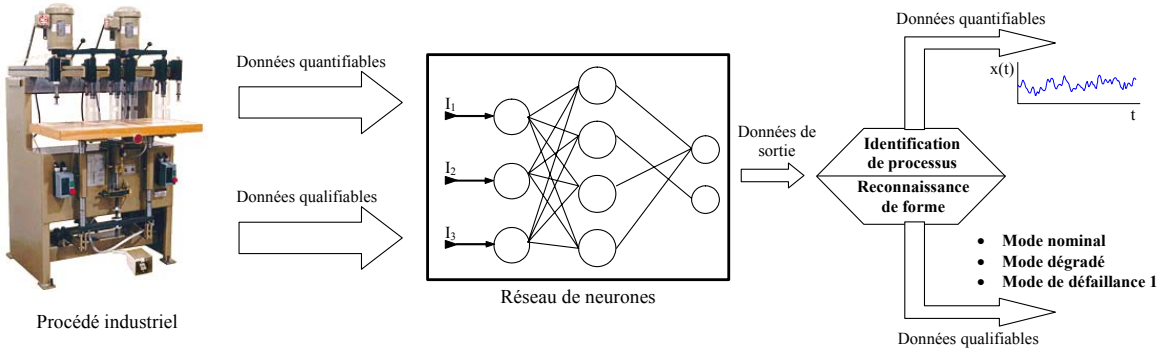


Figure 15. *Application des réseaux de neurones en surveillance d'équipements industriels.*

Koivo (Koivo, 1994) a publié un article de synthèse sur l'application des réseaux de neurones en surveillance industrielle. Cet article présente les architectures neuronales les plus utilisées dans ce domaine avec des résultats pratiques dans des applications de diagnostic en statique et en dynamique. Trois types de réseaux de neurones ont été testés : le Perceptron Multi Couche, le Réseau à Fonctions de base Radiales et la carte de Kohonen. Les deux premiers réseaux donnent d'assez bons résultats en classification et en identification de processus. Les RFR peuvent se montrer plus performants que les PMC à condition de déterminer judicieusement leurs paramètres. Par contre, le réseau de Kohonen n'est pas aussi performant que les deux premiers mais ses capacités d'auto-adaptation (apprentissage non supervisé) sont très appréciées. L'auteur insiste dans sa conclusion sur la pertinence du choix du type de réseaux de neurones en fonction de l'application. Les types de réseaux de neurones les plus utilisés en surveillance s'avèrent donc être :

- Le Réseau à Fonctions de base Radiales (*RFR*),
- Le Perceptron Multi Couches (*PMC*),

[6] On rencontre souvent en littérature les termes *classification* ou *discrimination* associés au domaine de la reconnaissance des formes.

et avec un degré moindre les mémoires auto associatives (modèle de Hopflield) et la carte de *Kohonen*. Chaque type de réseau de neurones a ses avantages et ses inconvénients selon l'application que l'on fait. Au paragraphe suivant nous allons présenter le principe de fonctionnement de chacune des quatre architectures.

II.3.1. Le Perceptron Multi Couches (PMC)[7]

II.3.1.1. Le Perceptron simple

Comme nous l'avons énoncé au début de ce chapitre, le premier modèle de réseau de neurones a été introduit par McCulloch et Pitts en 1949 (McCulloch *et al.,* 1949). Les neurones de ce modèle sont binaires. Ils reçoivent des signaux excitateurs et inhibiteurs provenant des neurones en amont afin d'effectuer une sommation. Si cette somme est supérieure à un seuil, le neurone est dans un état actif et émet un signal vers les autres neurones. Si ce n'est pas le cas, il est dans un état inactif et n'émet aucun signal.

Après la description de ce premier modèle de neurones, il fallait disposer d'un moyen de réaliser l'apprentissage. S'inspirant du conditionnement de type pavlovien chez l'animal, Donald O. Hebb a introduit en 1949 la première règle d'apprentissage qui permet de modifier les poids synaptiques. Le principe de cette règle – connue sous la règle de Hebb – est le suivant (Hebb, 1949) :

« *Lorsqu'une connexion entre deux cellules est très forte, si la cellule émettrice s'active alors la cellule réceptrice s'active aussi. Pour lui permettre de jouer ce rôle déterminant lors du mécanisme d'apprentissage, il faut donc augmenter le poids de cette connexion. En revanche, si la cellule émettrice s'active sans que la cellule réceptrice le fasse, ou si la cellule réceptrice s'active alors que la cellule émettrice ne s'était pas activée, cela traduit le fait que la connexion entre ces deux cellules n'est pas prépondérante dans le comportement de la cellule réceptrice. On peut donc, dans la phase d'apprentissage, laisser un poids faible à cette connexion. »*

En d'autres termes, si deux neurones connectés entre eux sont activés en même temps, alors on renforce la connexion qui les relie. Dans le cas contraire, elle n'est pas modifiée. Cette règle d'apprentissage a été le point de départ des travaux de Rosenblatt (Rosenblatt, 1958) qui a développé la première version d'un modèle neuronal très connu de nos jours, à savoir le Perceptron. C'est un réseau à deux couches (une couche d'entrée et une couche de sortie) de type *feedforward* (propagation avant). Les neurones de la couche d'entrée ont pour rôle de fournir au réseau les données externes. Chaque neurone de la couche de sortie effectue une somme pondérée de ses entrées :

[7] En anglais, ce type de réseau est appelé : Multi Layer Perceptron (MLP)

$$O_i = f(a_i) = f(\sum_{k=1}^{k=N} w_{ik}\xi_k) \qquad [42]$$

où w_{ik} est le poids de la connexion qui relie l'unité k à l'unité i, a_i est l'activation de l'unité i, f est la fonction d'activation des unités (Figure 16). Cette fonction d'activation est du type fonction à seuil avec l'expression suivante :

$$f(x) = \begin{cases} +1 & \text{si } x \geq \theta \\ -1 & \text{si } x < \theta \end{cases} \qquad [43]$$

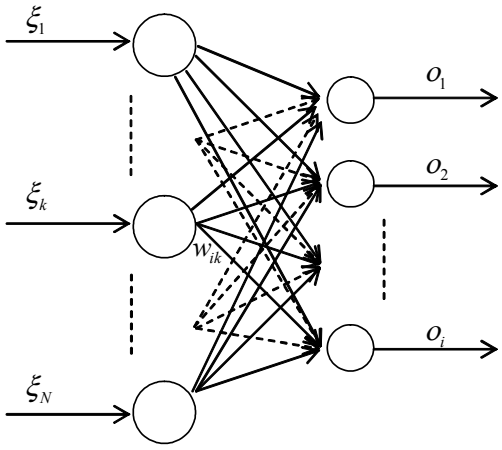


Figure 16. *Perceptron Simple, modèle de Rosenblatt.*

Le rôle de l'apprentissage est de modifier les poids des connexions entre les neurones d'entrée et ceux de sortie, de manière à obtenir une réponse que l'on souhaite reproduire par le réseau de neurones. Rosenblatt s'est inspiré de la règle de Hebb pour la modification des poids. Son principe est de rajouter, dans le cas où la sortie obtenue $o_i^{\wp}$ du réseau est différente de la sortie désirée $\zeta_i^{\wp}$, une quantité Δw_{ik} aux poids de chaque connexion. Dans le cas contraire, les connexions demeurent inchangées. On peut exprimer ce principe par :

$$w_{ik}^{nouveau} = w_{ik}^{ancien} + \Delta w_{ik} \qquad [44]$$

où Δw_{ik} est la quantité ajoutée au poids w_{ik}. Pour chaque exemple $\wp$ de l'ensemble des exemples d'apprentissage, on peut ainsi écrire :

$$\Delta w_{ik} = \eta(\zeta_i^{\wp} - o_i^{\wp})\xi_k^{\wp} \qquad [45]$$

Le paramètre η est appelé *taux d'apprentissage*. Il détermine la dynamique suivant laquelle les modifications vont avoir lieu.

Cette procédure d'apprentissage pour le Perceptron simple peut converger vers un état des poids des connexions donnant de bons résultats, à la seule condition que le problème soit linéairement séparable. Malheureusement, un grand nombre de problèmes rencontrés en pratique, ne sont pas linéairement séparables. Le Perceptron possède tout de même une bonne capacité de généralisation.

II.3.1.2. Règle de Delta

Contrairement au modèle de Rosenblatt où les neurones ont des fonctions d'activation à seuils, Widrow et Hoff en 1960 (Widrow *et al.,* 1960) ont proposé un modèle de Perceptron avec neurones linéaires (fonction d'activation linéaire) :

$$f(a_i) = a_i \qquad [46]$$

L'avantage d'utiliser des unités linéaires est qu'elles permettent de calculer une fonction de coût qui évalue l'erreur que commet le réseau. Cette erreur peut être définie en fonction des erreurs entre réponses désirées et réponses obtenues par le réseau. Cette erreur est donc fonction des poids du réseau :

$$E(w) = \frac{1}{2}\sum_{i\wp}(\zeta_i^{\wp} - o_i^{\wp})^2 = \frac{1}{2}\sum_{i\wp}(\zeta_i^{\wp} - \sum_k w_{ik}\xi_k^{\wp})^2 \qquad [47]$$

L'objectif de l'apprentissage est de modifier les valeurs des poids du réseau de façon à minimiser cette erreur. Il s'agit donc de descendre le long de la surface définie par l'erreur dans l'espace des poids du réseau. L'algorithme de descente du gradient suggère de changer chaque poids w_{ik} d'une quantité Δw_{ik} proportionnelle au gradient de l'erreur :

$$\Delta w_{ik} = -\eta\frac{\partial E}{\partial w_{ik}} = \eta\sum_{\wp}(\zeta_i^{\wp} - o_i^{\wp})\xi_k^{\wp} \qquad [48]$$

On peut également effectuer une modification des poids à chaque exemple d'entrée $\wp$. On obtient alors :

$$\Delta w_{ik} = \eta(\zeta_i^{\wp} - o_i^{\wp})\xi_k^{\wp} = \eta\vartheta_i^{\wp}\xi_i^{\wp} \qquad [49]$$

avec $\vartheta_i^{\wp}$ est définie par

$$\vartheta_i^{\wp} = \zeta_i^{\wp} - o_i^{\wp} \qquad [50]$$

Cette règle d'apprentissage est généralement appelée la règle *Delta* ou règle de *Widrow-Hoff*. Elle converge vers la solution des moindres carrés qui minimise la fonction d'erreur *E*.

II.3.1.3. La rétropropagation

Un des désavantages du Perceptron est qu'il minimise une erreur en tout ou rien à cause de sa fonction d'activation (expression [43]). Il ne prend donc pas en compte la notion de distance. De ce fait, il est très peu robuste. La règle d'apprentissage de *Widrow-Hoff* (règle de Delta) ne travaille plus en tout ou rien mais minimise une fonction d'erreur quadratique, donc plus robuste. Malheureusement, cette règle ne peut s'appliquer que sur des réseaux à une seule couche de poids adaptatifs. C'est donc en étendant la règle de *Widrow-Hoff* que plusieurs équipes de chercheurs (Le Cun, 1985) et (Werbos, 1974) ont développé un algorithme d'apprentissage appelé *rétropropagation du gradient de l'erreur*, généralisé ensuite par l'équipe de Rummelhart en 1986 (Rummelhart *et al.,* 1986). Cet algorithme fournit une façon de modifier les poids des connexions de toutes les couches d'un Perceptron Multi Couches (*PMC*).

Soit le réseau à deux couches décrit par la Figure 17 dans lequel les unités de sortie sont notées o_i, les unités cachées v_j et les unités d'entrée ξ_k. Les connexions des unités d'entrée aux unités cachées sont notées w_{jk} et celles des unités cachées aux unités de sortie par w_{ij}. L'entrée *k* a pour valeur $\xi_k^{\wp}$ lorsque la donnée $\wp$ est présentée au réseau. Ces valeurs peuvent être binaires (0/1 ou +1/-1) ou continues.

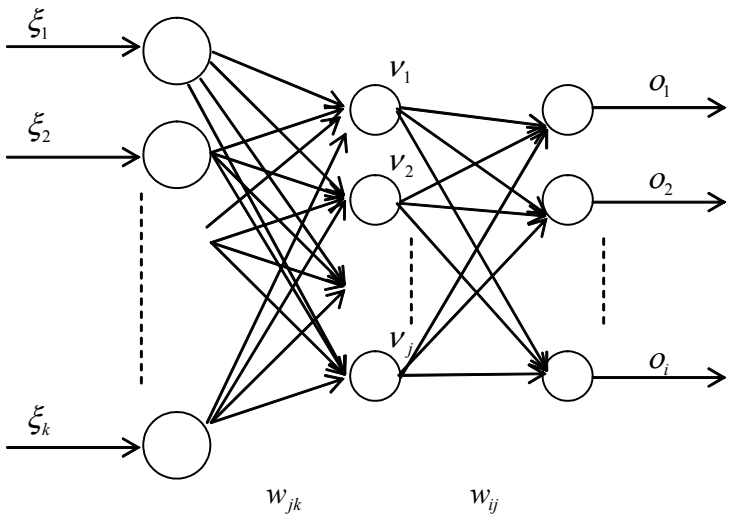


Figure 17. *Perceptron Multi Couches*

Pour la donnée d'entrée $\wp$, la valeur de sortie de l'unité cachée *j* est donnée par :

$$v_j^{\wp} = f(a_j^{\wp}) = f(\sum_k w_{jk}\xi_k^{\wp}) \qquad [51]$$

Les unités de sortie ont comme valeur :

$$o_i^{\wp} = f(a_i^{\wp}) = f(\sum_j w_{ij}v_j^{\wp}) \qquad [52]$$

Les fonctions d'erreurs partielles et globales sont alors définies par

$$E^{\wp} = \frac{1}{2}\sum_i (o_i^{\wp} - \zeta_i^{\wp})^2 \quad \text{et} \quad E = \sum_{\wp} E^{\wp} \qquad [53]$$

La minimisation de la fonction d'erreur globale va se faire par une descente de gradient. Par conséquent, après la présentation de tous les vecteurs d'entrée de la base d'apprentissage, nous modifierons la valeur de chaque connexion par la règle

$$\Delta w = -\eta \frac{\partial E}{\partial w} = -\eta \sum_{\wp} \frac{\partial E^{\wp}}{\partial w} \qquad [54]$$

Cette règle d'apprentissage est généralement appelée la règle de *delta généralisée*. Dans l'expression [53], seule la sortie o_i dépend du paramètre w. Selon la position des poids des connexions, deux cas se présentent :

- *Cas des connexions entre la couche cachée et celle de sortie (w_{ij}) :*

Pour le cas des neurones de sortie, l'expression [54] devient fonction du paramètre w_{ij} qui influe uniquement sur la sortie du neurone d'indice *i*. Nous pouvons donc décomposer la dérivée de l'expression [54] par :

$$\frac{\partial E^{\wp}}{\partial w_{ij}} = \frac{\partial E^{\wp}}{\partial o_i^{\wp}} \frac{\partial o_i^{\wp}}{\partial a_i^{\wp}} \frac{\partial a_i^{\wp}}{\partial w_{ij}} = (o_i^{\wp} - \zeta_i^{\wp}) f_i'(a_i^{\wp}) v_j^{\wp} \qquad [55]$$

L'expression [54] devient alors

$$\Delta w_{ij} = \eta \sum_{\wp} f_i'(a_i^{\wp}).(\zeta_i^{\wp} - o_i^{\wp}).v_j^{\wp} \qquad [56]$$

- *Cas des connexions entre la couche d'entrée et la couche cachée (w_{jk}) :*

Pour le cas des neurones cachés, l'expression [54] est fonction du paramètre w_{jk} qui influe non seulement sur la sortie du neurone *j* de la deuxième couche, mais aussi sur tous les neurones *i* de la couche de sortie (en aval) qui lui sont connectés. On obtient alors l'équation suivante :

$$\frac{\partial E^{\wp}}{\partial w_{jk}} = \frac{\partial E^{\wp}}{\partial v_j^{\wp}} \frac{\partial v_j^{\wp}}{\partial a_j^{\wp}} \frac{\partial a_j^{\wp}}{\partial w_{jk}} = \frac{\partial E^{\wp}}{\partial v_j^{\wp}} f_j^{'}(a_j^{\wp}) \xi_k^{\wp} \qquad [57]$$

Le premier terme de cette expression devient alors :

$$\frac{\partial E^{\wp}}{\partial v_j^{\wp}} = \sum_i \frac{\partial E^{\wp}}{\partial o_i^{\wp}} \frac{\partial o_i^{\wp}}{\partial v_j^{\wp}} = \sum_i \frac{\partial E^{\wp}}{\partial o_i^{\wp}} \frac{\partial o_i^{\wp}}{\partial a_i^{\wp}} \frac{\partial a_i^{\wp}}{\partial v_j^{\wp}} = \sum_i (o_i^{\wp} - \zeta_i^{\wp}) f_i^{'}(a_i^{\wp}) w_{ij} \qquad [58]$$

On obtient alors la modification des poids :

$$\Delta w_{jk} = \eta \sum_{\wp} \left(f_j^{'}(a_j^{\wp}) \xi_k^{\wp} \left(\sum_i (\zeta_i^{\wp} - o_i^{\wp}) f_i^{'}(a_i^{\wp}) w_{ij} \right) \right) \qquad [59]$$

Après avoir calculé la variation des poids des connexions pour tous les neurones de sortie (expression [56]), on calcule alors la variation des poids des connexions de la couche cachée (expression [59]). On met ainsi à jour les poids des connexions de la couche de sortie jusqu'à la couche d'entrée : on *rétropropage* ainsi le signal d'erreur. C'est de là que vient le nom de cet algorithme : *rétropropagation du gradient de l'erreur*. Du fait de sommer les Δw_{ij} pour tous les vecteurs $\wp$ de la base d'apprentissage puis de remettre à jour les poids avec la variation totale ainsi calculée, l'algorithme est appelé gradient total. Une autre façon de faire, appelée version séquentielle, modifie les poids des connexions après chaque présentation d'un vecteur d'entrée $\wp$. Une version stochastique permet de prendre en compte les vecteurs d'apprentissage $\wp$ d'une façon aléatoire.

L'algorithme de rétropropagation du gradient de l'erreur a permis de dépasser les limites du Perceptron simple. Il s'avère capable de résoudre un grand nombre de problèmes de classification et de reconnaissance des formes et a donné lieu à beaucoup d'applications. Cet algorithme souffre néanmoins de nombreux défauts, parmi lesquels :

- Une des limitations importantes est le temps de calcul : l'apprentissage est très long ;
- Une grande sensibilité aux conditions initiales, c'est-à-dire à la manière dont sont initialisés les poids des connexions ;
- De nombreux problèmes sont dus à la géométrie de la fonction d'erreur : minimums locaux. Ce problème est en partie résolu avec le gradient stochastique, mais il subsiste quand même ;
- Le problème de dimensionnement du réseau. La rétropropagation apprend une base d'apprentissage sur un réseau dont la structure est fixée a priori. La structure est définie par le nombre de couches cachées, le nombre de neurones par couches et la topologie des connexions. Un mauvais choix de structure peut dégrader considérablement les performances du réseau.

II.3.2. Le modèle de Hopfield

Le modèle de Hopfield est basé sur le concept de mémoire adressée par le contenu : *Mémoire Associative*. A partir d'un réseau entièrement connecté, un apprentissage basé sur la règle de Hebb est proposé. Chaque information mémorisée représente un point stable de l'espace d'état vers lequel l'évolution du système aboutit à partir d'un point initial voisin correspondant à une version déformée de l'information mémorisée. Autrement dit, l'espace d'état du système comporte des attracteurs qui correspondent aux informations mémorisées.

L'architecture du réseau est telle que chaque neurone est connecté à tous les autres sauf à lui même. L'architecture du réseau de Hopfield est symétrique ; c'est-à-dire le poids w_{ij} de la connexion entre le neurone *i* et le neurone *j* est identique à w_{ji}, poids de la connexion entre les neurones *j* et *i*.

L'évolution du réseau peut être conduite suivant plusieurs stratégies. Une évolution synchrone conduit à calculer l'état de tous les neurones du réseau à chaque unité de temps. Une autre possibilité consiste à mettre à jour un seul neurone à la fois de façon asynchrone. Ainsi, soit un neurone est choisi de façon aléatoire à chaque pas de temps, soit chaque neurone met à jour son état indépendamment des autres selon une probabilité par unité de temps déterminée.

Les connexions entre les neurones d'un réseau à *n* cellules peuvent être représentées par une matrice $n \times n$:

$$\mathbf{w} = \begin{bmatrix} 0 & w_{12} & \cdots & w_{1n} \\ w_{21} & 0 & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & 0 \end{bmatrix} \qquad [60]$$

où w_{ij} représente le poids de la connexion reliant le neurone *j* au neurone *i*. L'évolution de l'état o_i de chaque cellule *i* est donnée par la relation suivante :

$$o_i = \mathrm{sgn}(\sum_j w_{ij} o_j - \theta_i) \qquad [61]$$

θ_i le seuil du neurone *i* et *sgn* est la fonction *signe* définie par :

$$\mathrm{sgn}(x) = \begin{cases} +1 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases} \qquad [62]$$

Tout le problème consiste donc à déterminer, s'il existe, l'ensemble des poids w_{ij} qui permettent au réseau de se comporter comme une mémoire adressée par le contenu. La condition irrévocable de convergence du réseau de Hopfield vers un état stable est que la

matrice des poids des connexions **w** ([60]) soit symétrique et de diagonale égale à zéro (Hopfield, 1982). La particularité du réseau de Hopfield est que son évolution vers un état stable est caractérisée par une fonction $\Psi(\mathbf{o})$ appelée *fonction d'énergie*. La valeur de cette fonction d'énergie dépend du vecteur d'états $\mathbf{o}$ des neurones et de la matrice **w** :

$$\Psi(\mathbf{o}) = -\frac{1}{2}\mathbf{o}\mathbf{w}\mathbf{o}^t + \theta\mathbf{o}^t \qquad [63]$$

En développant cette expression, on obtient :

$$\Psi(\mathbf{o}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}o_i o_j + \sum_{i=1}^{n}\theta_i o_i \qquad [64]$$

Le réseau de Hopfield converge vers un des états stables (états attracteurs) correspondant à un minimum local de la fonction d'énergie $\Psi(\mathbf{o})$ (Rojas, 1996). Pour définir les états stables à mémoriser par le réseau, il faut donc trouver la matrice des poids **w** qui minimise la fonction d'énergie.

II.3.3. Le réseau de Kohonen

Il a été observé que, dans de nombreuses zones du cortex cérébral, des colonnes voisines ont tendance à réagir à des entrées similaires. Dans les aires visuelles, par exemple, deux colonnes proches sont en correspondance avec deux cellules proches de la rétine (Hubel *et al.,* 1977). Des observations identiques ont pu être faites dans le bulbe olfactif ou dans l'appareil auditif (Knudsen *et al.,* 1979). Ces observations ont mené Kohonen (Kohonen, 1989) à proposer un modèle de *carte topologique auto-adaptative* qui permet de coder des motifs présentés en entrée, tout en conservant la topologie de l'espace d'entrée.

Dans la plupart des applications, les neurones d'une carte de Kohonen sont disposés sur une grille 2D (Figure 18). Chaque neurone *i* de la carte effectue un calcul de la distance euclidienne entre le vecteur d'entrée ξ et le vecteur poids $\mathbf{w}_i$.

Dans les réseaux de Kohonen, la mise à jour des paramètres des neurones s'effectue sur tout un voisinage d'un neurone *i.* Un rayon de voisinage *r* représente donc la longueur du voisinage d'un neurone *i* en terme de nombre de neurones. On définit alors une fonction $\hbar(i,k)$ égale à 1 pour tous les neurones *k* voisins du neurone *i* compris dans le rayon *r* et égale à zéro pour tous les autres neurones. L'algorithme d'apprentissage de la carte de Kohonen se présente comme suit :

- Initialiser aléatoirement les vecteurs $\mathbf{w}_i$. On donne une valeur initiale au rayon r et au taux d'apprentissage η.

- Calculer la distance euclidienne entre le vecteur présenté ξ et le vecteur de poids de chaque neurone,
- Choisir le neurone *k* ayant la distance la plus petite,
- Les vecteurs de pondération de tous les neurones *i* de la carte de Kohonen sont alors mis à jour selon l'équation :

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta\hbar(i,k)(\xi - \mathbf{w}_i) \qquad [65]$$

- Réduire la taille du voisinage et reprendre l'apprentissage du vecteur des pondérations.

Après un long temps de convergence, le réseau évolue de manière à représenter au mieux la topologie de l'espace de départ. Il faut noter que la notion de conservation de la topologie est en fait abusive puisqu'en général, la taille du vecteur d'entrée est bien supérieure à la dimension de la carte (souvent égale à 2) et il est donc impossible de conserver parfaitement la topologie.

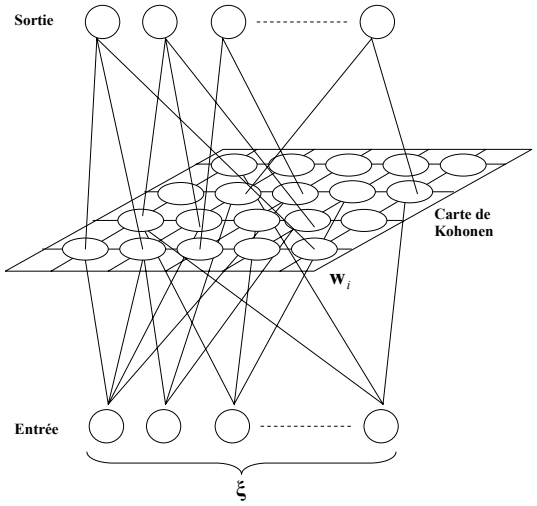


Figure 18. *Carte topologique auto-adaptative de Kohonen.*

II.3.4. Réseau de neurones à Fonctions de base Radiales (RFR)

II.3.4.1. Fondements théoriques

Les réseaux de neurones à fonctions de base radiales sont des réseaux de type *feedforward*[8] avec une seule couche cachée (Figure 19). L'utilisation de ces réseaux remonte aux années

[8] *feedforward* est l'appellation anglophone des réseaux à propagation avant

soixante-dix par (Hardy, 1971), (Agteberg, 1974), (Schagen, 1979) pour résoudre des problèmes d'interpolation multi variables. Les bases théoriques de ces réseaux ont été ensuite approfondies par (Powell, 1987), (Poggio *et al.,* 1989) et (Moody *et al.,* 1989). D'autres travaux se sont succédés où l'application des *RFR* a été élargie à d'autres domaines, à savoir la prédiction de l'évolution des systèmes dynamiques (Broomhead *et al.,* 1988), (Casdagli, 1989) et la classification de phonèmes (Renals et al., 1989). La particularité de ces réseaux réside dans le fait qu'ils sont capables de fournir une représentation locale de l'espace grâce à des fonctions de base radiales $\phi(\|.\|)$ dont l'influence est restreinte à une certaine zone de cet espace ($\|.\|$ représente la norme euclidienne).

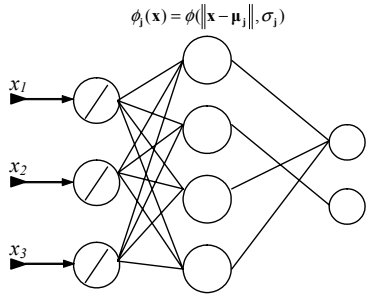


Figure 19. *Réseau à fonctions de base radiales*.

Deux paramètres sont associés à cette fonction de base : un vecteur de référence $\boldsymbol{\mu}_j$ appelé centre ou *prototype* et la dimension σ_i du champ d'influence appelé *rayon d'influence*. La réponse de la fonction de base dépend donc de la distance du vecteur d'entrée $\mathbf{x}$ au vecteur prototype $\boldsymbol{\mu}_j$, et de la taille du champ d'influence :

$$\phi_j(\mathbf{x}) = \phi_j(\|\mathbf{x} - \boldsymbol{\mu}_j\|, \sigma_j) \qquad [66]$$

où la fonction $\phi_j(\|.\|)$ est généralement maximale lorsque $\mathbf{x} = \boldsymbol{\mu}_j$ et décroît d'une façon monotone vers 0 quand $r = \|\mathbf{x} - \boldsymbol{\mu}_j\| \to \infty$. Pour qu'une fonction $r \mapsto \phi_j(r)$ soit utilisée comme fonction de base et résoudre le problème de l'interpolation, il faudrait qu'elle soit *définie positive* (Micchelli, 1986). Cette propriété se présente comme suit :

Définition 1

Une fonction g de $]0,+\infty[$ *dans* $\mathbb{R}$ *est dite* ***définie positive*** *si et seulement si la fonction* $x \mapsto g(\sqrt{x})$ *est absolument décroissante.*

Définition 2

Une fonction h de $]0,+\infty[$ *dans* $\mathbb{R}$ *est dite* ***absolument décroissante*** *si :*

- *Elle est indéfiniment dérivable,*
- *Pour tout* $x \in]0,+\infty[$ *et tout entier naturel n*

$$h^{(n)}(x) > 0 \quad \text{si n est pair,}$$
$$h^{(n)}(x) < 0 \quad \text{si n est impaire.}$$

Avec $h^{(n)}(x)$ *qui représente la* $n^{ème}$ *dérivée de h.*

Ainsi, nous pouvons produire les couples de fonctions absolument décroissantes et de fonctions définies positives suivantes :

Absolument décroissante	Définie positive
$r \mapsto e^{-r/2\sigma^2}$	$r \mapsto e^{-r^2/2\sigma^2}$
$r \mapsto \frac{1}{(\sigma^2 + r)^{\alpha}}$	$r \mapsto \frac{1}{(\sigma^2 + r^2)^{\alpha}}$

La fonction de base la plus utilisée est la gaussienne (Figure 20). Elle s'exprime, sous sa forme la plus générale, par :

$$\phi_j(\mathbf{x}) = \exp(-\frac{1}{2\sigma_j^2}(\mathbf{x}-\boldsymbol{\mu}_j)(\mathbf{x}-\boldsymbol{\mu}_j)^t) \qquad [67]$$

où σ_j^2 désigne la variance associée à la cellule. La décroissance de la gaussienne est la même pour toutes les directions de l'espace. Les courbes d'iso activation des cellules cachées sont alors des hyper sphères. Un nombre restreint de fonctions de base participent au calcul de la sortie pour une entrée donnée.

Les *RFR* peuvent être classés en deux catégories, en fonction du type du neurone de sortie (Mak *et al.*, 2000), (Moody *et al.*, 1989), (Xu, 1998) :

- *Normalisé* :

$$y(\mathbf{x}) = \frac{\sum_j w_j \phi_j(\mathbf{x})}{\sum_j \phi_j(\mathbf{x})} \qquad [68]$$

- *Non-Normalisé* :

$$y(\mathbf{x}) = \sum_j w_j \phi_j(\mathbf{x}) \qquad [69]$$

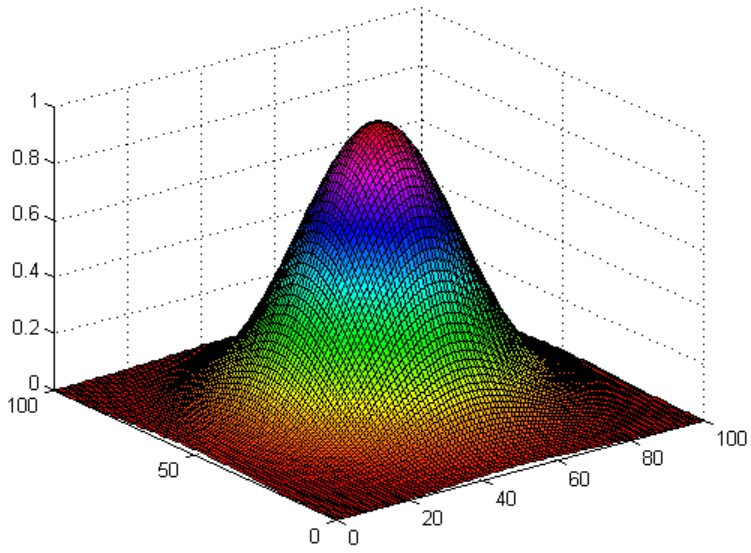


Figure 20. *Représentation d'une gaussienne en 3 dimensions.*

La raison essentielle du choix de la gaussienne comme fonction de base des *RFR* est que cette fonction est factorisable. En effet, on peut facilement démontrer que parmi toutes les fonctions de base respectant la condition de Micchelli (Micchelli, 1986), la gaussienne est la seule fonction qui peut être décomposée en produit de fonctions gaussiennes unidimensionnelles :

$$\phi(\mathbf{x}) = \mathrm{e}^{-\frac{\|\mathbf{x}-\boldsymbol{\mu}\|^2}{2\sigma^2}} = \prod_i \mathrm{e}^{-\frac{(x_i-\mu_i)^2}{2\sigma^2}} \qquad [70]$$

avec $\mathbf{x} = [x_i]$ et $\boldsymbol{\mu} = [\mu_i]$. Cette particularité devient intéressante pour l'adéquation biologique des réseaux de neurones artificiels. On a du mal à imaginer comment un neurone est capable de calculer la fonction $\phi(\mathbf{x})$ pour des problèmes de grande dimension. Par contre, le schéma de la Figure 21 est biologiquement plausible (Poggio *et al.,* 1989). La fonction $\phi(\mathbf{x})$ a été factorisée en gaussiennes unidimensionnelles. Celles-ci représentent les fonctions d'activation des neurones.

Un deuxième avantage de la factorisation de la fonction de base en produit de gaussiennes unidimensionnelles est la possibilité d'avoir une matrice de variance complète. La décroissance des gaussiennes pour chaque dimension n'est donc pas forcément la même. Les cellules cachées sont alors des hyper ellipses (Figure 22). Ce type particulier de réseaux RFR s'appelle Réseaux de neurones à Fonctions de base Radiales Généralisés (*RFRG*)[9] (Broomhead *et al.,* 1988), (Moody *et al.,* 1989), (Mak *et al.,* 2000), (Zemouri *et al.,* 2002 -a-).

[9] Generalized Radial Basis Functions (GRBF)

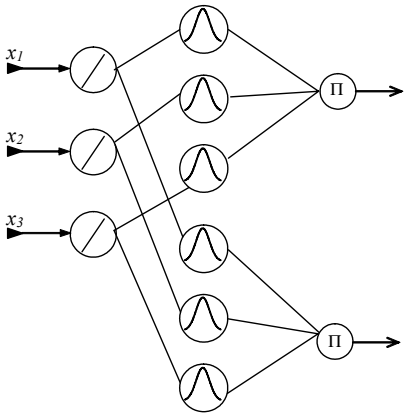
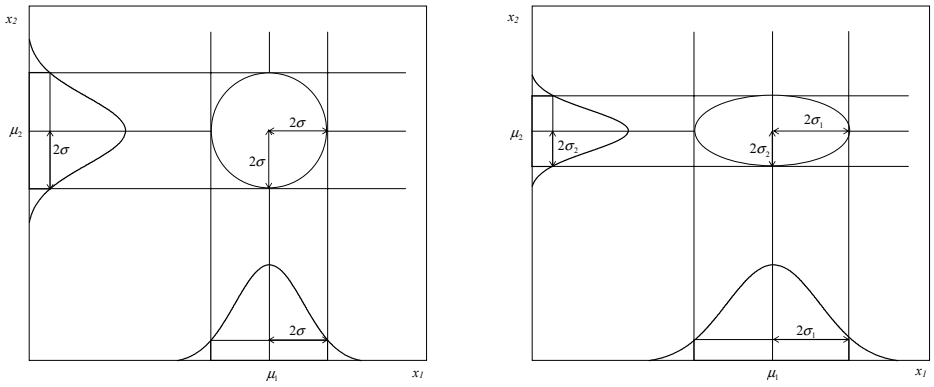
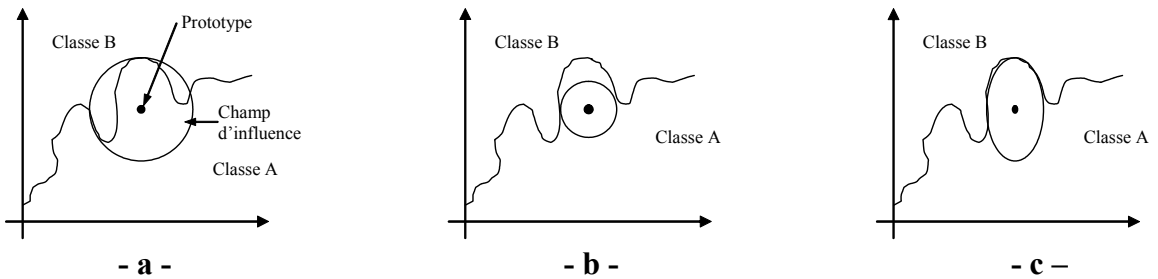


Figure 21. *Factorisation d'un Réseau à fonctions de base radiales.*



- a - **- b -**

Figure 22. *Intérêt de la factorisation de la fonction gaussienne. a) matrice de variance-covariance diagonale (RFR sphérique), b) matrice de variance-covariance complète (RFR Elliptique ou Généralisé).*



- a - **- b -** **- c –**

Figure 23. *Avantage de la zone de couverture elliptique : a) sur-couverture du prototype, b) sous-couverture du prototype, c) couverture idéale*.

a) Problème d'interpolation et approximation de fonctions

L'interpolation est définie comme suit : soit un ensemble de *N* vecteurs d'entrée $\mathbf{x}_n$ de dimension *d* et un ensemble à une dimension ζ_n (n=1,…..,N). Le problème est de trouver une fonction continue *h(***x***)* tel que : $h(\mathbf{x}_n) = \zeta_n$.

La solution à ce problème en utilisant les *RFR* consiste à choisir un groupe de *N* fonctions de base, centrées aux *N* points d'entrée ($\boldsymbol{\mu}_i = \mathbf{x}_i$) et en utilisant la définition des fonctions radiales avec w_n le poids de la connexion de la $n^{éme}$ fonction de base vers le neurone de sortie (Poggio, 1989), (Ghosh *et al.* 1992) :

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \phi(\|\mathbf{x} - \boldsymbol{\mu}_n\|) \qquad [71]$$

Tout le problème de l'interpolation consiste donc à résoudre les *N* équations linéaires pour trouver les coefficients inconnus qui sont les poids w_n :

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \vdots \\ \zeta_N \end{bmatrix} \qquad [72]$$

L'écriture matricielle devient alors :

$$\mathbf{\Phi}.\mathbf{w} = \boldsymbol{\zeta} \qquad [73]$$

La condition de Micchelli (Micchelli, 1986) sur les fonctions de base gaussienne permet d'avoir la matrice $\mathbf{\Phi} = [\phi_{ij} = \phi(\|\mu_i - \mu_j\|, \sigma_j)]$ inversible. Par conséquent, le vecteur de pondération peut s'écrire sous la forme suivante avec $\mathbf{w} = [w_n]_{n=1,\ldots,N}$:

$$\mathbf{w} = \mathbf{\Phi}^{-1}.\boldsymbol{\zeta} \qquad [74]$$

En pratique, le problème d'interpolation n'est pas toujours intéressant. Les données sont généralement bruitées et l'interpolation utilisant toutes les données de l'apprentissage provoque un sur-apprentissage et par conséquent, une faible généralisation. Si, en revanche, on considère un nombre de fonctions de base inférieur à celui des données d'apprentissage, les paramètres de ces fonctions de base peuvent être ajustés par apprentissage afin de minimiser l'erreur quadratique. Ainsi, au lieu de combiner des fonctions de base fixes, on combine des fonctions dont la forme elle-même est ajustée par des paramètres. On comprend que ces degrés de liberté supplémentaires permettent de réaliser une fonction donnée avec un plus petit nombre de fonctions élémentaires. On obtient alors un modèle non linéaire par

rapport à ses paramètres. Nous verrons plus loin les différents algorithmes d'apprentissage qui peuvent être utilisés pour déterminer les paramètres des fonctions de base.

b) Classification

Les *RFR* sont également utilisés dans des problèmes de classification. En théorie de la classification probabiliste, la loi du vecteur **x** quand on ne connaît pas sa classe d'appartenance est donnée par la loi mélange $\varphi(\mathbf{x})$:

$$\varphi(\mathbf{x}) = \sum_{k=0}^{M} \Pr(\alpha_k)\varphi(\mathbf{x}/\alpha_k) \qquad [75]$$

$\varphi(\mathbf{x}/\alpha_k)$, supposée connue, représente la loi conditionnelle d'appartenance du vecteur **x** à la classe α_k et $\Pr(\alpha_k)$ représente la probabilité a priori des classes α_k, supposée connue. Cette expression ressemble à l'expression de sortie d'un réseau *RFR* (Ghosh *et al.*, 2000):

$$f(\mathbf{x}) = \sum_{k=0}^{M} w_{ik}\phi_k(\mathbf{x}) \qquad [76]$$

avec w_{ik} représentant le poids de la connexion entre le $k^{éme}$ neurone radial et le $i^{éme}$ neurone de sortie. D'après cette représentation, les centres des gaussiennes peuvent être considérés comme étant des vecteurs représentatifs. La sortie des unités cachées représente la probabilité a posteriori d'appartenance du vecteur d'entrée **x** à la classe (α_k). Les poids des connexions représentent la probabilité *a priori* des classes. La sortie de tout le réseau représente la probabilité a posteriori de la classe de sortie C_i.

$$\phi_k(\mathbf{x}) = \frac{f(\mathbf{x}/\alpha_k)\Pr(\alpha_k)}{\sum_{k=1}^{M} f(\mathbf{x}/\alpha_k)\Pr(\alpha_k)} = \Pr(\alpha_k/\mathbf{x}) \qquad [77]$$

et

$$w_{ik} = \frac{\Pr(\alpha_k/C_i)\Pr(C_i)}{\Pr(\alpha_k)} = \Pr(C_i/\alpha_k) \qquad [78]$$

II.3.4.2. Techniques d'apprentissage

L'apprentissage des *RFR* permet de déterminer les paramètres de ces réseaux qui sont :

- les centres des fonctions radiales (*prototypes*) $\boldsymbol{\mu}_j$,

- la variance σ_j^2 (*carré du rayon d'influence*),
- les poids des connexions entre les neurones de la couche intermédiaire et ceux de la couche de sortie $\mathbf{w}_k$.

On peut classer ces techniques en trois groupes :

- techniques supervisées,
- techniques heuristiques,
- techniques d'apprentissage en deux temps.

a) Techniques supervisées

Le principe de ces techniques est de minimiser l'erreur quadratique :

$$E = \sum_n E_n \qquad [79]$$

avec :

$$E_n = \frac{1}{2}\sum_k (\zeta_k^n - y_k(\mathbf{x}^n))^2 \qquad [80]$$

où ζ_k^n représente la sortie désirée du $k^{éme}$ neurone de sortie pour l'exemple *n* et $y_k(\mathbf{x}^n)$ représente sa sortie réelle par rapport à l'entrée **x** de l'exemple *n*.

En utilisant la fonction gaussienne et en considérant les variations suivantes : Δw_{ij} pour les poids des connexions entre la couche intermédiaire et celle de sortie, $\Delta\mu_j$ pour les centres des prototype et $\Delta\sigma_j$ pour les rayons d'influence des fonctions gaussiennes. A chaque pas d'apprentissage, la loi de mise à jour est obtenue en utilisant la descente de gradient sur E_n (Le Cun, 1985), (Rumelhart *et al.*, 1986), (Ghosh *et al.*, 1992) :

$$\Delta w_{kj} = \eta_1(\zeta_k^n - y_k(\mathbf{x}^n))\phi_j(\mathbf{x}^n) \qquad [81]$$

$$\Delta\mu_j = \eta_2\phi_j(\mathbf{x}^n)\frac{\left\|\mathbf{x}^n - \boldsymbol{\mu}_j\right\|}{\sigma_j^2}\left(\sum_k(\zeta_k^n - y_k(\mathbf{x}^n))w_{kj}\right) \qquad [82]$$

$$\Delta\sigma_j = \eta_3\phi_j(\mathbf{x}^n)\frac{\left\|\mathbf{x}^n - \boldsymbol{\mu}_j\right\|^2}{\sigma_j^3}\left(\sum_k(\zeta_k^n - y_k(\mathbf{x}^n))w_{kj}\right) \qquad [83]$$

avec η_1, η_2, η_3 représentant les paramètres d'apprentissage.

b) Techniques heuristiques

Le principe de ces techniques est de déterminer les paramètres du réseau d'une manière itérative. Généralement, on commence par initialiser le réseau sur un centre avec un rayon d'influence initial ($\boldsymbol{\mu}_0, \sigma_0$). Les centres des prototypes $\boldsymbol{\mu}_i$ sont créés au fur et à mesure de la présentation des vecteurs d'apprentissage. L'étape suivante a pour but de modifier les rayons d'influence et les poids des connexions (σ_i, w_i) (uniquement les poids entre la couche intermédiaire – neurones gaussiens – et la couche de sortie). Nous présentons par la suite quelques heuristiques utilisées dans ce sens :

Algorithme RCE (Restricted Coulomb Energy) (Hudak, 1992)

L'algorithme *RCE* a été inspiré de la théorie des systèmes de charges de particules. Le principe de l'algorithme est de modifier l'architecture du réseau d'une façon itérative durant l'apprentissage. Les neurones intermédiaires ne sont ajoutés que lorsque cela est nécessaire. Un seuil θ permet d'ajuster les rayons d'influence (Figure 24). Le pseudo code suivant présente une itération d'apprentissage d'un vecteur $\mathbf{x}$ de classe *c*:

//Mise à zéro des poids :
pour tout prototype *k* de classe *i* $\boldsymbol{\mu}_k^i$ **faire**
    $w_k^i = 0.0$
fin
//Itération d'apprentissage
pour tout vecteur d'apprentissage $\mathbf{x}$ de classe *c* **faire**:
    si $\exists \boldsymbol{\mu}_k^c : \phi_k^c(\mathbf{x}) \geq \theta$ **alors**
        $w_k^c += 1.0$
    sinon
        //création d'un nouveau prototype
        ajouter un nouveau prototype $\boldsymbol{\mu}_{m_c+1}^c$ avec:
        $\boldsymbol{\mu}_{m_c+1}^c = \mathbf{x}$
        $\sigma_{m_c+1}^c = \max_{i \neq c \wedge 1 \leq j \leq m_i} \left\{ \sigma : \phi_{m_c+1}^c(\boldsymbol{\mu}_j^i) < \theta \right\}$
        $w_{m_c+1}^c = 1.0$
        $m_c += 1$
    fin
    //ajuster les zones de conflits
    pour tout $i \neq c, 1 \leq j \leq m_i$ **faire**
        $\sigma_j^i = \max \left\{ \sigma : \phi_j^i(\mathbf{x}) < \theta \right\}$
    fin
fin

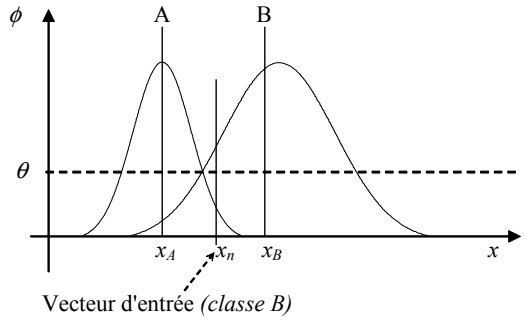


Figure 24 *Ajustement des rayons d'influence avec un seul seuil θ (algorithme RCE). Pas d'ajout de prototype pour le nouveau vecteur d'entrée ($\phi^B(x_n) > \theta$). Le seuil θ permet de réduire les zones de conflits par les relations suivantes : $\phi^B(x_A) < \theta$, $\phi^A(x_n) < \theta$, $\phi^A(x_B) < \theta$.*

Algorithme « DDA :Dynamic Decay Adjustment » (Berthold et al., 1995)

Cette technique, extraite partiellement de l'algorithme *RCE*, est utilisée pour des applications en classification (discrimination). Le principe de la technique est d'introduire deux seuils θ^- et θ^+ afin de réduire les zones de conflit entre prototypes (problème essentiel rencontré dans l'algorithme *RCE*). Pour assurer la convergence de l'algorithme d'apprentissage, le réseau doit satisfaire les deux inégalités [84] ci-après pour chaque vecteur $\mathbf{x}$ de classe *c* de la base d'apprentissage (Figure 25).

$$\begin{array}{c} \exists i : \phi_i^c(\mathbf{x}) \geq \theta^+ \\ \forall k \neq c, \forall j : \phi_j^k(\mathbf{x}) < \theta^- \end{array} \qquad [84]$$

Les auteurs ont testé cette technique simple d'apprentissage sur plusieurs bases de données, en comparant les performances du *DDA* avec d'autres techniques d'apprentissage ainsi qu'avec les performances du *PMC* (Perceptron Multi Couche). D'une part, les résultats du *DDA* semblent nettement meilleurs que les autres, surtout en terme de nombre d'itérations avant que l'apprentissage converge. A titre d'exemple, pour une application sur le problème des deux spirales[10] *(Berthold et al., 1995)*, le réseau *RFR* « *boosté* » par la technique *DDA* a convergé au bout de 4 périodes (ici une période représente un cycle de présentation de tous les vecteurs de la base d'apprentissage), alors que le *PMC* avec l'algorithme de rétro-propagation, a convergé au bout de 40000 périodes. D'autre part, tous les vecteurs faisant partie de la base d'apprentissage ont été correctement classés avec le *RFR*, résultat qui n'est pas forcement obtenu avec le *PMC*. Cet exemple montre clairement la différence en classification entre la représentation locale des *RFR* et globale du *PMC*.

[10] Problème type de classification (Lang *et al.,* 1988)

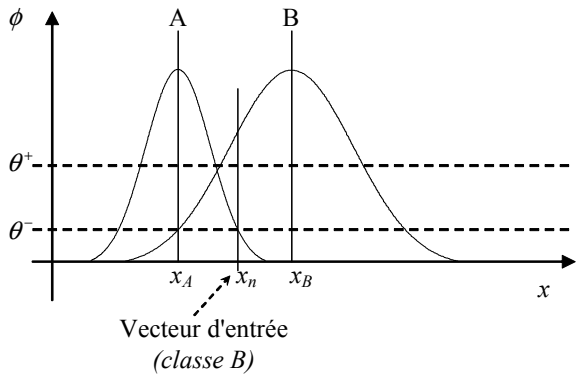


Figure 25. *Ajustement des rayons d'influence avec deux seuils* θ^- et θ^+ *(algorithme DDA). Pas d'ajout de prototype pour le nouveau vecteur d'entrée (*$\phi^B(x_n) > \theta^+$*). Le seuil* θ^- *permet de réduire les zones de conflits par les relations suivantes :* $\phi^B(x_A) < \theta^-$, $\phi^A(x_n) < \theta^-$, $\phi^A(x_B) < \theta^-$.

Le pseudo code suivant présente une itération d'apprentissage d'un vecteur **x** de classe *c*:

//Mise à zéro des poids :
pour tout prototype *i* de classe *k* $\boldsymbol{\mu}_i^k$ **faire**
    $w_i^k = 0.0$
fin
//Itération d'apprentissage
pour tout vecteur d'apprentissage **x** de classe *c* **faire**:
    si $\exists \boldsymbol{\mu}_i^c : \phi_i^c(\mathbf{x}) \geq \theta^+$ **alors**
        $w_i^c += 1.0$
    sinon
        //création d'un nouveau prototype
        ajouter un nouveau prototype $\boldsymbol{\mu}_{m_c+1}^c$ avec:
        $\boldsymbol{\mu}_{m_c+1}^c = \mathbf{x}$
        $\sigma_{m_c+1}^c = \max_{k \neq c \wedge 1 \leq j \leq m_k} \left\{ \sigma : \phi_{m_c+1}^c(\boldsymbol{\mu}_j^k) < \theta^- \right\}$
        $w_{m_c+1}^c = 1.0$
        $m_c += 1$
    fin
    //ajuster les zones de conflits
    pour tout $k \neq c, 1 \leq j \leq m_k$ **faire**
        $\sigma_j^k = \max \left\{ \sigma : \phi_j^k(\mathbf{x}) < \theta^- \right\}$
    fin
fin

c) Techniques d'apprentissage en deux temps

Ces techniques permettent d'estimer les paramètres du *RFR* en deux phases : une première phase sert à déterminer les centres et les rayons des fonctions de base. Dans cette étape, on utilise uniquement les vecteurs d'entrée. L'apprentissage est considéré comme étant non supervisé. La deuxième phase a pour but de calculer les poids des connexions de la couche cachée vers la couche de sortie (apprentissage supervisé).

Première phase (Non-Supervisée)

- *Segmentation en k-moyennes des centres*

Les centres des prototypes et la matrice des variances d'un *RFR* peuvent être calculés en deux phases. Dans la première phase, l'algorithme de segmentation en *k*-moyennes est appliqué pour déterminer le centre du nuage de N^j points faisant partie de la même classe. Cette technique représente la méthode de quantification vectorielle la plus connue (MacQueen, 1967) et permet de partitionner itérativement les données en minimisant la variance intragroupe. Chaque itération de l'algorithme des *k-moyennes* comporte deux étapes :

– *Une étape d'affectation* : dans cette étape on procède à l'affectation de l'ensemble des points d'apprentissage appartenant à la classe *c* aux *k* centres (de la même classe) en minimisant la variance intragroupe. Le minimum de cette variance s'obtient en affectant chaque point au centre le plus proche au sens de la distance euclidienne. On obtient ainsi une segmentation de l'espace d'apprentissage χ, en *k* groupes disjoints $\{\chi_j\}_{j=1}^{k}$ de N^j points.

– *Une étape de minimisation* : la seconde étape fait décroître à nouveau la variance intragroupe en calculant le nouveau centre de chaque groupe par la moyenne $\hat{\boldsymbol{\mu}}_j$:

$$\boldsymbol{\mu}_j \approx \hat{\boldsymbol{\mu}}_j = \frac{1}{N^j} \sum_{\mathbf{x} \in \chi_j} \mathbf{x} \qquad [85]$$

La convergence de l'algorithme est prononcée à la stabilité des centres des nuages de points. On calcule alors la matrice de covariance de la fonction gaussienne. Celle-ci est estimée par l'expression de la covariance simple ci-dessous :

$$\sigma^2{}_j \approx \hat{\sigma}_j^2 = \frac{1}{N^j} \sum_{\mathbf{x} \in \chi_j} (\mathbf{x} - \hat{\boldsymbol{\mu}}_j)(\mathbf{x} - \hat{\boldsymbol{\mu}}_j)^t \qquad [86]$$

Malgré sa facilité de mise en œuvre, cet algorithme présente néanmoins quelques inconvénients (Zemouri *et al.,* 2002 –b–), (Dreyfus et al., 2002) :

- Il n'existe aucune méthode formelle pour déterminer le nombre adéquat de centres ou prototypes,
- Cette méthode dépend fortement des valeurs initiales. Généralement, les centres sont initialisés aléatoirement. Plusieurs itérations avec des initialisations différentes sont nécessaires pour atteindre le meilleur résultat,
- On peut également avoir des situations où un nuage de points appartenant à un prototype soit vide.

D'autres dérivées de cet algorithme peuvent aussi être utilisées mais sans résoudre les points précédents : *l'algorithme de Segmentation en k-moyennes flou (Fuzzy k-Means*) qui a été présenté au chapitre précédent et *l'algorithme de Segmentation séquentielle en k-moyennes* (ou *Sequential k-Means Clustering)*. Nous reviendrons plus en détail sur les points négatifs de cet algorithme au chapitre V dans lequel nous proposons une version améliorée de l'algorithme des *k-moyennes*. Cette version présente des performances meilleures que la version classique.

- *Méthode EM (Expectation Maximisation)* (Dempster *et al.*, 1977)

De par l'analogie existante entre les *RFR* et les modèles de mélange de gaussiennes, une autre technique appelée algorithme *EM* (Expectation Maximisation) est utilisée pour déterminer, de manière itérative, les paramètres d'un mélange de gaussiennes (par le maximum de vraisemblance). L'algorithme permet ainsi d'obtenir les paramètres du réseau en deux étapes : l'étape *E* qui calcule la valeur de l'espérance mathématique des données inconnues par rapport aux données connues et l'étape *M* qui maximise le vecteur des paramètres de l'étape *E*. Un des problèmes de l'algorithme EM concerne la convergence qui n'est pas assurée à chaque fois (Hernandez, 1999).

Deuxième phase (Supervisée)

- *Maximum d'appartenance*

Cette technique, utilisée dans les applications de classification, prend les valeurs les plus importantes des fonctions de base $\phi_i(\mathbf{x})$:

$$\phi_{\max} = \max_{i=1}^{N} \phi_i \qquad [87]$$

où N est le nombre de fonctions de base pour toutes les classes. La sortie du réseau de neurones est alors donnée par :

$$y = classe(\phi_{\max}) \qquad [88]$$

- *Algorithme des moindres carrés*

Supposons que soit fixée une fonction de risque empirique à minimiser (R_{emp}). Comme pour les *PMC*, la détermination des paramètres peut alors être conduite de façon supervisée par une méthode de descente de gradient. Si la fonction de coût choisie est quadratique à fonctions de base fixées ϕ, la matrice de pondération **w** est obtenue par simple résolution d'un système linéaire. Soit, en effet, un échantillon d'apprentissage $\left\{\left(\mathbf{x}^n, \zeta^n\right)\right\}_{n=1}^{N}$. On suppose que le Réseau à Fonctions de base Radiales comporte *M* sorties. On cherche les poids **w** qui minimisent le risque empirique:

$$R_{emp} = \sum_{n=1}^{N}\sum_{i=1}^{M}(y_i(\mathbf{x}^n) - \zeta_i^n)^2 = \sum_{n=1}^{N}\sum_{i=1}^{M}\left(\sum_{k=1}^{J} w_{ik}\phi_k(\mathbf{x}^n) - \zeta_i^n\right)^2 \qquad [89]$$

Les conditions d'optimalités sont donc obtenues en annulant la dérivée de cette quantité par rapport à w_{ik}, on obtient alors :

$$\sum_{n=1}^{N}\sum_{k'=1}^{J} w_{ik'}\phi_{k'}(\mathbf{x}^n)\phi_k(\mathbf{x}^n) = \sum_{n=1}^{N}\zeta_i^n\phi_k(\mathbf{x}^n) \qquad [90]$$

qui peut s'écrire sous la forme matricielle suivante :

$$(\mathbf{\Phi}^t\mathbf{\Phi})\mathbf{w}^t = \mathbf{\Phi}^t\boldsymbol{\zeta} \qquad [91]$$

Si la matrice $\mathbf{\Phi^t\Phi}$ est non singulière[11], la solution optimale pour les poids, à fonctions de base fixées, s'écrit :

$$\mathbf{w^t} = (\mathbf{\Phi^t\Phi})^{-1}\mathbf{\Phi^t}.\boldsymbol{\zeta} \quad \Rightarrow \quad \mathbf{w^t} = \mathbf{\Phi^{-1}}.\boldsymbol{\zeta} \qquad [92]$$

II.4. Réseaux de neurones et surveillance

Nous présentons dans cette partie une liste non exhaustive de quelques travaux sur les applications de surveillance industrielle. D'autres références non citées dans cette partie peuvent être trouvées dans (Koivo, 1994), (Bernauer, 1996), (Dubuisson, 2001). Ce que l'on peut retenir de cet état de l'art est que les réseaux de neurones sont : soit utilisés comme outil secondaire pour la surveillance, c'est-à-dire comme approximateur de fonctions pour l'identification des systèmes dynamiques grâce à une boîte noire neuronale ; soit comme outil principal de détection et diagnostic, en l'occurrence tous les travaux de classification. Les réseaux de neurones peuvent fournir, dans certains cas, des solutions plus intéressantes que les

[11] Condition respectée pour les fonctions gaussiennes (voir condition de (Micchelli, 1986)).

autres outils de surveillance, à condition de choisir judicieusement le type de l'architecture neuronale et surtout de bien mener le processus d'apprentissage.

Les avantages les plus importants que l'on peut attribuer à une application de surveillance par réseaux de neurones sont :

- Modélisation et estimation de fonctions non linéaires par apprentissage
- Fusion de données et Parallélisme
- Généralisation et reconstruction des signaux capteurs

II.4.1. Problème d'approximation de fonction et de prédiction

La relation entrées-sorties d'un réseau de neurones peut avoir pour objet, non pas de donner un diagnostic direct, mais de reconstruire une quantité utile à une décision ultérieure. La variable de sortie n'est donc pas une variable catégorielle mais une variable réelle. Les réseaux de neurones sont dans ce cas utilisés comme un approximateur de fonctions non linéaires liant les variables de sortie aux variables d'entrée du système industriel. Ils représentent donc une boîte noire non linéaire modélisant le système. Cette boîte noire peut être *dynamique* ou *statique*, selon la nature du réseau de neurones (réseaux de neurones *temporels* ou *statiques*). L'avantage d'une telle technique de modélisation est qu'aucune connaissance (mathématique, physique ou autre) du système à modéliser n'est nécessaire. Le réseau de neurones intègre implicitement ces connaissances à travers le processus d'apprentissage. La réussite d'une telle technique de modélisation est donc tributaire du processus d'apprentissage.

Les travaux présentés par (Böhme et *al.,* 1999) constituent une application très intéressante de détection et localisation des défauts capteurs d'une centrale d'épuration hydraulique. Ceci est effectué par la reconstruction des signaux capteurs avec une comparaison de deux architectures neuronales : le Perceptron Multi Couches comparé à la carte de Kohonen. Le *PMC* est constitué de cinq couches, six neurones d'entrée et six neurones de sortie. Son objectif est de reconstituer six mesures de six sorties capteurs après un apprentissage par rétro-propagation. Dans ce cas, ce réseau peut donc être considéré comme une mémoire auto associative. La détection ainsi que la localisation sont effectuées après une phase de comparaison avec seuillage de la sortie estimée avec la sortie réelle du capteur. Cette mémoire auto associative est donc comparée à la carte de Kohonen appelée carte topologique auto adaptative. Cette carte contient 15*15 neurones avec un vecteur d'entrée de 6 neurones (dimension de l'ensemble des sorties capteurs). Chaque neurone de la carte est caractérisé par un prototype et un paramètre définissant le rayon d'influence, déterminés par le processus d'apprentissage non supervisé. Pour chaque vecteur d'entrée, la réponse est donnée par un neurone gagnant qui correspond à celui dont la réponse de la fonction gaussienne est la plus importante. D'après la conclusion des auteurs, les deux techniques ont des performances similaires. Ces réseaux ont été testés sur deux types de fautes isolées et une succession de

deux fautes. La carte de Kohonen se montre plus rapide pour la détection et l'identification du capteur défaillant. La technique devient insignifiante si plus de 50% des variables d'entrée sont erronées.

Une autre application de surveillance d'un moteur utilisant le même principe est présentée dans (Petsche et *al.,* 1996). Un Perceptron à trois couches utilisé comme mémoire auto associative sert à reconstituer le spectre du courant électrique. Après un apprentissage par l'algorithme de rétro-propagation, la sortie de la mémoire auto associative est comparée à celle du spectre original. Tout écart supérieur à un seuil est synonyme de défaillance. Dans (Vemuri, 1997)-(Vemuri et *al.,* 1998) les réseaux de neurones ont été utilisés comme outils complémentaires à la modélisation d'un manipulateur robot. Un Perceptron à trois couches sert à estimer une fonction inconnue du modèle mathématique de fonctionnement du robot. Cette fonction représente des perturbations (défaillances) du manipulateur. L'apprentissage des paramètres du réseau s'effectue séquentiellement. La détection d'une défaillance se traduit par une génération de résidus. La variable de sortie du modèle de fonctionnement nominal (valeur estimée) est comparée à celle mesurée sur le robot. L'originalité de cette application se situe dans la modélisation non-linéaire du fonctionnement du manipulateur où les défaillances sont une fonction du temps, contrairement aux autres méthodes classiques où les modèles sont linéaires avec des défaillances additives. Dans (Lopes et *al.,* 1999) deux *PMC* à trois couches sont utilisés pour une application d'estimation et de prédiction de la qualité des pièces qui sortent d'un moule à injection. Un *PMC* sert à prédire (reconnaître) la classe représentant la qualité de ces pièces, et un autre *PMC* pour quantifier cette qualité. Les deux réseaux ont été appris par rétro-propagation (1000 itérations pour le premier et 6000 pour le deuxième).

II.4.2. Problème de discrimination ou de classification

Lors de la mise en place d'un système de surveillance par reconnaissance des formes, l'expert est censé connaître les modes de bon fonctionnement et certains des modes de défaillances. Une grande partie des modes de bon fonctionnement est généralement fournie par les données du constructeur de l'équipement. Par contre, les informations concernant les modes de défaillance peuvent provenir de deux origines différentes : soit fournies par le constructeur ou par le bureau des études (provenance de haut), soit collectées en cours de fonctionnement de l'équipement (provenance de bas). Ces connaissances sont emmagasinées dans un historique de fonctionnement (base de données). Celui-ci contient les différentes relations de "*causes à effets*" des situations de dysfonctionnement d'un équipement. L'opération de diagnostic menée par l'expert est souvent très complexe[12] et demande des connaissances ainsi qu'un raisonnement, généralement difficiles à formaliser. Les informations contenues dans l'historique de fonctionnement, représentent la base d'apprentissage supervisé du réseau de neurones (Figure 26). La réussite d'une telle

[12] Complexité qui dépend du type de défaillance et aussi de la complexité de l'équipement.

application est donc tributaire de la qualité des informations contenues dans l'historique de fonctionnement.

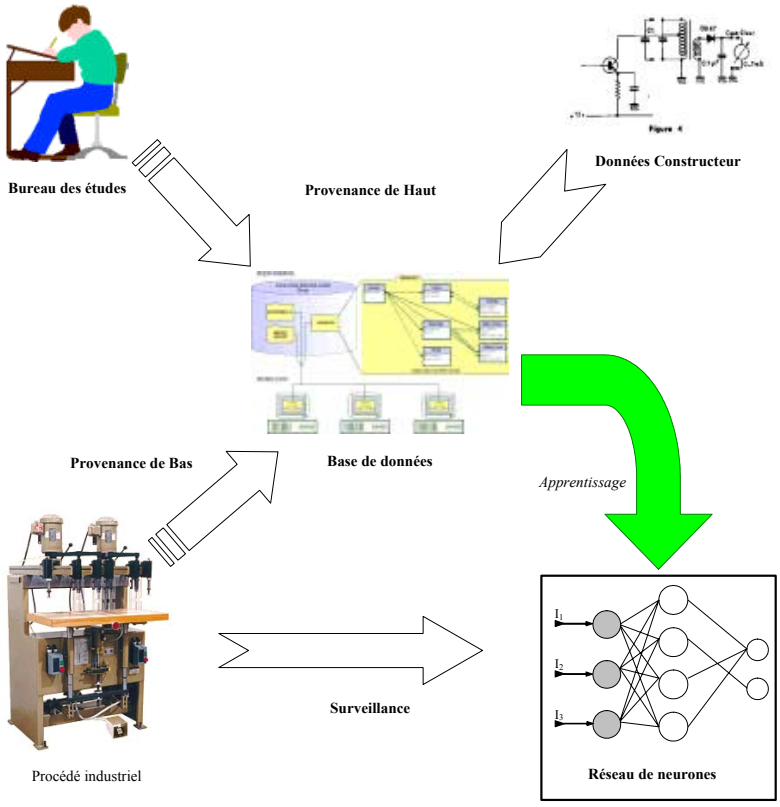


Figure 26. *Sources des informations d'apprentissage.*

Les variables d'entrée peuvent être constituées par les différents paramètres mesurés sur le procédé. On cherche à associer un mode de fonctionnement (fonctionnement nominal, dégradé, …) à ces variables d'entrée. Les variables de sortie sont alors des variables catégorielles où chaque catégorie représente un mode de fonctionnement. La relation entrées-sorties représente dans ce cas directement l'opération de diagnostic (Dubuisson, 2001). L'application décrite dans (Terstyanszky et *al.,* 2002) est caractéristique de cette démarche. Un réseau *RFR* est utilisé pour le diagnostic de défaillance d'un véhicule mobile autonome (*AMV* : Autonomous Mobile Vehicle). Le réseau de neurones associe cinq modes de défaillance à neuf variables mesurées en entrée. Les auteurs mettent en évidence les capacités du réseau *RFR* à reconnaître des nouvelles situations jusqu'alors non apprises. Cette capacité de généralisation locale des réseaux *RFR* est souvent appréciée dans les applications de surveillance industrielle.

Un autre exemple d'application des réseaux de neurones en surveillance par reconnaissance des formes est présenté dans (Keller et *al.,* 1994). Les auteurs ont utilisé deux

réseaux de neurones pour la surveillance de l'environnement : le premier réseau de neurones de type *PMC* a pour but d'identifier et de quantifier les vapeurs chimiques de l'air. Le réseau de neurones associe neuf classes de sortie (huit classes représentant huit gaz ménagers et une classe de rejet en distance) à une signature de douze capteurs (neuf capteurs de gaz, un capteur de d'humidité et deux capteurs de température). L'apprentissage du réseau à trois couches (12 neurones d'entrée, 6 neurones cachés et 9 neurones de sortie) par l'algorithme de rétro-propagation reste tout de même assez gourmand en temps (15000 itérations). Le deuxième réseau de neurones permet d'identifier et de quantifier un certains nombre d'isotopes radioactifs à partir des capteurs de radiations. Les auteurs ont utilisé une mémoire associative linéaire[13] à deux couches : une couche d'entrée de 512 neurones et une couche de sortie de 8 neurones. Chaque neurone d'entrée est excité par l'amplitude d'une fréquence parmi les 512 fréquences d'un spectre de rayons gamma. L'avantage d'une telle application est de donner une réponse sur le type et la proportion d'isotopes contenue dans un objet et ceci en prenant en compte l'ensemble des fréquences du spectre. Cette propriété de fusion de données représente également un atout non négligeable pour les applications de surveillance par réseaux de neurones (Hashem et *al.*, 1995).

On peut trouver également dans (Wu et *al.,* 1994) et (Meador et *al.,* 1991) une application de diagnostic de défaillances de circuits intégrés par reconnaissance des formes. Les auteurs ont comparé les performances d'un *PMC* face à deux techniques de classification : l'estimateur gaussien avec le maximum d'appartenance et l'estimateur des *k* plus proches voisins. Les résultats de cette application semblent être plus favorables au *PMC*. La raison principale de ce résultat est que le *PMC* tend vers une structure qui minimise l'erreur quadratique moyenne lors du processus d'apprentissage. Cette caractéristique démarque l'utilisation des réseaux de neurones dans les problèmes de classification, d'autant plus que la procédure de calcul de la matrice de covariance de l'estimateur gaussien peut s'avérer très complexe. Le seul point négatif relevé par les auteurs est la lourdeur du temps d'apprentissage du Perceptron Multi Couches.

(Hines et *al.,* 1995) ont publié un travail assez original sur la surveillance des équipements d'énergie nucléaire. Leur méthode est basée sur une technique hybride qui combine méthode analytique à base de modèle de l'équipement et une technique neuronale qui permet d'identifier la défaillance à partir des résidus obtenus par les techniques classiques de générations de résidus. Le réseau de neurones utilisé est un réseau à trois couches. Le nombre des neurones en entrée du réseau est donc égal au nombre de résidus. Chaque neurone de sortie représente une classe de fonctionnement (mode de fonctionnement de l'équipement). Les auteurs semblent hésiter entre deux types d'architectures neuronales : le Perceptron Multi Couches et le Réseau à Fonctions de bases Radiales. Le *PMC* est considéré comme une méthode de discrimination non-linéaire ; par contre, le *RFR* ne couvre qu'une partie de son espace de données en fonction de ce qu'il a appris. Ce réseau est donc capable de dire « *je ne sais pas* », contrairement au *PMC* qui peut mal classer un mode nouvellement rencontré et

[13] Réseau de neurones dans lequel tous les neurones ont une fonction d'activation linéaire.

induire ainsi l'expert en erreur. Les auteurs ont tout de même choisi le *PMC*, malgré les arguments précédemment évoqués. La raison essentielle de leur choix est que leur base de connaissance (base d'apprentissage) était assez exhaustive pour couvrir la quasi-totalité de l'espace des données d'entrée. Cette technique hybride semble donner des résultats assez satisfaisants. L'intérêt majeur des réseaux de neurones dans cette application par rapport aux autres méthodes classiques de détection des défaillances, concerne la résolution du problème de la redondance des alarmes.

Une autre application de détection et diagnostic par reconnaissance des formes sur un problème d'hypovigilance d'un conducteur est présentée dans (Hernandez, 1999). Les auteurs ont utilisé deux techniques hybrides : la technique des Ensembles Flous Multidimensionnels[14] et la technique des Réseaux de neurones à base de Fonctions Radiales Généralisés[15]. Le problème consiste à associer des modes de conduite (normal, fatigue, alcool et inattention) à un ensemble de sorties capteurs. Ce problème devient très complexe si l'on souhaite modéliser le comportement du conducteur. Les techniques de l'*IA* sont donc indispensables dans une telle situation où les connaissances du système sont très minimes. Le traitement des chevauchements des classes de la base d'apprentissage (base de données superposées) est l'une des raisons du choix des auteurs pour les deux outils cités précédemment. Un test de chevauchement et décision finale permet de quantifier la proportion du chevauchement des classes de sortie. Les performances des deux techniques (*EFM* et *RFRG*) semblent être identiques avec un taux de réussite du diagnostic compris entre 78% et 90%.

II.4.3. Réseaux de neurones et surveillance dynamique

Les applications précédentes montrent que les réseaux de neurones peuvent fournir des solutions très intéressantes pour les problèmes de surveillance sans modèle. Toutefois, un aspect fort important n'a pas encore été abordé jusque là, à savoir la dynamique du système à surveiller. En effet, toutes les applications présentées ne traitent que l'aspect statique des données capteurs. Ce type de traitement ne permet pas d'apprendre la dynamique d'un signal capteur afin de prédire une défaillance, ni même d'apprendre des séquences de fonctionnement d'un système à événements discrets. Les réseaux de neurones temporels (qui feront l'objet du chapitre suivant) offrent cette possibilité de prise en compte de l'aspect temporel des données.

Nous pouvons citer quelques travaux de surveillance par reconnaissance des formes avec réseaux de neurones temporels. Les travaux présentés dans (Bernauer et *al.,* 1993) et (Demmou et *al.,* 1995) traitent d'un problème de détection et diagnostic d'une cellule flexible d'assemblage. Un réseau de neurones récurrents permet d'apprendre des séquences temporelles booléennes qui représentent des gammes de bon fonctionnement. Après une

[14] Multidimensional Fuzzy Sets
[15] Generalized Radial Basis Functions

phase d'apprentissage assez complexe, le réseau est capable de détecter des situations de défaillance système et de localiser l'origine de la défaillance. L'article présenté par (Rengaswamy et *al.,* 1995) constitue une autre application très intéressante des réseaux de neurones pour la surveillance dynamique d'une unité de craquage catalytique fluide (*FCCU*[16]). Un Perceptron à trois couches permet de prendre en compte l'évolution temporelle des signaux capteurs. 21 défaillances sont associées aux variations de 7 sorties capteurs. Le diagnostic s'effectue grâce à une base de connaissance sous forme d'un arbre de défaillance reliant les effets constatés sur le système aux causes potentielles (les 21 défaillances du système). Les effets constatés sur le système sont caractérisés par le type d'évolution des sorties capteurs (brusque changement, évolution exponentielle, faible palier de dégradation, important palier de dégradation, …etc.). Cette classification des signaux en fonction de leur évolution est effectuée par des réseaux de neurones à trois couches, un réseau par signal capteur. L'aspect temporel des signaux est pris en compte par cinq neurones d'entrée. Chaque neurone d'entrée représente une réponse d'un capteur pris à différents intervalles de temps (*x(t),..x(t-4)*). Chaque neurone de la couche de sortie représente la classe du type d'évolution du signal. Le réseau est capable de distinguer sept classes différentes appelées primitives : trois droites (horizontale, croissante et décroissante) et quatre courbes (deux croissantes -convexe et concave- et deux décroissantes -convexe et concave-). L'apprentissage de 100 formes par l'algorithme de rétro-propagation a nécessité prés de 5000 itérations. La technique a été testée sur 63 cas de défaillances avec, pour chaque défaillance, trois degrés de sévérité. La technique semble donner de bons résultats en diagnostic, par contre aucun degré de sévérité n'a pu être estimé.

Les réseaux de neurones peuvent aussi être des outils assez efficaces pour des applications de prédiction ou pronostic où l'aspect dynamique (temporel) joue un rôle très important. Les résultats présentés dans (Chang et *al.,* 2001) semblent être très prometteurs pour des applications de prédiction neuronale des systèmes non-linéaires. Les auteurs ont appliqué un réseau *RFR* avec une fenêtre temporelle pour la prédiction des inondations d'un important fleuve situé à Taiwan sur un horizon de trois heures. Les paramètres du réseau sont déterminés en deux phases : une phase non supervisée avec l'algorithme Fuzzy Min-Max et une phase supervisée avec la méthode de régression linéaire. Le réseau donne d'assez bons résultats sur plusieurs points tests. Nous pouvons encore citer les travaux de (Freitas et *al.,* 1999) pour la détection des défaillances d'un actionneur pneumatique. Le principe de la méthode consiste à comparer la sortie estimée avec celle du système réel. Après avoir intégré le résultat de la comparaison, le signal de sortie du système et le signal de commande, un PMC a pour rôle de reconnaître les modes de fonctionnement à partir de ces trois derniers signaux. L'estimation de la sortie de l'actionneur est assurée par un réseau de neurones récurrent (dynamique). L'estimation d'un tel système peut être très complexe avec d'autres outils mathématiques classique, car le système est non seulement non linéaire mais loin d'être périodique. La modélisation se fait par apprentissage. Notons que les réseaux de neurones statiques sont incapables de modéliser un tel système.

[16] Fluidized Catalytic Cracking Unit

II.5. Conclusion

Ce chapitre a été dédié à la présentation des réseaux de neurones artificiels pour la surveillance des équipements industriels. Après avoir introduit leurs concepts de base, nous avons présenté l'application des réseaux de neurones pour la surveillance industrielle. Les réseaux de neurones sont alors utilisés selon deux façons différentes : une application de reconnaissance des formes pour la surveillance (détection du mode) et une application d'identification des systèmes dynamiques pour la surveillance (pronostic). Dans le premier type d'application (reconnaissance des formes), les réseaux de neurones associent un mode à chaque ensemble de données quantifiables (sorties capteurs) ou qualifiables (observations sur le système). Le mode reconnu est caractérisé par l'état de fonctionnement du système (bon fonctionnement, défaillance, etc.). Dans le deuxième type d'application (approximation de fonctions), les réseaux de neurones sont utilisés pour donner un modèle de l'équipement sous forme d'une boîte noire. Nous avons particulièrement détaillé deux architectures neuronales : le *Perceptron Multi Couches* (*PMC*) et les *Réseaux à Fonctions de base Radiales* (*RFR*).

Pour les deux types d'application (reconnaissance des formes et approximation de fonctions), nous nous sommes basés sur deux critères pour le choix du type de réseau de neurones pour le développement d'un outil de surveillance neuronal dynamique paramétrable via la couche TCP/IP : le premier critère concerne la couverture locale de l'espace des connaissances, le deuxième concerne la prise en compte de l'aspect dynamique (temporel) des données du système.

Des différences considérables existent entre le *PMC* et les *RFR*. Le *PMC* est caractérisé par une approche globale tandis que les *RFR* ont une approche locale. Une première conséquence directe de cette différence, en classification, est que les *RFR* sont capables de dire « *je ne sais pas* » contrairement au *PMC* qui donne une réponse (même mauvaise) quel que soit le vecteur d'entrée.

Cette caractéristique propre aux *RFR* est très importante en surveillance industrielle. Les bases de connaissances du fonctionnement des équipements industriels sont rarement exhaustives. Le système de surveillance se doit donc de pouvoir reconnaître les nouveaux modes susceptibles d'être rencontrés tout au long du fonctionnement de l'équipement. Les *RFR* attirent ainsi l'attention de l'expert humain sur un nouveau mode rencontré alors que le *PMC* a tendance à donner une mauvaise réponse et l'expert ne saura jamais qu'un nouveau mode a été rencontré.

Les Réseaux de neurones à base de Fonctions Radiales laissent une part importante à l'expert humain lors de l'occurrence d'un nouveau mode. L'expert peut ainsi analyser le nouveau mode et le faire apprendre au réseau sans vraiment modifier les informations apprises au préalable. Cette caractéristique de mise à jour est aussi une deuxième conséquence de l'architecture locale des *RFR*. En effet, chaque neurone est associé à une région d'activation et une partie des neurones participe à la réponse du réseau. L'apprentissage d'un

nouveau vecteur n'aura d'influences que sur le voisinage proche du nouveau neurone mémorisant ce nouveau vecteur. Cette deuxième caractéristique importante ne s'applique pas aux architectures globales (*PMC*) car l'ensemble des neurones du réseau participe à l'élaboration de la sortie. Dans ce cas, si le réseau apprend dans une région de l'espace des données, le modèle peut *oublier* ce qu'il a appris dans d'autres régions. Le PMC se montre ainsi moins robuste.

Ces arguments justifient le choix des *RFR* comme architecture neuronale la mieux adaptée aux problématiques de surveillance industrielle, d'autant plus que l'apprentissage des *RFR* est beaucoup plus souple et moins coûteux en temps de calcul. Par ailleurs, grâce à leur approche locale, les RFR présentent une sûreté de fonctionnement intéressante en surveillance.

Le deuxième critère tout aussi important en surveillance est la prise en compte de la dynamique du système. Cette dynamique permet de mieux identifier les modes de défaillance (fausses alarmes et vraies dégradations) et de pouvoir anticiper sur l'évolution d'un équipement (surveillance préventive). Nous concluons l'état de l'art des réseaux de neurones en surveillance en montrant que certaines fonctions ne peuvent être réalisées que par la prise en compte de l'aspect temporel.

Le chapitre suivant approfondit la notion de réseaux de neurones dynamiques. Il a pour objet la présentation des réseaux de neurones temporels, avec leur représentation spatiale et dynamique du temps.

Chapitre III

Représentation du temps dans les réseaux de neurones

Résumé : *Les réseaux de neurones présentés au chapitre précédent peuvent offrir des solutions très intéressantes dans des applications de reconnaissance des formes ou approximation de fonctions mais ne peuvent en aucun cas être appliqués sur des données où le temps joue un rôle déterminant dans la résolution du problème. Comment ces réseaux de neurones statiques peuvent être adaptés pour être dynamiques ? Quelles sont les architectures de réseaux temporels qui existent en littérature ? Comment est mené l'apprentissage pour la prise en compte de la dimension temporelle des données ? Nous essaierons de répondre à ces questions à travers ce troisième chapitre consacré à la représentation du temps dans les réseaux de neurones.*

Abstract : *The Neural networks exposed in the previous chapter may offer very interesting solutions in pattern recognition or function approximation, but cannot be applied to dynamic data. How these static neural networks can be transformed in temporal architecture? Which are the existing temporal network architectures in the literature? How the training process takes into account the temporal dimension of the data? We will try to answer to these questions through this third chapter devoted to the representation of the time in neural networks.*

Chapitre III

Représentation du temps dans les réseaux de neurones

III.1. Introduction

Certaines fonctions de la surveillance industrielle, telle que la détection prédictive qui consiste à reconnaître le type de dégradation en fonction de l'évolution d'un signal (Zemouri *et al.,* 2003 -a-) et aussi à éliminer les fausses alarmes (Zemouri *et al.,* 2002-b-), nécessitent la prise en compte d'un certain « *passé* » des signaux capteurs. Ce passé est nécessaire si l'on veut prédire l'évolution d'un signal à l'instant $t + \Delta t$ afin de prendre de l'avance sur les actions à entreprendre soit sur un équipement, comme la détection précoce des défaillances d'une colonne de distillation (Ploix *et al.,* 1997), soit sur tout un système donné comme la prédiction du trafic de la téléphonie mobile (Legrand *et al.,* 2002). La surveillance d'un système à événements discrets par réseau de neurones peut également exiger la prise en compte de l'aspect temporel pour l'apprentissage de séquences booléennes (Bernauer, 1996). Une autre application importante des réseaux de neurones nécessitant la prise en compte de la dimension temporelle est constituée par l'identification des systèmes dynamiques non linéaires (Urbani, 1995)-(Yu *et al.,* 2001)-(Mirea *et al.,* 2002)-(Ferariu *et al.,* 2002). Les applications où l'intégration du temps dans les réseaux de neurones est nécessaire sont bien évidemment plus larges que le domaine de la surveillance. Nous pouvons citer par exemple la reconnaissance en ligne de l'écriture (Garcia-Salicetti, 1996)-(Chappelier, 1996) ou de la parole (Mellouk, 1994)-(Adamson *et al.,* 1996), ainsi que la prédiction de séries temporelles (Aussem, 1995)-(Mangeas, 1996)- (Atiya *et al.,* 1999)-(Rynkiewi, 2000)-(Sinha *et al.,* 2002) très utile pour la problématique des prédictions financières (Mozer, 1993).

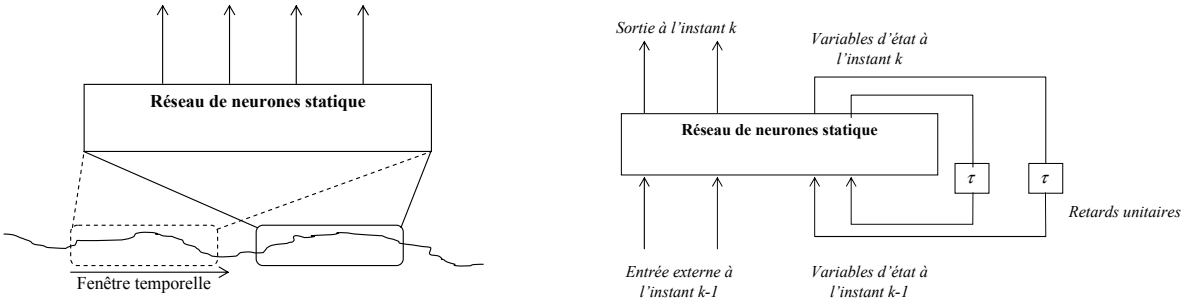
Les réseaux de neurones que nous avons présenté au chapitre précédent ne peuvent être appliqués que sur des données statiques. Comment ces réseaux de neurones statiques peuvent être adaptés pour être dynamiques ? Quelles sont les architectures de réseaux temporels qui existent en littérature ? Comment est mené l'apprentissage pour la prise en compte de la dimension temporelle des données ? Nous essaierons de répondre à ces questions à travers ce troisième chapitre entièrement consacré à la représentation du temps dans les réseaux de neurones.

Les réseaux de neurones temporels sont divisés en deux grandes familles : les réseaux à représentation ***externe*** du temps, et ceux à représentation ***interne***. Nous nous sommes principalement intéressés aux ***réseaux récurrents*** qui sont une sous catégorie des réseaux à représentation interne du temps. Nous argumenterons le choix des réseaux de neurones récurrents par une analyse critique des deux façons de représenter le temps.

III.2. Architectures Neuronales Temporelles

La prise en compte de l'aspect temporel des données par les réseaux de neurones artificiels nécessite certaines modifications architecturales des modèles neuronaux statiques présentés au chapitre précédent. Il existe en littérature deux façons distinctes d'aborder le temps par les réseaux de neurones (Chappelier *et al.*, 1996), (Chappelier, 1996) : dans la première, le temps est représenté comme un ***mécanisme externe*** au réseau de neurones. Des retards (ou temporisations) servent à mémoriser les données d'entrée pendant une certaine durée τ_i. On présente au réseau un vecteur d'entrée comportant les données à l'instant *t* et ceux des instants $t-\tau_i$ (avec $i=1,...N$). On obtient alors une fenêtre temporelle de taille *N+1*. Cette technique a l'avantage de pouvoir utiliser les architectures de réseaux de neurones statiques. La prise en compte de l'aspect temporel est complètement transparente (Figure 27.a). Cette technique est aussi appelée ***représentation spatiale*** du temps selon *Elman* (Elman, 1990). Par contre, dans la deuxième façon de prendre en compte le temps, le réseau de neurones est capable de traiter le temps sans aucun mécanisme externe (Figure 27.b). Cette représentation est appelée ***représentation interne*** selon *Chappelier* et ***représentation dynamique*** selon *Elman*. Nous présentons sur la Figure 28 les différentes façons de prendre en compte le temps dans les réseaux de neurones selon *Chappelier* et *Grumbach*. On peut également voir que la représentation interne se divise en deux possibilités : soit que le temps est pris en compte implicitement par la récurrence des connexions (réseaux de neurones récurrents), soit qu'il est pris en compte d'une manière explicite. Dans ce dernier cas, deux types de réseaux existent : dans le premier cas, les temporisations apparaissent au niveau des connexions. Dans ce type de réseaux, les connexions entre neurones possèdent non seulement des pondérations mais aussi des retards τ_i. L'apprentissage de ces réseaux consiste alors à trouver les valeurs des pondérations et des retards. Le deuxième cas de la prise en compte explicite du temps se situe au niveau du neurone. On trouve alors soit des modèles biologiques ayant le souci de reproduire des comportements biologiques des neurones, soit des modèles où le temps est pris en compte par des mécanismes algébriques afin de résoudre des problèmes d'ingénierie, sans forcement se soucier de l'aspect biologique.

Sans souci d'exhaustivité, nous avons donné quelques architectures de réseaux de neurones temporels pour chaque façon de représenter le temps. Ces types de réseaux temporels sont encadrés sur la Figure 28. Certains types de réseaux seront détaillés dans ce chapitre, tandis que d'autres seront simplement cités en indiquant les références bibliographiques correspondantes.



- a - - b -

Figure 27. *Exploitation des réseaux statiques pour des traitements dynamiques par utilisation de fenêtre temporelle (a) ou par utilisation de connexions récurrentes (b).*

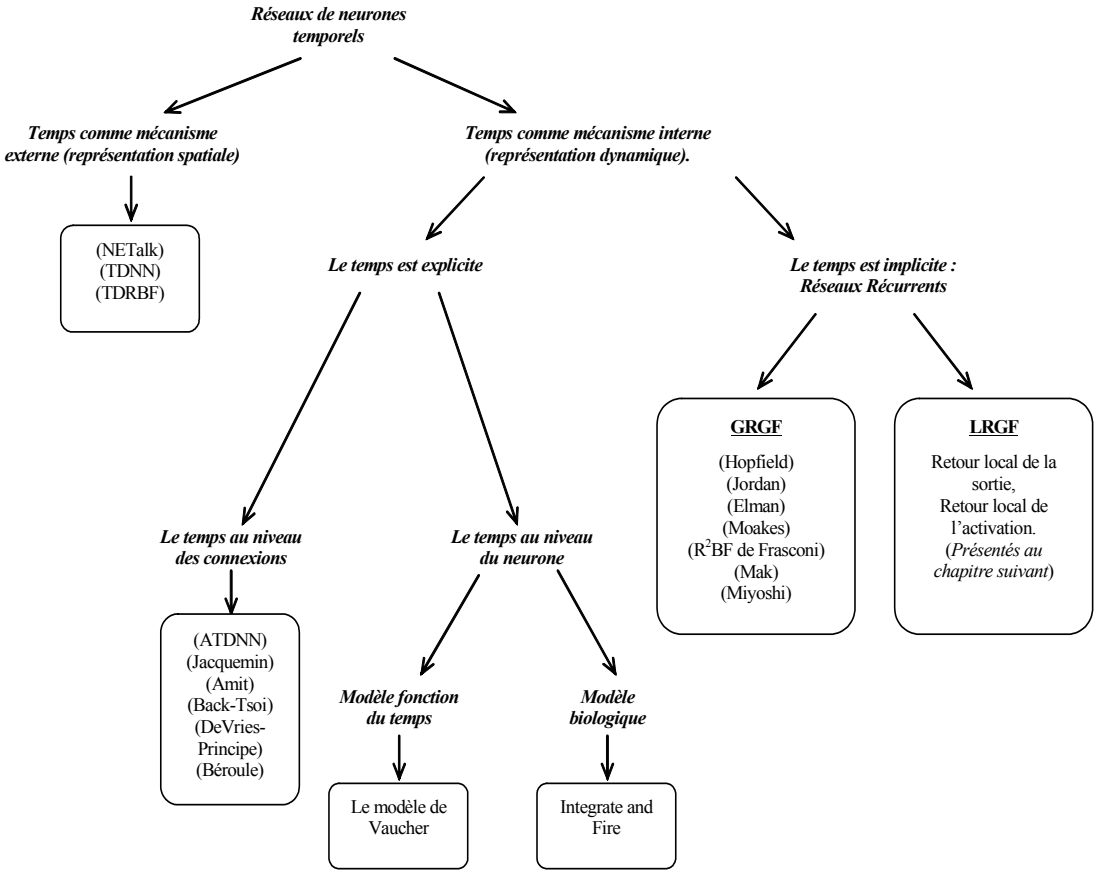


Figure 28. *Représentation du temps dans les réseaux de neurones. Nous détaillerons dans ce chapitre les représentations Globalement Récurrentes Globalement Feedforward (GRGF). Les représentations Localement Récurrentes Globalement Feedforward (LRGF) font l'objet du chapitre suivant (Chapitre IV).*

III.2.1. Représentation spatiale du temps

La façon la plus simple et immédiate de représenter le temps dans les réseaux de neurones est d'utiliser une représentation spatiale du temps. L'information temporelle contenue dans les données est alors transformée en une information spatiale, c'est à dire une forme qu'il s'agit de reconnaître. Dès lors, les techniques de classification par réseaux de neurones habituellement employées deviennent applicables. Cette transformation du temporel en spatial s'obtient par l'utilisation classique de ligne à retard. Au lieu de présenter au réseau chaque événement, dès son apparition, il convient d'attendre un certain temps avant de procéder à la classification de la forme obtenue. Chaque retard temporel représente une dimension de la représentation spatiale. Ce type de représentation du temps fait donc appel à un mécanisme externe qui est chargé de retarder ou de retenir un certain temps les données, ce qui conduit à l'appeler également représentation externe du temps. Nous présentons dans cette partie trois architectures neuronales utilisant ce principe : le *NETtalk*, le *TDNN* et le *TDRBF*.

III.2.1.1. NETtalk (Sejnowski *et al.*, 1986)

Le *NETtalk* est l'une des premières applications des réseaux de neurones dans le domaine du traitement de la parole. Le but est d'apprendre à un réseau de neurones à prononcer un texte en anglais à partir des phrases proposées, lettre par lettre, à l'entrée du réseau. *NETtalk* utilise une représentation spatiale du temps sous la forme d'une fenêtre temporelle d'une longueur de 7 lettres. L'objectif est alors de prononcer correctement le phonème qui se trouve au centre de la fenêtre. Le réseau est constitué d'une couche d'entrée de 7x29 neurones (chaque lettre est codée sur 29 neurones), d'une couche cachée de 80 neurones et d'une couche de sortie de 26 neurones (Figure 29). L'apprentissage est réalisé avec l'algorithme de rétropropagation du gradient.

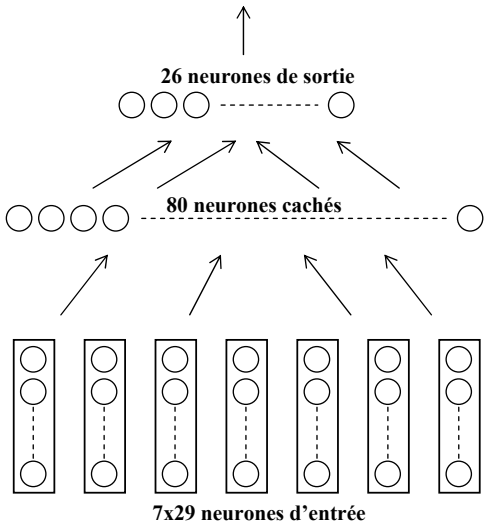


Figure 29. *Architecture du NETtalk.*

III.2.1.2. TDNN (Time Delay Neural Networks) (Waibel *et al.*, 1989)

En 1989, l'équipe de A. Waibel (Waibel *et al.*, 1989) a voulu introduire une architecture neuronale appliquée toujours au domaine de la reconnaissance de la parole. Ce réseau de neurones devait satisfaire certaines conditions parmi lesquelles :

- le réseau doit contenir suffisamment de couches et d'interactions entre ces couches pour pouvoir apprendre des données complexes non linéaires,
- le réseau doit être capable de représenter les relations temporelles entre les données d'entrée,
- le réseau doit être insensible aux positions temporelles absolues des données,
- l'algorithme d'apprentissage ne doit également pas prendre en compte les positions absolues des données,
- le nombre de poids dans tout le réseau doit être considérablement petit par rapport au nombre des données d'apprentissage. Cette condition permet au réseau d'encoder certaines régularités des données.

Le principe de base de la prise en compte de l'aspect temporel par le *TDNN* est d'utiliser des retards temporels τ_n (*Time Delay*) pour chaque neurone (Figure 30), où *n* varie de *0* à *N*. Ce dernier effectue alors une sommation pondérée par les poids de ses connexions amont, et donne en réponse la sortie de sa fonction d'activation (la sigmoïde). La Figure 31 montre la topologie complète du *TDNN* telle utilisée par A. Waibel (Waibel *et al.*, 1989) pour l'apprentissage de trois phonèmes : « B », « D » et « G ». La première couche contient 15 vecteurs. Chaque vecteur contient 16 acquisitions de parole à différentes fréquences comprises entre 141 Hz et 5437 Hz (donc un total de 240 neurones). Cette couche est entièrement connectée à la première couche cachée dont les neurones ont des retards de *N=2*. En d'autres termes, chaque neurone de cette couche est connecté à 48 neurones amont. Cette première couche cachée est également entièrement connectée à la deuxième couche cachée qui contient des retards de *N=4* ; donc chaque neurone de cette couche est connecté à 40 neurones amont. La dernière couche permet d'intégrer le résultat obtenu sur l'ensemble des données d'entrée. Chaque phonème est représenté par un neurone de sortie connecté à 9 neurones amont.

L'apprentissage du *TDNN* est réalisé avec l'algorithme de rétropropagation. Les auteurs n'ont pas utilisé la méthode classique de mise à jour des poids des connexions où chaque poids est ajusté par rapport à son gradient d'erreur. L'ensemble des connexions d'une même ligne[17] ayant le même retard *n* est ajusté par la même valeur. Cette valeur correspond à la moyenne des erreurs de tous les neurones de cette ligne (neurones ayant le même retard *n*). Si l'on considère par exemple les connexions entre les neurones des premières lignes de la

[17] Une ligne de neurones de chaque couche de la Figure 31

couche d'entrée et de la première couche cachée, le calcul de la variation moyenne des poids s'effectue comme suit :

$$\Delta w^n = \frac{1}{13}\sum_{i=1}^{i=13} \Delta w_i^n \qquad [93]$$

avec w_i^n qui représente le poids de la connexion du $i^{éme}$ neurone de la première ligne de neurones de la première couche cachée ayant un retard de *n* (Figure 31).

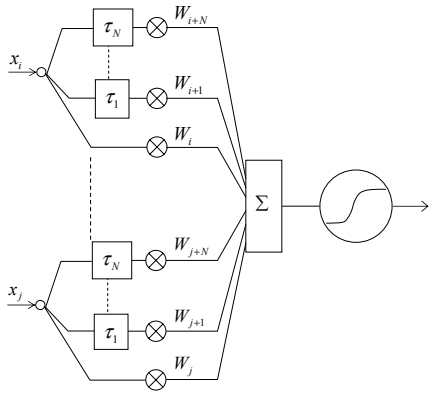


Figure 30. *Représentation d'une unité du TDNN. Des retards temporels τ_n sont appliqués à chaque entrée. Une fenêtre temporelle de taille N+1 est ainsi obtenue (n= 0,…,N).*

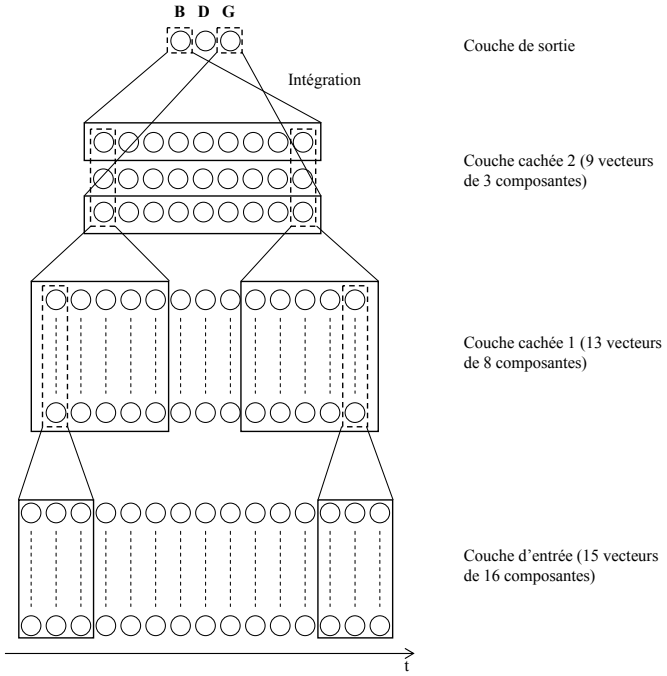


Figure 31. *Architecture du TDNN.*

L'apprentissage du *TDNN* reste tout de même assez lent, environ 50 000 itérations pour un apprentissage de 800 données. Le choix de la taille de la fenêtre temporelle de la couche d'entrée est particulier à ce type d'application ; par contre, la fenêtre de la première couche cachée a été choisie arbitrairement. Les auteurs ont comparé les performances du *TDNN* avec celles des chaînes de Markov cachées ou *Hidden Markov Model –HMM–*. Le *TDNN* a un taux de réussite de 98,5 % contre 93,7 % pour les *HMM*.

III.2.1.3. TDRBF (Time Delay Radial Basis Function) (Berthold, 1994 –a–)

Le *TDRBF* a été introduit par Berthold en 1994 (Berthold, 1994 –a–) pour la reconnaissance de phonèmes. Ce réseau combine les caractéristiques du *TDNN* et des *RFR*[18]. En d'autres termes, le *TDRBF* profite du traitement spatial des données d'entrée du *TDNN* et de la souplesse de l'apprentissage des *RFR*. Nous montrons sur la Figure 32 l'architecture du réseau. Une fenêtre temporelle constituée de plusieurs vecteurs d'entrée à différents instants ($t, t\text{-}1, ...,t\text{-}\Delta t$) peut être vue comme un seul vecteur de grande taille. Chaque neurone caché calcule alors la distance par rapport à son prototype et donne en sortie la réponse de sa fonction d'activation. Pour la simplicité du calcul, l'auteur a utilisé une fonction *porte* au lieu d'une fonction *gaussienne*. La sortie du réseau est obtenue par le calcul suivant :

$$d(t) = \sum_{\tau=0}^{\Delta t} \sum_{i=0}^{n-1} (x_i^{\tau} - \mu_{i,\tau})^2 \qquad [94]$$

d(t) est la distance euclidienne des vecteurs d'entrée (x_i^{τ} : i^{eme} composant du vecteur d'entrée à l'instant $t-\tau$) au vecteur prototype ($\mu_{i,\tau}$) du neurone *RFR*. La réponse de ce neurone en fonction d'un rayon σ est alors :

$$\Pi(t) = \begin{cases} 1 & d(t) \le \sigma \\ 0 & sinon \end{cases} \qquad [95]$$

L'inconvénient de cette architecture est sa forte sensibilité aux positions temporelles absolues. En d'autres termes, si on présente au réseau un vecteur déjà mémorisé mais décalé dans l'espace temps, le réseau est incapable de le reconnaître. Pour contourner cet obstacle, les auteurs ont proposé une couche supplémentaire appelée *couche d'intégration* qui permet d'intégrer dans le temps les sorties de chaque neurone à fonction de base radiale (Figure 33).

Les auteurs ont donc appelé ce réseau : *x(y)-TDRBF ;* où *x* représente la taille de la fenêtre temporelle que chaque neurone *RFR* doit traiter et *y* le nombre de fenêtres que la couche d'intégration doit intégrer. Cette structure peut être moins sensible aux positions absolues des

[18] Nous rappelons que l'appellation francophone des RBF est Réseaux à Fonctions de base Radiales.

données d'entrée. Les auteurs ont utilisé la sigmoïde comme fonction d'activation de la couche d'intégration.

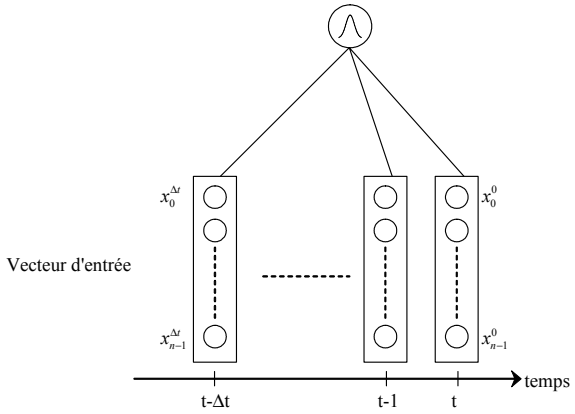


Figure 32. *Topologie du réseau TDRBF.*

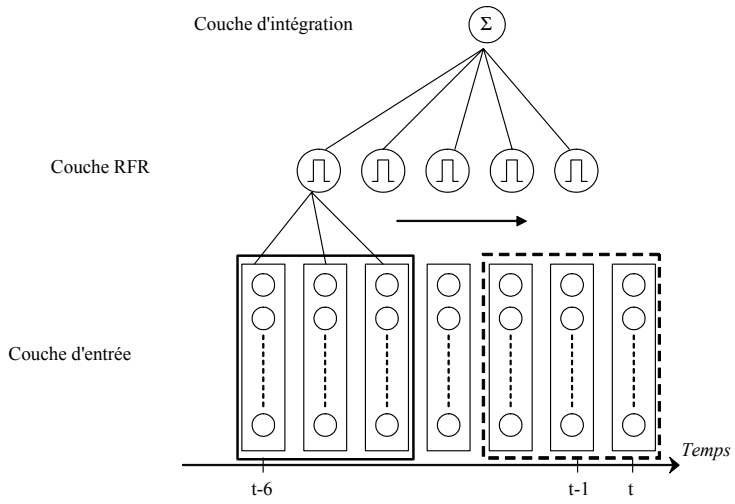


Figure 33. *La structure du 3(5)-TDRBF. Pour des raisons de simplicité, une seule classe a été représentée. En réalité, chaque classe a son neurone d'intégration.*

Le seul réglage nécessaire pour ce type de réseau correspond à trouver la relation entre l'espace temps de la couche *RFR* (noté *x* précédemment) et la taille de la fenêtre de la couche d'intégration (noté *y* précédemment). Ce dernier a été ajusté par tâtonnement. L'apprentissage du réseau consiste à déterminer les prototypes ainsi que les rayons d'influence des neurones *RFR*. Les auteurs ont utilisé l'algorithme *RCE* que nous avons décrit au chapitre précédent.

Le *TDRBF* a été testé sur une base de données de reconnaissances de phonèmes japonais. Plusieurs combinaisons *x(y)* ont été testées (de *1(15)* à *15(1)*). Les meilleurs résultats ont été obtenus avec les combinaisons *7(9)* et *8(8).* Ces résultats sont pratiquement semblables à ceux

obtenus avec le *TDNN* (taux de reconnaissance égal à : 98,3 % pour le *TDRBF* et 98,5 % pour le *TDNN*). Par contre, au niveau de la rapidité de convergence et de la simplicité de l'algorithme d'apprentissage, le *TDRBF* se montre beaucoup plus performant que le *TDNN* (10 itérations pour le *TDRBF* contre 50 pour le *TDNN*). Notons qu'une itération d'apprentissage du *TDRBF* et beaucoup plus courte que celle du *TDNN*. Le *TDRBF* est encore plus performant avec l'algorithme d'apprentissage *DDA* (Berthold, 1994 –b–) présenté au chapitre précédent (5 itérations d'apprentissage pour le *TDRBF* contre 50 pour le *TDNN*).

Pour finir, on peut également citer d'autres applications du *TDRBF* pour la reconnaissance d'expression du visage (Howell *et al.,* 1997) et de mouvement (Howell *et al.,*1998-a-), (Howell *et al.,*1998-b-).

III.2.2. Représentation dynamique du temps

Contrairement aux réseaux de neurones qui interprètent le temps comme un mécanisme *spatial* ou *externe,* les réseaux de neurones dits *dynamiques* traitent le temps d'une façon totalement interne au réseau. On peut distinguer deux manières de gérer le temps en interne : le temps est pris en compte *implicitement* ou *explicitement*. Dans les deux cas, le réseau possède la capacité de mémoriser des informations soit *implicitement* par la récurrence des connexions, soit *explicitement* par des retards au niveau des connexions. Nous présentons ces deux techniques internes de traitement du temps, tout en insistant sur les réseaux récurrents.

III.2.2.1. Représentation implicite du temps : Réseaux de neurones récurrents

La connectivité des unités dans les réseaux de neurones récurrents ne se limite pas, comme dans le cas des réseaux à propagation avant (*feedforward*), à des architectures dans lesquelles l'information se propage de l'entrée vers la sortie, couche après couche. Tout type de connexion est admis, c'est à dire d'un neurone à n'importe quel autre, y compris lui-même. En d'autres termes, lorsqu'on se déplace dans le réseau en suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ. Un tel chemin est désigné sous le terme de *cycle*. Une grandeur, à un instant donné, ne peut pas être fonction de sa propre valeur au même instant. Par conséquent, tout cycle du graphe des connexions d'un réseau de neurones récurrent doit comprendre au moins une connexion de retard unitaire (Dreyfus *et al.,* 2002). Ceci donne lieu à des comportements dynamiques fort intéressants mais qui peuvent être très complexes (Haykin, 1999). Contrairement aux réseaux de neurones statiques où cette dynamique est totalement absente [96], la loi d'évolution en temps continu des réseaux dynamiques peut être définie par l'équation différentielle [97] (Pearlmutter, 1990), (Warwick *et al.,* 1992).

$$y_i = f(a_i) \ , \ a_i = \sum_j (w_{ij} v_j) + \xi_i \qquad [96]$$

$$\beta_i \frac{dy_i}{dt} = -y_i + f(a_i) \ , \ a_i = \sum_{j=1}^{N} (w_{ij} y_j) + \xi_i \qquad [97]$$

où y_i représente la sortie du neurone *i*, *f(.)* sa fonction d'activation, ξ_i une entrée externe supposée constante, w_{ij} le poids de la connexion entre les neurones *i* et *j* et v_j de la relation [96] représente les entrées du neurone *i* provenant des neurones *j*. Ces entrées sont totalement indépendantes de y_i (uniquement les neurones amont) (Figure 34); par contre dans la relation [97], la récurrence des connexions fait que l'activation a_i du neurone *i* peut dépendre de toutes les sorties y_j de l'ensemble *N* des neurones du réseau (Figure 35).

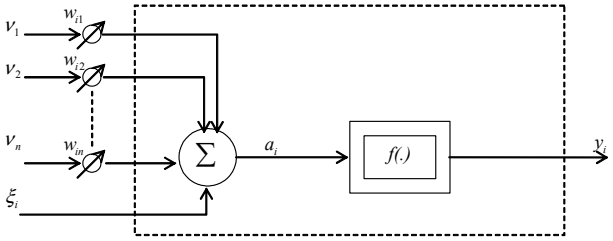


Figure 34. *Représentation du comportement statique d'un neurone statique.*

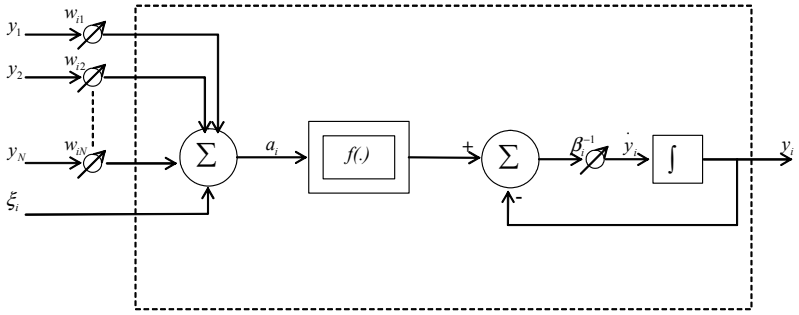


Figure 35. *Représentation du comportement dynamique d'un neurone récurrent.*

La représentation continue de la loi d'évolution d'un neurone dynamique (équation [97]) offre essentiellement l'avantage d'être efficace pour les calculs mathématiques (Pearlmutter, 1990). L'aspect implicite du temps est bien mis en évidence par l'équation différentielle. D'un autre côté, dès que l'on souhaite simuler un système dynamique continu sur un ordinateur, il est impératif de le discrétiser (Pearlmutter, 1990). L'équivalent de l'équation [97] en discret serait alors :

$$y_i(t+1) = f(a_i(t)) \qquad [98]$$

La dynamique du réseau de neurones sera donc fonction de la période d'échantillonnage du système. Cette période d'échantillonnage devra être choisie judicieusement afin de synchroniser la dynamique du réseau de neurones avec celle du système à surveiller. Nous nous sommes confrontés concrètement à ce type de problématique lors du développement d'une solution de surveillance temps réel (voir dernier chapitre). Nous expliquons alors en détail comment nous avons procédé pour résoudre ce genre de souci applicatif.

Les réseaux récurrents peuvent présenter deux types de comportements : le premier est que ces réseaux sont capables de se stabiliser dans un certain nombre de points de leur espace d'état, appelés points d'équilibre ou « *Fixed Point* ». Ces points constituent la réponse du réseau en présence d'une donnée et l'apprentissage consiste à affecter aux poids des connexions les valeurs permettant cette relaxation vers un point d'équilibre. L'architecture la plus connue et la plus ancienne ayant ce type de comportement est le modèle de Hopfield (Hopfield, 1982) (présenté au chapitre précédent). Bien que ce réseau soit généralement utilisé pour apprendre un certain nombre d'associations statiques pour des problèmes d'optimisation multicritères, le réseau de Hopfield possède une certaine dynamique de convergence, c'est-à-dire que le réseau passe par un certain nombre de points de son espace d'états avant d'atteindre un point d'équilibre. Ce type de réseau peut donc générer une séquence finie à partir d'un point donné de son espace d'état. Une variante de ce modèle a été proposée par Amit (Amit, 1988) pour la mémorisation de séquences temporelles. Son modèle se base sur celui de Hopfield pour stocker les diverses formes de la séquence et le niveau temporel, représenté par des retards, est capable d'enchaîner les formes apprises.

Le second comportement temporel que produisent les réseaux récurrents est une succession d'états ou plutôt de points dans l'espace d'états mais sans qu'il y ait stabilisation en un point particulier. Il peut s'agir, par exemple, d'un cycle limite au cours duquel le réseau passe cycliquement par certains états. L'apprentissage appelé *Trajectory Learning* consiste alors à donner aux poids des connexions les valeurs qui permettent au réseau de produire ce comportement particulier.

a) Principales architectures de réseaux récurrents

Architecture de Jordan (Jordan, 1986)

Dans l'architecture proposée par Jordan, les unités de la couche de sortie sont dupliquées sur une couche appelée *couche de contexte*. Les unités de cette couche tiennent également compte de leur propre état à l'instant précédent. Cette connexion récurrente d'une unité de contexte à elle-même lui donne une dynamique ou une mémoire individuelle. L'activation de chaque unité de cette couche est calculée selon l'équation suivante :

$$C_i(t+1) = \alpha C_i(t) + O_i(t) \qquad [99]$$

En supposant $\alpha<1$ et les sorties O_i fixes, les unités de la couche de contexte exhibent un comportement d'oubli où leur sortie décroîtrait vers $O_i/(1-\alpha)$, oubliant progressivement leurs états précédents.

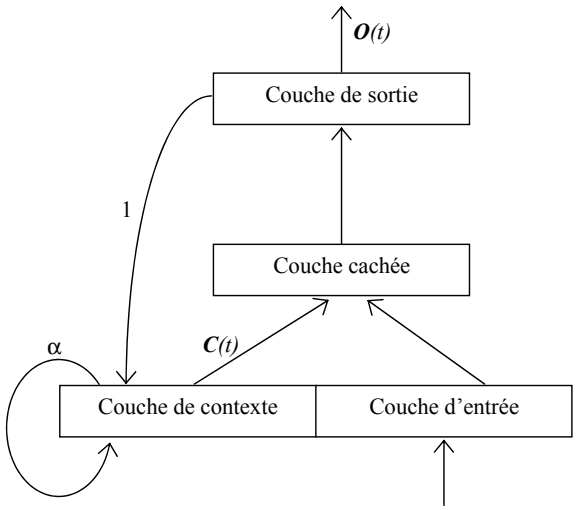


Figure 36. *Architecture de Jordan.*

Architecture d'Elman (Elman, 1990)

Elman s'est inspiré en grande partie du réseau de Jordan pour proposer son architecture (Figure 37). Cette fois-ci, ce sont les unités de la couche cachée qui sont dupliquées dans la couche contexte avec un poids unitaire. L'apprentissage s'effectue par l'algorithme de rétropropagation et ne concerne que les poids de propagation avant.

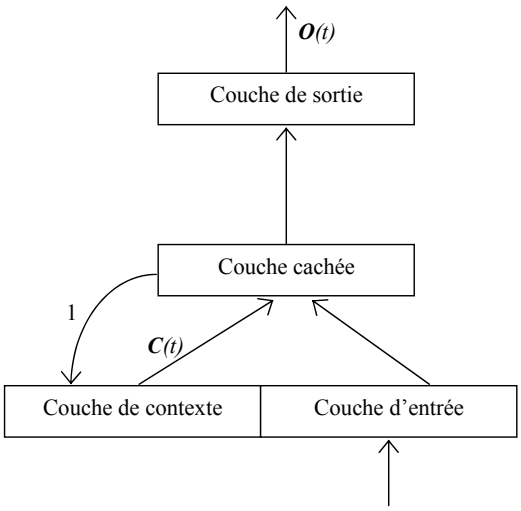


Figure 37. *Architecture d'Elman.*

Architecture de Moakes (Moakes et al., 1994)

Le modèle de la Figure 38 introduit par Moakes (Moakes *et al.,* 1994) représente une architecture particulière des réseaux *RFR* : Réseau Récurrent à Fonctions de base Radiales (*RRBF*)[19] appliqué au traitement dynamique de la parole. Ce réseau combine récurrence des connexions et fenêtre temporelle. Des retards τ_n et τ_m sont associés respectivement au signal d'entrée *s(k-1)* et au signal de sortie *y(k)* rebouclé sur l'entrée du réseau. La sortie *y(k)* représente la prédiction à l'instant *k* du signal d'entrée *s(k-1)*. Ce réseau peut être vu comme un réseau *RFR* statique avec le vecteur d'entrée suivant :

$$\mathbf{x}(k) = \left(s(k-1), s(k-2), ..., s(k-n), y(k-1), y(k-2), ..., y(k-m)\right) \quad [100]$$

où *n* et *m* représentent le nombre de retards associés respectivement à l'entrée *s* et au signal de sortie *y*.

Cette architecture de *RFR* récurrent a été utilisée par (Billings *et al.,* 1995) comme un filtre non linéaire de bruit.

[19] ***R**ecurrent **R**adial **B**asis **F**unction networks*

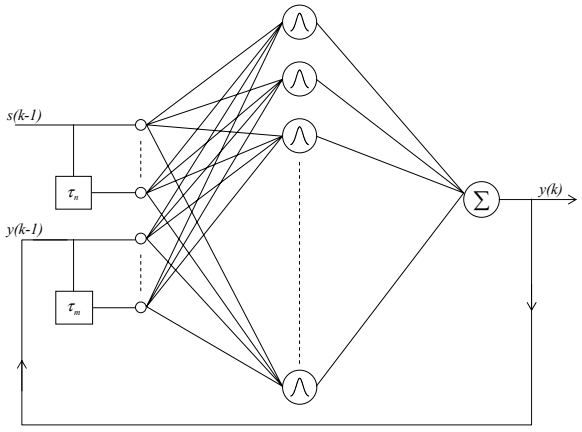


Figure 38. *L'architecture de Moakes pour un RFR récurrent.*

Architecture de M.W. Mak (Mak, 1995)

La Figure 39 présente une autre architecture de réseau *RFR* récurrent. La récurrence des connexions se situe au niveau des sorties de chaque neurone gaussien. La réponse de sortie du réseau à l'instant *t* pour chaque vecteur d'entrée **x***(t)* est :

$$y(t) = \sum_{i=1}^{M} w_i h_i(\mathbf{x}(t)) \qquad [101]$$

avec :

$$h_i(\mathbf{x}(t)) = \phi_i(\|\mathbf{x}(t) - \boldsymbol{\mu}_i\|) + \sum_{k=1}^{M} u_{ik} h_k(\mathbf{x}(t-1)) \qquad [102]$$

où w_i représente le poids des connexions de sortie, $\phi_i(\|.\|)$ est la réponse gaussienne des neurones cachés et u_{ik} le poids des connexions récurrentes. Les paramètres des fonctions gaussiennes sont déterminés par l'algorithme des *k-moyennes ;* par contre, les poids des connexions récurrentes et de sortie sont déterminés par l'algorithme de rétropropagation du gradient. Le réseau a été testé sur une application d'apprentissage de séquences temporelles réelles d'une réponse impulsionnelle d'un filtre passe bas.

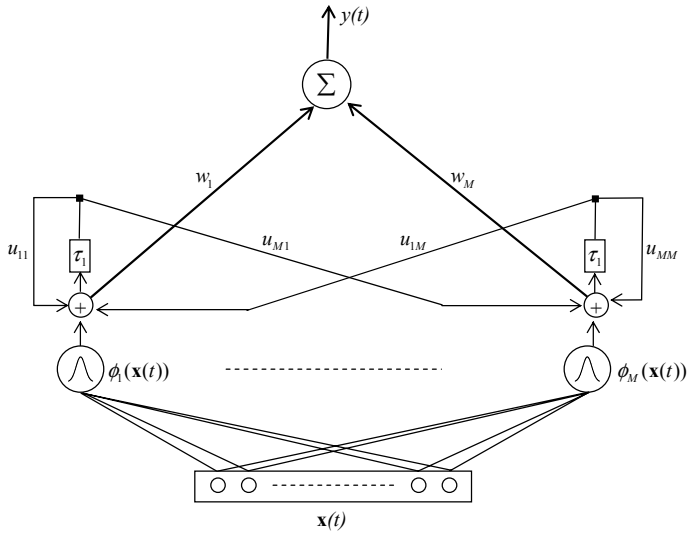


Figure 39. *L'architecture de M.W. Mak pour un RFR récurrent.*

Architecture de Miyoshi (Miyoshi et al., 1995)

Le réseau proposé par *Miyoshi* présenté sur la Figure 40 a été conçu pour l'identification et la prédiction des systèmes dynamiques chaotiques. Ce réseau récurrent se compose de plusieurs sous réseaux *RFR*. Chaque sous réseau *r* (*r = 1,....,N*) contient *K* cellules gaussiennes et un neurone de sortie de fonction d'activation linéaire. L'expression de sortie de ces sous réseaux est la suivante :

$$g_r = \sum_{i=1}^{i=K} \phi_i^r w_i^r \qquad [103]$$

où w_i^r est le poids de la connexion entre le neurone de sortie du sous réseau *r* et le $i^{ème}$ neurone gaussien ayant comme expression de sortie :

$$\phi_i^r = \prod_{j=1}^{N} \exp\left(-\frac{(x_j - \mu_{ji}^r)^2}{2(\sigma_{ji}^r)^2} \right) \qquad [104]$$

avec $\mathbf{x} = \left[x_j \right]_{j=1,...,N}$ représentant le vecteur d'entrée, $\boldsymbol{\mu}_i^r = \left[\mu_{ji}^r \right]_{j=1,...,N}$ et $\boldsymbol{\sigma}_i^r = \left[\sigma_{ji}^r \right]_{j=1,...,N}$ respectivement le prototype et le rayon d'influence du $i^{ème}$ neurone du sous réseau *r*. Le réseau de neurones se comporte alors comme un système différentiel :

$$\frac{d\mathbf{x}}{dt} = \mathbf{g} \quad \text{avec} \quad \mathbf{x}(t_0) = \mathbf{x}_0 \qquad [105]$$

La sortie de réseau $\mathbf{y} = [y_i]_{i=1,\ldots,M}$ est représentée par une partie du vecteur $\mathbf{g}$ comme le montre la Figure 40 (avec $M<N$). Cette architecture a été reprise par (Honda *et al.* 1998) qui a proposé un algorithme d'apprentissage.

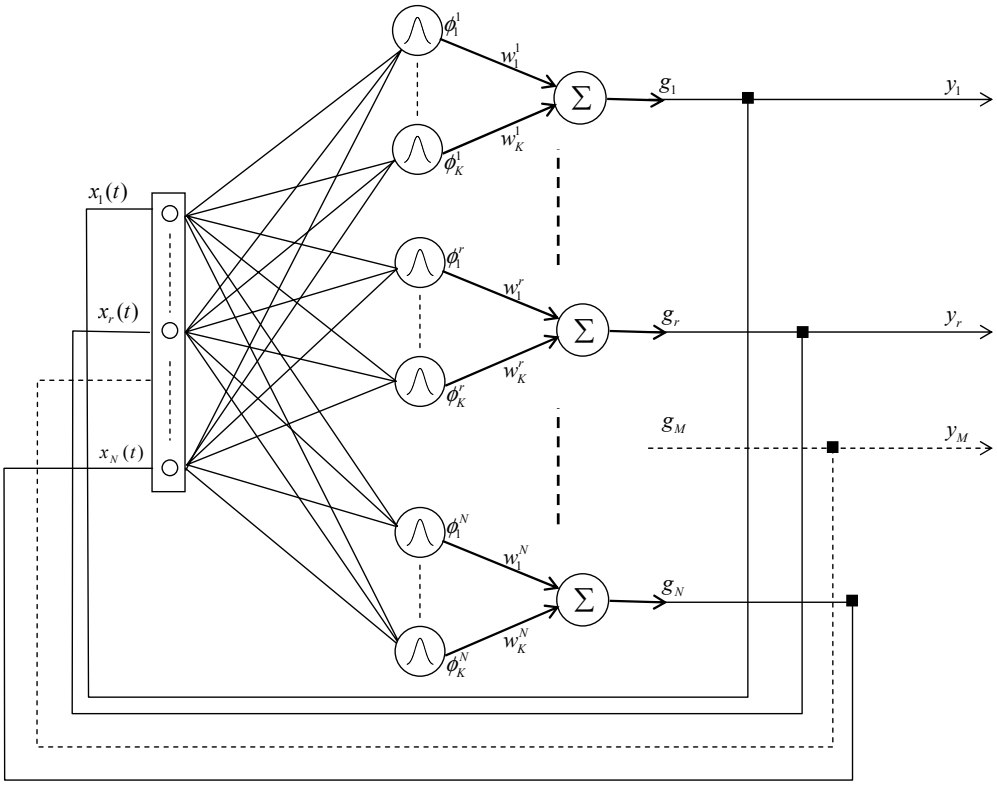


Figure 40. *L'architecture de Miyoshi pour un RFR récurrent.*

L'architecture R²BF (Frasconi et al., 1996)

Le réseau *RFR* Récurrent de la Figure 41 représente une architecture hybride entre neurones gaussiens (*représentation locale*) et neurones sigmoïdes (*représentation globale*). Une première couche cachée de neurones gaussiens est entièrement connectée à la deuxième couche cachée de neurones sigmoïdes. La sortie de ces neurones est connectée à un neurone de sortie, mais également réinjectée sur l'entrée des neurones gaussiens. Le vecteur d'entrée de chaque neurone gaussien est alors :

$$\mathbf{x}(t) = \left(s_1(t), s_2(t), \ldots, s_n(t), y_1(t-1), y_2(t-1), \ldots, y_m(t-1)\right) \qquad [106]$$

avec $s_i(t)|_{i=1,..,n}$ représentant les entrées du réseau à l'instant *t*, et $y_i(t-1)|_{i=1,..,m}$ les sorties des neurones sigmoïdes de la deuxième couche cachée à l'instant *t-1*.

Ce réseau de neurones a été appliqué sur un problème d'apprentissage de séquences d'une grammaire d'un automate à états finis. La même application avec une architecture légèrement différente d'un *RFR* récurrent peut être trouvée dans (Sorel *et al.,* 2000), et avec un *PMC* récurrent dans (Giles *et al.,* 1992).

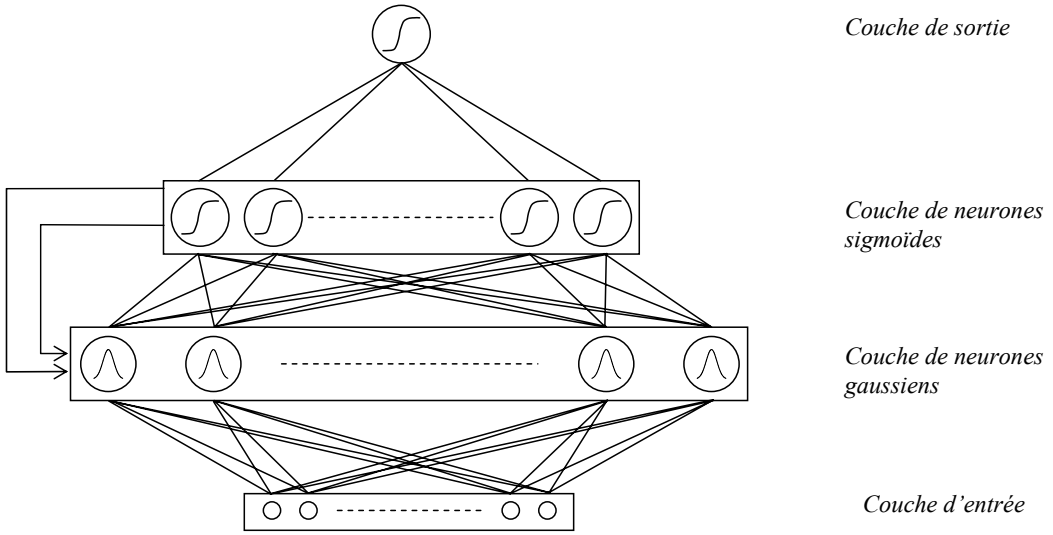


Figure 41. *L'architecture de Frasconi pour un RFR récurrent.*

b) Algorithmes d'apprentissage pour réseaux récurrents

Fixed Point Learning

Ce type d'algorithme d'apprentissage permet d'affecter aux poids des connexions les valeurs assurant la convergence vers un point d'équilibre. Néanmoins, la convergence de ce type d'algorithme souffre dans certains cas d'une dépendance des conditions initiales. La Figure 42 montre qu'un changement infinitésimal des conditions initiales ou de la pente d'un point intermédiaire sur la trajectoire, peut changer le point d'équilibre vers lequel le système évolue (Pearlmutter, 1990).

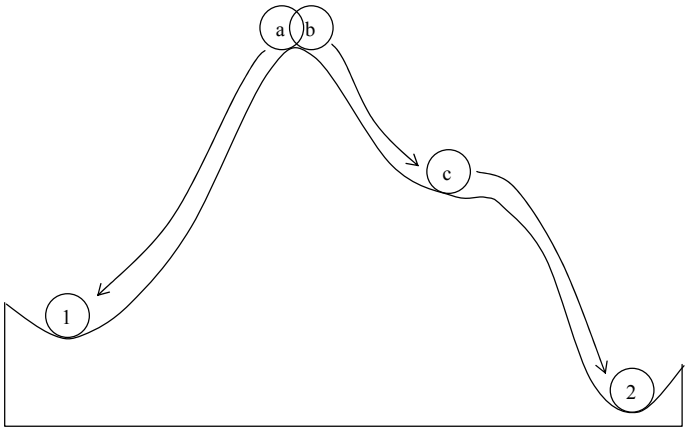


Figure 42. *Exemple de stabilisation vers deux points d'équilibre différents à partir de deux points quasiment semblables.*

- ***Rétropropagation récurrente***

Soit le réseau récurrent de la Figure 43 ayant *N* neurones. Parmi ces *N* neurones, certains peuvent être des neurones d'entrée avec comme valeur d'entrée $\xi_i^{\wp}$ pour la donnée $\wp$, et $\xi_i^{\wp} = 0$ pour les autres neurones. De même, certains peuvent être des neurones de sortie possédant pour sortie désirée (ou cible) $\zeta_i^{\wp}$. Dans la suite, l'indice $\wp$ sera volontairement omis et *f* est la fonction d'activation des *N* neurones ayant pour sortie v_i. L'algorithme de rétropropagation récurrente a été proposé par Pineda (Pineda, 1987) et Almeida (Almeida, 1988) qui ont remarqué que l'algorithme de rétropropagation du gradient est un cas particulier d'un gradient de l'erreur plus globale.

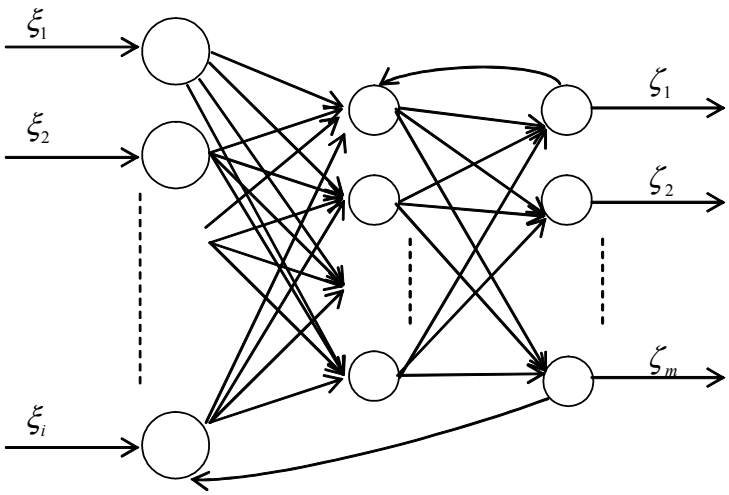


Figure 43. *Exemple de réseau récurrent.*

Si l'on reprend la loi d'évolution d'un neurone récurrent [97], on obtient :

$$\beta_i \frac{d\nu_i}{dt} = -\nu_i + f(\sum_j (w_{ij}\nu_j) + \xi_i) \qquad [107]$$

Cette définition conduit aux points d'équilibre où $d\nu_i / dt = 0$ donnés par :

$$\nu_i = f(a_i) = f(\sum_j (w_{ij}\nu_j) + \xi_i) \qquad [108]$$

La méthode conduit à supposer qu'au moins un tel point fixe existe et qu'il constitue un attracteur stable de l'espace d'état du réseau.

La mesure de l'erreur habituellement considérée est l'erreur des moindres carrés définie par :

$$E = \frac{1}{2}\sum_k E_k^2 \qquad [109]$$

avec

$$E_k = \begin{cases} \zeta_k - \nu_k & si\ k\ est\ un\ neurone\ de\ sortie \\ 0 & si\ ce\ n'est\ pas\ le\ cas \end{cases} \qquad [110]$$

La descente du gradient de cette erreur donne par conséquent :

$$\Delta w_{pq} = -\eta \frac{\partial E}{\partial w_{pq}} = \eta \sum_k E_k \frac{\partial \nu_k}{\partial w_{pq}} \qquad [111]$$

où pour calculer $\partial \nu_k / \partial w_{pq}$ il faut dériver l'équation [108], ce qui donne pour l'unité *i* :

$$\frac{\partial \nu_i}{\partial w_{pq}} = f'(a_i)\left[\delta_{ip}\nu_q + \sum_j w_{ij}\frac{\partial \nu_j}{\partial w_{pq}}\right] \qquad [112]$$

où a_i est l'activation du neurone *i* et δ_{ij} est le symbole de *Kronecker* :

$$\delta_{ij} = \begin{cases} 1 & si\ i = j \\ 0 & si\ non \end{cases} \qquad [113]$$

Les termes de cette égalité peuvent être réécrits sous la forme :

$$\sum_j L_{ij} \frac{\partial \nu_j}{\partial w_{pq}} = \delta_{ip} f'(a_i) \nu_q \qquad [114]$$

où

$$L_{ij} = \delta_{ij} - f'(a_i) w_{ij} \qquad [115]$$

on obtient alors, pour l'unité *k,* en inversant l'équation [114] :

$$\frac{\partial \nu_k}{\partial w_{pq}} = (L^{-1})_{kp} f'(a_p) \nu_q \qquad [116]$$

ce qui conduit selon [111] à :

$$\Delta w_{pq} = \eta \sum_k E_k (L^{-1})_{kp} f'(a_p) \nu_q \qquad [117]$$

la règle delta obtenue est alors :

$$\Delta w_{pq} = \eta \vartheta_p \nu_q \qquad [118]$$

avec

$$\vartheta_p = f'(a_p) \sum_k E_k (L^{-1})_{kp} \qquad [119]$$

L'équation [118] constitue donc la règle d'apprentissage du réseau récurrent. L'algorithme nécessite alors une inversion de matrice à chaque itération pour le calcul des quantités ϑ. En écrivant :

$$\vartheta_p = f'(a_p) Y_p \qquad [120]$$

avec

$$Y_p = \sum_k E_k (L^{-1})_{kp} \qquad [121]$$

En utilisant une nouvelle inversion de matrices, il est possible d'obtenir des équations linéaires en Y_p

$$\sum_p Y_p L_{pi} = E_i \qquad [122]$$

ce qui revient selon l'équation [115] à écrire :

$$Y_i = g\left(\sum_p (f'(a_p) w_{pi} Y_p) + E_i \right) \qquad [123]$$

On remarque que cette équation est de la même forme que l'équation du point d'équilibre [108] avec la fonction linéaire $g(x) = x$. Cette équation peut être résolue en considérant l'évolution d'un nouveau réseau, appelé réseau de propagation de l'erreur, qui a pour dynamique celle définie de façon analogue à [107] par l'équation :

$$\beta_i \frac{dY_i}{dt} = -Y_i + g(\sum_p (f'(a_p) w_{pi} Y_p) + E_i) \qquad [124]$$

La topologie du réseau de propagation de l'erreur est la même que celle du réseau d'origine mais dans laquelle le poids w_{ij} du neurone *j* au neurone *i* est remplacé par le terme $f'(a_i) w_{ij}$ de *i* à *j*. Par ailleurs, le terme d'entrée ξ_i devient E_i, erreur du neurone *i* dans le réseau d'origine.

Les étapes de l'algorithme sont :

- laisser le réseau évoluer suivant l'équation [107] pour obtenir les quantités v_i,
- effectuer les comparaisons pour déterminer les quantités E_i d'après [110],
- laisser le réseau de propagation de l'erreur évoluer suivant l'équation [124] pour obtenir les quantités Y_i,
- mettre à jour les poids suivant les équations [118] et [120].

Trajectory Learning

L'algorithme « *fixed point learning* » présenté précédemment est incapable de reproduire des séquences ou des trajectoires temporelles. On dira que ce sont des modèles qui *utilisent* le temps (Chappelier, 1996). En d'autres termes, la convergence du réseau vers un état stable suppose une certaine temporalité. Par contre, dans la deuxième catégorie d'algorithmes d'apprentissage, le temps est *traité* par le réseau de neurones. Le paramètre temps représente une donnée à *traiter* par le processus d'apprentissage. Le but est donc d'apprendre au réseau de neurones à reproduire une séquence temporelle grâce à sa mémoire dynamique. On peut recenser cinq algorithmes d'apprentissage pour réseaux récurrents dit *Trajectory Learning* (Atiya *et al.,* 2000), (Aussem, 2002) : la Rétropropagation dans le temps ou *Back-*

Propagation Through Time : BPTT (Werbos, 1990), (Rumelhart *et al.,* 1986) ; la Propagation Avant ou *Forward Propagation* appelé aussi *Real Time Recurrent Learning : RTRL* (Williams *et al.,* 1989) ; la Propagation Avant Rapide ou *Fast Forward Propagation : FFP* (Toomarian *et al.,* 1991) ; l'approche par Fonction de *Green* ou *Green Function : GF* (Sun *et al.,* 1992) et enfin l'approche par *Block Update : BU* (Schmidhuber, 1992). D'autres algorithmes d'apprentissage ont été proposés ces dernières années parmi lesquels : Temporal Recurrent Back-Propagation qui se trouve à la croisée du *BPTT* et l'algorithme de rétropropagation récurrente par (Aussem, 1995) ; *Recursive Back-Propagation* (*RBP*) et sa version temps réel *Causal Recursive Back-Propagation* (*CRBP*) par (Campolucci *et al.,* 1999), une autre technique qui se base sur une approximation du gradient de l'erreur proposée par (Atiya *et al.,* 2000) et, enfin, une technique d'apprentissage appelée Statistical Approximation Learning (*SAL*) appliquée à une architecture de réseau récurrent bien particulière appelée Simultaneous Recurrent Networks (*SRN*) proposée par (Sakai *et al.,* 2002). Nous présentons les deux algorithmes d'apprentissage les plus utilisés pour les réseaux de neurones récurrents : *BPTT* et *RTRL.*

- ***Rétropropagation dans le temps « Back-Propagation Through Time : BPTT »***

Soit un réseau de neurones récurrent entièrement connecté où chaque unité est connectée à n'importe quelle autre unité. Supposons que l'évolution du réseau soit menée de façon synchrone en temps discret et donc que chaque neurone du réseau ait pour équation de mise à jour :

$$\nu_i(t+1) = f\left(a_i(t)\right) = f\left(\sum_j (w_{ij}\nu_j(t)) + \xi_i(t)\right) \qquad [125]$$

dans laquelle $\xi_i(t)$ est l'entrée, si elle existe, du neurone *i* à l'instant *t*.

Supposons en outre que le comportement que l'on souhaite obtenir du réseau ne soit plus de posséder un certain nombre de points d'équilibre mais plutôt de décrire un ensemble d'états donnés. En d'autres termes, en présence des entrées $\xi_i(t)$ pour chaque neurone et à chaque instant, on souhaite obtenir une réponse $\zeta_i(t)$ pour certains neurones et à certains instants. Si l'horizon temporel sur lequel le réseau doit réaliser cette tâche est borné par une longueur maximale *T*, il est possible d'utiliser un artifice pour transformer un réseau récurrent en un réseau feedforward (à propagation avant). C'est cette idée utilisée avec l'algorithme de la rétropropagation qui a donné lieu à l'algorithme de la rétropropagation dans le temps. Cette transformation consiste à dupliquer les neurones sur l'horizon temporel *t=1,2,…T* de façon à ce qu'une unité ν_i^t représente l'état $\nu_i(t)$ du réseau récurrent équivalent. La Figure 44 illustre cette transformation pour un réseau récurrent comportant 2 unités et sur un horizon temporel de longueur *T=4*.

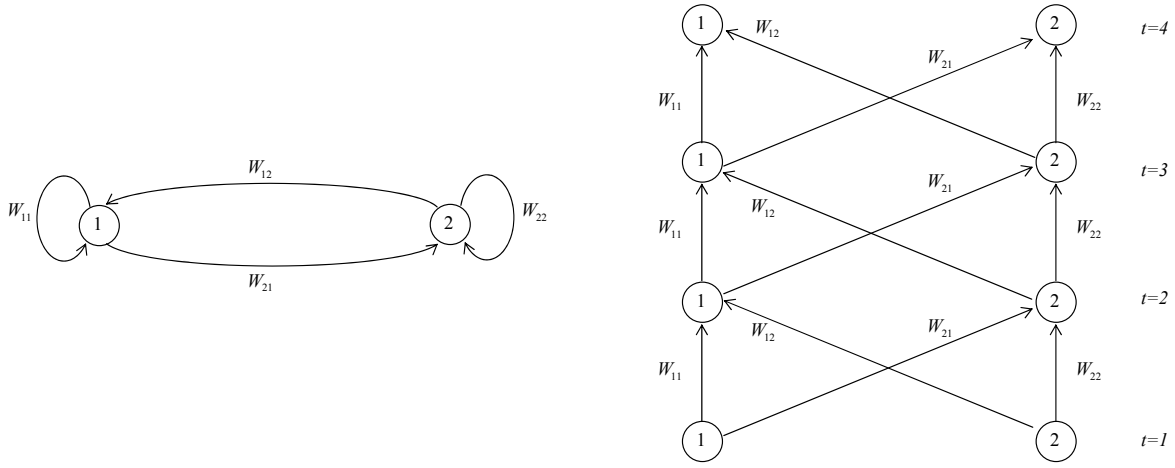


Figure 44. Un réseau récurrent et sa version dépliée.

Le réseau ainsi déplié est de type *feedforward* et peut faire l'objet d'un apprentissage par une version légèrement modifiée de l'algorithme de rétropropagation. Chaque unité calcule ainsi la somme pondérée de ses entrées où, lorsqu'elle existe, l'entrée ou la cible de l'unité i à l'instant t est appliquée à l'unité ν_i^t. Dans le cas des unités de sortie, l'erreur doit être appliquée non seulement sur la dernière couche mais également sur toutes les autres et doit être propagée des couches où elle est apparue vers les couches inférieures. Le calcul de cette erreur entre les bornes $[t_0, t_1]$ est le suivant :

$$E(t_0, t_1) = \frac{1}{2} \sum_{t=t_0}^{t_1} \sum_{i=1}^{N} E_i(t)^2 \qquad [126]$$

avec

$$E_i(t) = \begin{cases} \zeta_i(t) - \nu_i(t) & \textit{si } i \textit{ est un neurone de sortie} \\ 0 & \textit{si ce n'est pas le cas} \end{cases} \qquad [127]$$

La difficulté de l'algorithme vient du fait que toutes les copies des poids w_{ij} doivent rester identiques alors que l'application de l'algorithme de rétropropagation entraînerait un incrément de Δw_{ij} différent pour chaque copie. La solution consiste à ajouter tous les incréments et à modifier les valeurs des copies des poids de la quantité obtenue.

$$\Delta w_{pq} = -\eta \frac{\partial E(t_0, t_1)}{\partial w_{pq}} = \eta \sum_{t=t_0}^{t_1} \sum_{i=1}^{N} E_i(t) \frac{\partial \nu_i(t)}{\partial w_{pq}} \qquad [128]$$

Le principal problème que pose l'algorithme de rétropropagation du gradient dans le temps est qu'il nécessite beaucoup de ressources informatiques. Pour de longues séquences ou des séquences de longueur inconnue, il devient inutilisable. La duplication ne s'applique que pendant la phase d'apprentissage. Plus de détails sur ce type d'algorithme sont présentés dans (Pearlmutter, 1990), (Werbos, 1990), (Warwick *et al.,* 1992).

- ***Le RTRL (Real Time Recurrent Learning)***[20]

L'algorithme de rétropropagation dans le temps représente une technique d'apprentissage hors-ligne[21] : on attend que la totalité des données de l'horizon temporel soient présentées au réseau pour calculer l'expression [128]. Williams et Zipser (Williams *et al.,* 1989) ont proposé un algorithme pour l'apprentissage des réseaux récurrents sans avoir à connaître la longueur de l'horizon temporel de la séquence d'apprentissage. La mise à jour des poids s'effectue au fur et à mesure que les données sont présentées au réseau par une quantité $\Delta w_{pq}(t)$ ([129]) et, ceci, sans avoir à dupliquer le réseau récurrent (Robinson *et al.,* 1987). Leur principal désavantage est que le réseau de neurones doit être entièrement connecté et souffre d'une certaine lenteur du temps d'apprentissage.

$$\Delta w_{pq}(t) = -\eta \frac{\partial E(t)}{\partial w_{pq}} = \eta \sum_{i=1}^{N} E_i(t) \frac{\partial \nu_i(t)}{\partial w_{pq}} \qquad [129]$$

avec $E_i(t)$ l'erreur commise à chaque instant *t* par les neurones de sortie (équation [127]). D'après l'équation [125], on obtient :

$$\frac{\partial \nu_i(t)}{\partial w_{pq}} = f'\left(a_i(t-1)\right)\left[\delta_{ip}\nu_q(t-1) + \sum_j w_{ij} \frac{\partial \nu_j(t-1)}{\partial w_{pq}}\right] \qquad [130]$$

où δ_{ij} est le symbole de *Kronecker* (voir [113]). Notons que cette démarche est analogue à celle utilisée habituellement sur les réseaux à propagation avant, qui consiste à appliquer les modifications aux poids après chaque exemple au lieu d'attendre la fin du cycle complet de présentation des données. Par ailleurs, cette technique ne garantit pas le suivi du gradient total de l'erreur de toute une séquence d'apprentissage (Tsoi *et al.,* 1994). En effet, la trajectoire suivie par le réseau dans l'espace d'état dépend des modifications apportées aux poids à chaque instant. Cet effet peut être éliminé soit en prenant un taux d'apprentissage assez faible pour diminuer les variations $\Delta w_{pq}(t)$ de l'équation [129] (Williams *et al.,* 1989), soit des techniques d'apprentissage du second ordre (Le Cun *et al.,* 1990), (Hassibi *et al.,* 1993) et (Svarer *et al.,* 1993). Il existe aussi une variante du *RTRL* appelée *Teacher-Forced Real-Time Recurrent Learning* (Jordan, 1986-b-), (Pineda, 1988) qui force la sortie du réseau aux valeurs

[20] On peut trouver en littérature l'appellation « *Forward Propagation : FP*»
[21] Off-line technique

désirées. La mise à jours des poids des connexions ([129]) s'effectue uniquement sur l'ensemble des neurones qui ne sont pas des neurones de sortie.

III.2.2.2. Représentation explicite du temps dans les réseaux de neurones

a) Représentation explicite du temps au niveau des connexions : les connexions à délais

Une des représentations explicites du temps dans les réseaux de neurones est l'utilisation des retards aux niveaux des connexions. La différence entre ce type de réseau de neurones et les réseaux récurrents se situe au niveau sens de la propagation du signal. Les réseaux à délais au niveau des connexions sont des réseaux *feedforward*. La Figure 45 montre un exemple d'un réseau de neurones avec des délais internes au niveau des connexions. Ce modèle introduit par Day et Davenport (Day *et al.,* 1993) est appelé *Adaptative Time-Delay Neural Network* : *ATDNN*.

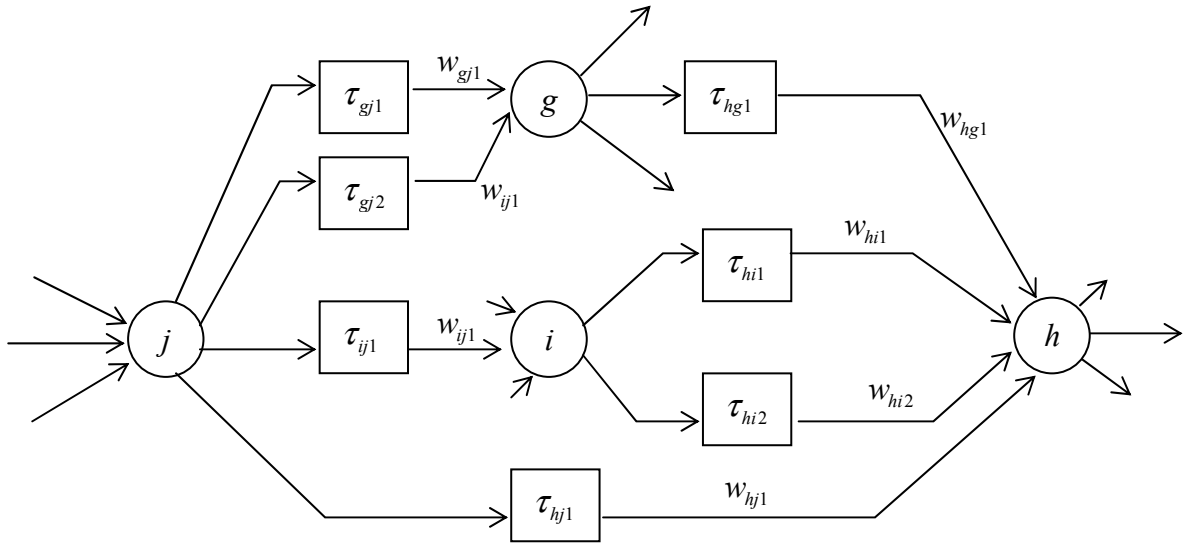


Figure 45. Exemple d'un réseau avec des connexions à délais.

L'évolution du réseau est dictée par la loi suivante :

$$a_i(t) = \sum_j \sum_k w_{ijk}(t) v_j\left(t - \tau_{ijk}(t)\right) \qquad [131]$$

où $a_i(t)$ représente l'activation du neurone i à l'instant t, w_{ijk} représente le poids de la $k^{ème}$ connexion entre les neurones i et j avec un retard de τ_{ijk}, $v_j(t)$ est la sortie du $j^{ème}$ neurone à l'instant t. L'algorithme d'apprentissage doit permettre non seulement une adaptation des poids des connexions mais également une adaptation des délais. Toute la difficulté que pose ce type de modèle à délais est la coordination entre la propagation du signal et la

rétropropagation du gradient de l'erreur. Parmi les réseaux à retard que l'on peut trouver en littérature, on peut citer le modèle de Béroule (Béroule, 1985), le modèle d'Amit (Amit, 1988), le modèle de Back-Tsoi (Back *et al.,* 1990), le modèle de DeVries-Principe (DeVries *et al.,* 1991) et le réseau de Jacquemin (Jacquemin, 1994).

b) Représentation explicite du temps au niveau des neurones

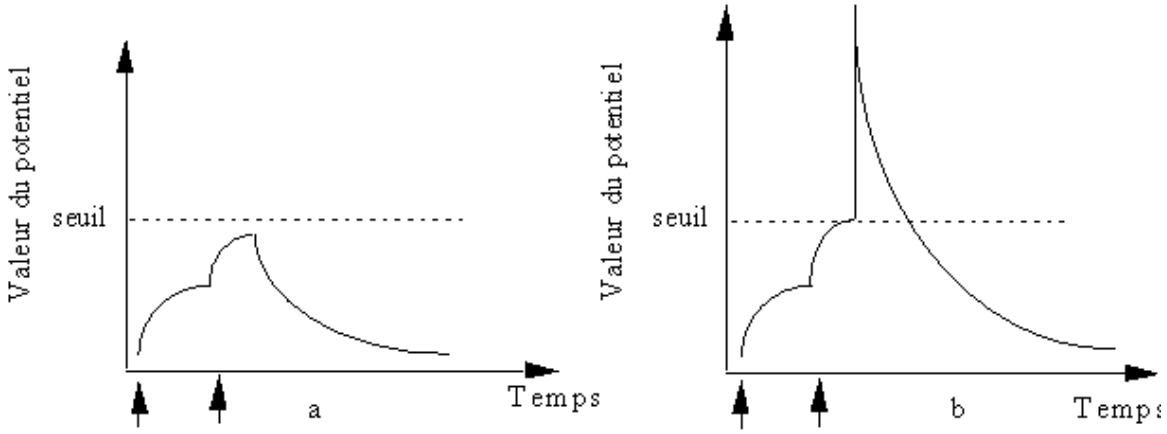


Figure 46. Sommation spatio-temporelle : addition des potentiels évoqués à la fois dans l'espace et dans le temps. a) 2 potentiels évoqués (repérés par les flèches) ne dépassent pas la valeur seuil. b) 2 potentiels évoqués qui dépassant la valeur seuil génèrent un potentiel d'action.

Le temps peut également être traité *localement* au niveau du neurone, ce qui permet d'avoir une certaine robustesse temporelle : deux entrées d'un neurone donné ne doivent plus être nécessairement synchrones pour avoir un certain effet (Chapellier, 1996). Ce modèle de neurone temporel peut être réalisé de deux manières différentes : soit en gardant les propriétés biologiques d'un neurone soit en construisant un modèle pour résoudre un problème « d'ingénierie » sans autre type de préoccupations (ignorant complètement l'aspect biologique). Le premier cas conduit à des modèles plus ou moins compliqués. Il existe de nombreux modèles d'inspiration biologique. La plus grande classe est ainsi formée par ce que l'on nomme les modèles « *Integrate and Fire* » (Koch, 1999), (Liu *et al.,* 2001). Leur principe est de sommer spatialement et temporellement « *Integrate* » les entrées leur provenant. Lorsque cette somme dépasse un seuil qui leur est propre, ils émettent « *Fire* » une impulsion (potentiel d'action) (Figure 46). Ces modèles possèdent ainsi des propriétés temporelles inspirées des neurones biologiques tout en permettant un calcul rapide. Le second point de vue consiste à temporaliser les modèles existants performants du point de vue statique. On peut citer par exemple l'approche de Vaucher (Vaucher, 1993) qui a temporalisé un modèle

statique d'une façon purement algébrique. Le corps de représentation d'un *PMC* n'est plus dans le domaine des réels ($\mathbb{R}$) mais dans celui des complexes ($\mathbb{C}$).

III.2.3. Analyse comparative entre représentation dynamique et spatiale pour la surveillance dynamique

Les réseaux de neurones temporels se divisent donc en deux grandes catégories : réseaux de neurones *dynamiques* et *spatiaux*. Ces deux représentations du temps correspondent respectivement à une représentation *interne* et *externe* du temps. Les topologies des réseaux de neurones pour chaque représentation temporelle sont complètement différentes et chacune possède ses avantages et ses inconvénients.

Nous définissons trois types d'applications des réseaux de neurones temporels pour la surveillance dynamique industrielle : la reconnaissance de séquences temporelles (booléennes et réelles), la prédiction temporelle et la reproduction de séquences temporelles.

- *La reconnaissance de séquences temporelles :*

La reconnaissance de séquences temporelles consiste à produire une réponse spécifique lorsqu'une séquence particulière se produit à l'entrée du réseau. En d'autres termes, le rôle du réseau de neurones dans ce cas est de reconnaître une séquence temporelle bien particulière (application de reconnaissance des formes). C'est le cas par exemple de la reconnaissance de la parole où la sortie du réseau indique le mot qui vient d'être émis.

En surveillance, ce type d'application est utilisé selon deux manières différentes : la reconnaissance d'une séquence booléenne et la reconnaissance d'une séquence réelle. Pour le cas d'une séquence booléenne, le réseau de neurones est utilisé pour surveiller tout un Système à Evénements Discrets (SED). Les variables d'entrée au réseau de neurones sont de type booléen (événementiel). Dans le deuxième cas, le réseau de neurones surveille un signal capteur d'un équipement industriel (variable de surveillance de type réel). Le but est de reconnaître un palier d'une dégradation précoce de l'équipement et d'éliminer les pics de changements brusques du signal, synonymes de fausses alarmes.

- *La prédiction temporelle :*

La prédiction temporelle consiste à donner une valeur future d'un signal capteur d'un équipement industriel ($s(t+n)$ avec $n>0$) à partir des connaissances aux instants présents et passés de ce signal ($s(t), s(t-1), \ldots$). En surveillance dynamique, ce type d'application est très important car prédire l'évolution d'un paramètre d'un équipement permet d'anticiper l'évolution future du signal d'un capteur afin de prendre des décisions préventives. Le réseau de neurones est dans ce cas utilisé comme approximateur universel (modélisation dynamique de l'équipement à surveiller).

- *La reproduction de séquences temporelles :*

La reproduction de séquences temporelles consiste à reproduire toute une séquence temporelle par le réseau de neurones. Ce type d'application est aussi très important en surveillance dynamique. Le réseau de neurones peut être utilisé pour reproduire un régime transitoire d'une évolution temporelle d'un paramètre d'un équipement industriel. La comparaison entre la sortie réelle et celle du réseau de neurones permet d'avoir un résidu pour la détection et le diagnostic de cet équipement. Souvent, le régime transitoire d'un équipement, obtenu soit au démarrage de l'équipement soit entre changement de modes de fonctionnement, est très riche en informations utiles au diagnostic préventif.

Nous avons récapitulé sur le tableau ci-dessous l'analyse des performances des architectures de réseaux de neurones temporels présentés dans ce chapitre pour chacun des trois types d'application cités précédemment.

	Représentation spatiale	**Représentation dynamique**			
		Le temps est explicite			Le temps est implicite *-Réseaux récurrents-*
		Le temps au niveau de connexions -Réseaux à retards-	Le temps au niveau du neurone		
			Modèle fonction du temps	Modèle biologique	
Reconnaissance de séquences	oui	oui	oui	oui	oui
Prédiction temporelle	oui	oui	oui	oui	oui
Reproduction de séquences	non	non	non	non	oui

Tableau 1. *Résultats comparatifs entre les performances des architectures temporelles citées dans ce chapitre (voir Figure 28).*

Les réseaux spatiaux (*TDNN* et *TDRBF*) ainsi que les réseaux dynamiques à délais (*ATDNN*) se prêtent bien aux problèmes de reconnaissance de séquences. Les fenêtres temporelles dans le premier cas et les délais au niveau des connexions dans le deuxième cas permettent au réseau de neurones de prendre en compte un certain passé du signal afin de pouvoir donner une réponse par rapport à toute une séquence. Les réseaux biologiques, en particulier les réseaux *Integrate and Fire,* sont également capables de reproduire ce type de comportement. L'utilisation des réseaux spatiaux peut être plus souple que les deux autres architectures puisqu'un simple ajout d'une fenêtre temporelle peut rendre les architectures *feedforward statiques* capables de traiter le temps. L'inconvénient majeur de la représentation spatiale du temps est qu'elle suppose l'existence d'une interface avec le monde extérieur dont le rôle est de retarder ou de retenir les données jusqu'au moment de leur utilisation par le

réseau : *comment connaître l'instant où les données doivent être traitées ?* La longueur de la fenêtre temporelle est finie et déterminée a priori, soit par la plus longue information à traiter, soit en supposant la même longueur pour toutes les données. C'est donc bien dans la nature même de la représentation spatiale que se pose la difficulté de différencier une position temporelle relative d'une position temporelle absolue (Elman, 1990).

Les architectures à représentation spatiale du temps peuvent également être utilisées pour des applications de prédiction de séries temporelles (prédire la valeur de $x(t+\theta)$ à partir des connaissances des valeurs $\left[x(t-i)\right]_{i=0,\ldots,\alpha}$) (Chang *et al.,* 2001).

Par contre, aucune des architectures neuronales temporelles, excepté les réseaux récurrents, n'est capable de reproduire des séquences temporelles. La raison est que seuls les réseaux récurrents possèdent des mémoires dynamiques grâce à la récurrence des connexions (Aussem, 1995). Le signal d'entrée ne se propage pas seulement de la couche d'entrée vers la couche de sortie comme dans les réseaux *feedforward,* mais se rétropropage également de la sortie vers l'entrée. Cette boucle fermée permet au réseau de garder en mémoire une trace interne d'un signal d'entrée, par exemple une impulsion, et de reproduire en sortie une séquence temporelle grâce aux algorithmes d'apprentissage de type *trajectory learning*. D'un autre côté, les temps d'apprentissage ainsi que les ressources informatiques nécessaires à leur mise en œuvre peuvent être relativement importants (Bernauer, 1996).

Le domaine d'application des réseaux récurrents semble être plus large que les autres architectures. Leur application dans des problématiques de surveillance dynamique peut être plus prometteuse que les autres architectures.

III.3. Conclusion

Nous avons abordé dans ce troisième chapitre un aspect fort important en surveillance : la prise en compte de la *dimension temporelle*. Nous avons donné un état de l'art aussi large que possible des différentes façons de prendre en compte cet aspect temporel par les réseaux de neurones, les différentes architectures de réseaux de neurones temporels et la façon dont est mené l'apprentissage temporel.

Cette étude nous a permis de conclure que les façons d'aborder le temps par les réseaux de neurones sont nombreuses de même que les travaux et publications concernant les applications et les architectures de réseaux de neurones temporels. Néanmoins, à travers la liste de références non exhaustives mais représentatives que nous avons consulté, nous remarquons deux représentations temporelles des réseaux de neurones : une représentation *spatiale* ou *externe* et une représentation *dynamique* ou *interne*. Nous nous sommes principalement attachés à exposer les modèles à représentation interne du temps, obtenue par l'utilisation de connexions récurrentes. Les raisons évoquées pour ce choix sont :

- d'un côté la prise en compte du temps est implicite, c'est-à-dire qu'on n'a pas besoin d'avoir un mécanisme externe pour retarder les données d'entrée (comme pour la représentation spatiale du temps),
- d'un autre côté, les réseaux récurrents sont bien les seuls réseaux à posséder une mémoire dynamique interne à travers la récurrence des connexions. Cette mémoire leur permet, non seulement de reconnaître des séquences temporelles et de faire de la prédiction de séries temporelles (comme les autres représentations temporelles) mais aussi d'apprendre à reproduire des séquences temporelles.

En contre partie, la présentation des principaux algorithmes d'apprentissage des réseaux récurrents montre que la phase de calcul des poids des connexions est très laborieuse. Les temps d'apprentissage et les ressources informatiques nécessaires à leur mise en œuvre sont relativement importants. Leur application en surveillance dynamique peut être très complexe, surtout pour des traitements en temps réel. Le développement d'un outil de surveillance neuronal qui soit à la fois dynamique et facile à paramétrer à distance via la couche de communication TCP/IP se montre souvent plus délicat à mettre en œuvre avec un réseau entièrement récurrent (globalement récurrent).

Ce coût est parfois considérablement réduit lorsque le problème est résolu à l'aide de réseaux partiellement récurrents. Une autre façon encore plus simple de reproduire une mémoire dynamique interne sans trop compliquer l'architecture neuronale est d'utiliser des réseaux localement récurrents appelés réseaux *LRGF* : *Locally Recurrent Globally Feedforward* ou architecture *Localement Récurrente Globalement Feedforward*. Dans ce type d'architecture, la récurrence des connexions est présente uniquement au sein du neurone lui-même. Ceci réduit considérablement le processus d'apprentissage tout en gardant un aspect dynamique fort important du réseau de neurones récurrent.

Afin de pouvoir bénéficier de tous ces avantages, nous exploitons cette solution de récurrences locales pour proposer une nouvelle architecture de réseau *RFR* dynamique plus souple avec un algorithme d'apprentissage simplifié. Dans ce sens, le chapitre suivant marque une première étape de notre contribution en présentant le *RRFR (Réseau Récurrent à Fonctions de base Radiales)* et, plus particulièrement, en effectuant une étude approfondie de sa mémoire dynamique interne. Nous donnons ainsi les différentes architectures des réseaux localement récurrents (architectures *LRGF)*, avec des développements mathématiques associés aux différents comportements dynamiques de chaque architecture.

Chapitre IV

Proposition d'un réseau de neurones dynamique : Le RRFR.

Résumé : *La façon la plus simple de prendre en compte l'aspect temporel d'une manière implicite est de représenter le temps par des récurrences locales au niveau du neurone sans trop compliquer l'architecture du réseau. Les réseaux à représentation locale du temps sont appelés architectures LRGF : Locally Recurrent Globally Feedforward ou architectures Localement Récurrente Globalement Feedforward. Deux types d'architectures LRGF existent en littérature : une architecture à retour local de la sortie du neurone et une architecture à retour local de l'activation du neurone. Quels sont les différents modèles existants pour chaque architecture ? Quels sont les comportements dynamiques de chacune d'elles ? Quelle architecture serait la plus adaptée à notre problématique de surveillance dynamique ? Nous essaierons de répondre à ces questions à travers ce quatrième chapitre.*

Abstract : *The simplest way to take into account the temporal aspect by an implicit way is to represent the time by local feedbacks at the neuron level. We call this class of architectures Locally Recurrent Globally Feedforward (LRGF). There are two methods in witch feedback may be incorporated: local output feedback architecture, and local activation feedback architecture. What are the different existing models for each architecture? What is the dynamic behavior for each of them? What architecture would be the most adapted to our dynamic monitoring task? We will try to answer to these questions through this fourth chapter.*

Chapitre IV

Proposition d'un réseau de neurones dynamique : Le RRFR

IV.1. Introduction

Le chapitre précédent montre que parmi l'ensemble des représentations temporelles, les réseaux de neurones récurrents sont les plus performants pour des applications de surveillance dynamique. Les réseaux de neurones récurrents se montrent favorables aux trois types d'applications des réseaux de neurones temporels en surveillance dynamique, à savoir la reconnaissance de séquences temporelles, la prédiction temporelle et la reproduction de séquences temporelles. Les réseaux récurrents sont donc les seuls à posséder une mémoire dynamique interne. Les techniques d'apprentissage de ces réseaux sont souvent très lourdes à mettre en œuvre. Une application de surveillance dynamique avec ce type de réseaux de neurones peut être très compliquée à cause de cette phase d'apprentissage très complexe et surtout coûteuse en temps de calcul. Le développement d'un outil de surveillance dynamique paramétrable à distance via le Web peut être très difficile à développer avec les réseaux de neurones globalement récurrents. Pour éviter cette complexité du processus d'apprentissage, une façon simple d'avoir une mémoire dynamique interne au réseau de neurones est d'utiliser des récurrences locales au niveau du neurone lui-même. Ce type bien particulier de réseaux de neurones récurrents est appelé représentation *Localement Récurrente Globalement Feedforward* ou *Locally Recurrent Globally Feedforward*.

L'utilisation des récurrences locales au niveau du neurone permet d'avoir une mémoire dynamique interne au réseau de neurones. Nous adoptons ce type de mémoire dynamique pour proposer un réseau *RFR* dynamique appelé *RRFR* : *Réseau Récurrent à Base de Fonctions Radiales*. Le réseau que nous proposons possède alors deux types de mémoire : une mémoire ***statique*** grâce aux neurones gaussiens de la couche cachée et une mémoire ***dynamique*** grâce aux neurones localement récurrents de la couche d'entrée. Le réseau *RRFR* proposé profite de l'aspect dynamique des représentations *LRGF* tout en gardant la simplicité et l'efficacité des réseaux *RFR*.

Ce quatrième chapitre est structuré en trois parties. Dans la première partie, nous présentons l'architecture générale du réseau *RRFR*. En deuxième partie, nous présentons une

étude approfondie des différentes mémoires dynamiques du réseau de neurones *RRFR* obtenues par des récurrences locales (architectures *LRGF*). Nous appuyons cette étude par des simulations de comportement dynamique de chaque type de mémoire. Nous concluons par une analyse comparative des différentes mémoires dynamiques possibles du réseau *RRFR* proposé.

IV.2. Architecture du réseau RRFR

Pour aller dans le sens du principe de la parcimonie[22] des réseaux de neurones, on peut se poser la question suivante : *Pourquoi compliquer l'architecture d'un réseau de neurones avec des récurrences globales quand on peut simplifier le réseau en utilisant des récurrences locales* ? Cette question résume en quelque sorte notre souci pour proposer un outil neuronal pour la surveillance dynamique. Cet outil de surveillance devrait être à la fois capable de prendre en compte l'aspect dynamique des données d'entrée, afin de pouvoir détecter une dégradation ou prédire l'évolution d'une sortie capteur, d'apprendre en continu les différents modes de fonctionnement d'un équipement sans avoir à oublier les connaissances précédemment acquises et surtout de garder une certaine simplicité d'utilisation du réseau de neurones pour des applications industrielles. L'architecture neuronale que nous proposons s'inspire des avantages des réseaux *RFR* et de ceux des réseaux récurrents. La Figure 47 présente l'architecture du *RFR* récurrent que nous proposons, appelé réseau *RRFR* : *Réseau Récurrent à Base de Fonctions Radiales*. Le réseau *RRFR* est composé de trois couches :

Une mémoire dynamique

La couche ℓ_1 appelée ***mémoire dynamique*** du réseau, a comme rôle principal de prendre en compte la dynamique des données d'entrée (Figure 47). En d'autres termes, cette couche sert à mémoriser le « *passé* » du signal d'entrée du réseau. Cette mémoire dynamique est constituée par des récurrences locales au niveau des neurones d'entrée sans trop compliquer la topologie du réseau *RFR*. Cela revient, en quelque sorte, à « *greffer* » une mémoire dynamique aux réseaux *RFR* statiques. C'est pour cette raison que nous nous sommes intéressés aux réseaux *Localement Récurrents* connus sous l'appellation *LRGF* « *Locally Recurrent Globally Feedforward ».* Nous présentons une étude sur les architectures *LRGF* existantes en littérature avec des tests de performance pour chaque modèle afin de choisir celle qui serait la plus adaptée à la surveillance dynamique.

[22] *Un réseau de neurones est dit parcimonieux : pour obtenir un modèle non linéaire de précision donnée, un réseau de neurones a besoin de moins de paramètres ajustables que les méthodes de régression classique (polynomiale). Le principe de la parcimonie est donc de réduire le nombre de paramètres ajustables du réseau de neurones.*

Une mémoire statique

La couche ℓ_2, appelée ***mémoire statique*** du réseau, a pour rôle de mémoriser les prototypes, comme pour les réseaux *RFR* classiques. Les entrées de cette couche ne proviennent plus directement des données d'entrée mais résultent de la sortie de la première couche, à savoir la mémoire dynamique. Un premier traitement des données d'entrée est donc effectué par la première couche avant que celles-ci soient mémorisées par la mémoire statique. Le prototype ainsi mémorisé par les neurones gaussiens aura implicitement pris en compte la dimension temporelle des données d'entrée. Nous détaillerons davantage les effets de la mémoire dynamique sur la mémoire statique au chapitre suivant.

Une couche de décision

La couche de sortie ℓ_3 qui donne la décision de la classification (dans le cas d'une application en reconnaissance de formes), ou la valeur de la fonction à prédire (dans le cas d'une approximation de fonction).

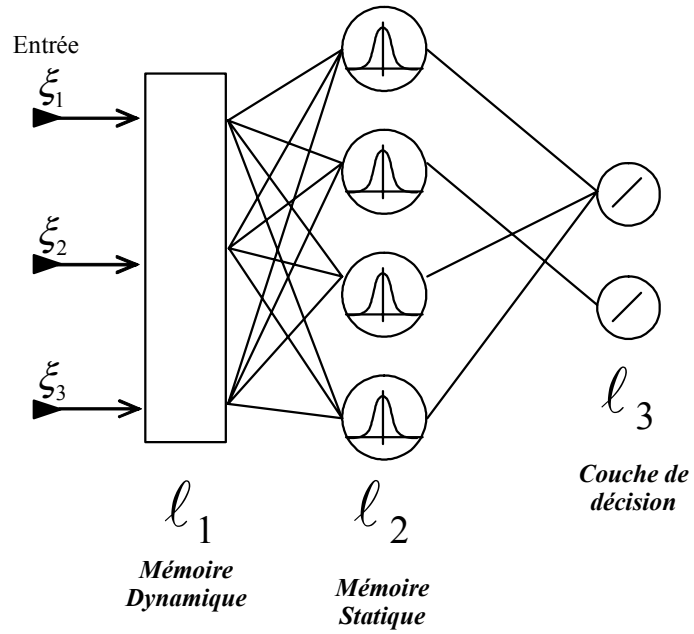


Figure 47. *Réseau RRFR (Réseaux Récurrents à Fonctions de base Radiales).*

IV.3. Mémoire dynamique du réseau RRFR : Architecture LRGF

La façon la plus simple de représenter le temps d'une manière implicite par la récurrence des connexions est de le représenter localement au niveau du neurone sans trop compliquer l'architecture du réseau de neurones. Cette représentation locale du temps est appelée architecture *LRGF* : *Locally Recurrent Globally Feedforward* ou architecture *Localement Récurrente Globalement Feedforward*. Cette appellation reflète clairement le principe de fonctionnement de ces réseaux : les récurrences des connexions ne sont permises que localement au niveau du neurone (*Locally Recurrent*), la propagation du signal s'effectue comme pour les réseaux statiques à propagation avant (*Globally Feedforward*). Une étude a été réalisée par A.C. Tsoi et A.D. Back (Tsoi *et al.,* 1994) sur les différentes architectures *LRGF* existantes en littérature. D'après cette étude, les auteurs ont classé les architectures *LRGF* en trois catégories :

- Les architectures à *retour local synaptique (Local Synapse Feedback)*,
- Les architectures à *retour local de l'activation (Local Activation Feedback)*,
- Les architectures à *retour local de la sortie* du neurone *(Local Output Feedback)*.

Les architectures à retour local de l'activation et de la sortie interprètent bien la dimension temporelle implicitement par une récurrence des connexions. Par contre, concernant l'architecture à retour local synaptique, le temps n'est pas pris en compte par la récurrence des connexions, mais tout simplement par des retards synaptiques. Nous plaçons donc cette catégorie plutôt dans la représentation explicite du temps au niveau des connexions (voir chapitre précédent). Nous divisons donc les architectures *LRGF* en deux catégories : *retour local de l'activation* et *retour local de la sortie*. Ces deux catégories telles qu'elles sont présentées dans ce chapitre, sont illustrées sur la Figure 48. D'après A.C. Tsoi et A.D. Back (Tsoi *et al.,* 1994), nous pouvons recenser deux modèles de neurones pour la représentation *LRGF* à retour local de la sortie : le modèle de *Frasconi-Gori-Soda* et celui de *Poddar-Unnikrishnan*, et un seul modèle pour l'architecture *LRGF* à retour local de l'activation, le modèle de *Frasconi-Gori-Soda*.

Les conclusions de l'article de A.C. Tsoi et A.D. Back sont nettement en faveur des architectures *LRGF* dont les performances ont été comparées à celles des réseaux globalement récurrents (précisément le réseau récurrent de *Williams-Zipser*[23] (Williams *et al.,* 1989)). En effet, les auteurs ont comparé quatre architectures neuronales : le réseau de *Back-Tsoi* (présenté au chapitre précédent), le neurone à retour local de la sortie *Frasconi-Gori-Soda*, le réseau entièrement récurrent de *Williams-Zipser* et le *TDNN* (présenté au chapitre précédent). La comparaison a été faite sur un problème de prédiction d'une fonction non-linéaire. L'horizon des retards utilisés pour les 4 architectures est de 5 unités : une fenêtre temporelle de 5 retards pour le *TDNN*, cinq couches de neurones bouclés pour les neurones de *Frasconi-Gori-Soda* et de *Back-Tsoi* (avec bouclage unitaire) et un *PMC* à cinq couches entièrement

[23] Réseau récurrent dont l'algorithme d'apprentissage *RTRL* a été présenté au chapitre précédent.

récurrent pour le modèle de *Williams-Zipser*. Le réseau le plus performant est celui de *Frasconi-Gori-Soda* suivi par le *TDNN*. Le réseau le moins performant est le réseau *Williams-Zipser* avec la convergence la plus lente. Le réseau de *Williams-Zipser* souffre également de difficultés considérables pour la modélisation de signaux dynamiques (précisément des signaux vocaux).

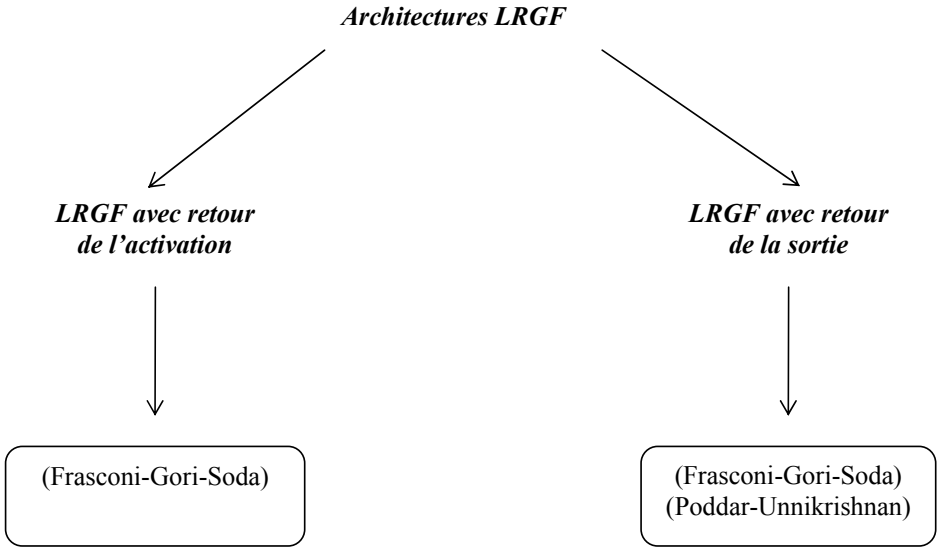


Figure 48. *Représentation des différentes architectures LRGF.*

IV.3.1. Architecture LRGF avec retour de la sortie

La première représentation implicite du temps au niveau du neurone est caractérisée par un bouclage de la sortie du neurone vers son entrée. Ce type de neurone est appelé neurone à retour local de la sortie ou *Local Output Feedback network.* Nous pouvons recenser en littérature deux types de neurones à retour local de la sortie (Tsoi *et al.,* 1994) : le modèle de *Frasconi-Gori-Soda* illustré sur la Figure 49 (Frasconi *et al.,* 1992), (Gori *et al.,* 1989), (Gori *et al.,* 1990) et le modèle de *Poddar-Unnikrishnan* illustré par la Figure 58 (Poddar *et al.,*1991-a-), (Poddar *et al.,* 1991-b-).

IV.3.1.1. Le modèle de Frasconi-Gori-Soda

L'élément qui différencie le modèle du retour local de l'activation et celui du retour local de la sortie introduits par Frasconi-Gori-Soda est, bien évidemment le point de retour qui se situe au niveau de l'activation dans le premier modèle et au niveau de la réponse du neurone dans le second modèle (Figure 49).

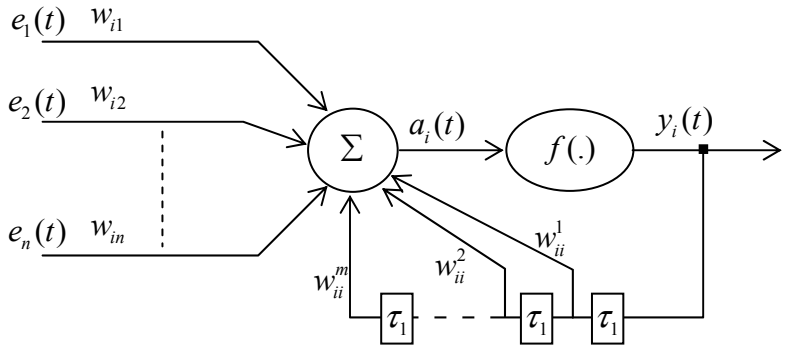


Figure 49. *Architecture générale d'un réseau LRGF avec bouclage de la sortie (modèle de Frasconi-Gori-Soda). Les* τ_1 *représentent des retards unitaires.*

Dans ce modèle, l'évolution de la sortie du neurone dépend de ses propres réponses antérieures. En d'autres termes, l'activation du neurone est fonction de ses entrées pondérées (comme le neurone classique) et des valeurs retardées de sa sortie. Sa sortie $y_i(t)$ est donc définie par les équations suivantes :

$$\begin{aligned} y_i(t) &= f\left(a_i(t)\right) \\ a_i(t) &= \sum_{j=1}^{n} w_{ij} e_j(t) + \sum_{q=1}^{m} w_{ii}^q y_i(t-q) \end{aligned} \qquad [132]$$

Lorsqu'il s'agit d'un neurone d'entrée au réseau et si l'on simplifie le modèle de la Figure 49 en ne considérant qu'un seul retard, son activation peut être ramenée à l'équation ci-dessous. Ce type de neurone est communément appelé *neurone bouclé* (Figure 50).

$$a_i(t) = \xi_i(t) + w_{ii} y_i(t-1) \qquad [133]$$

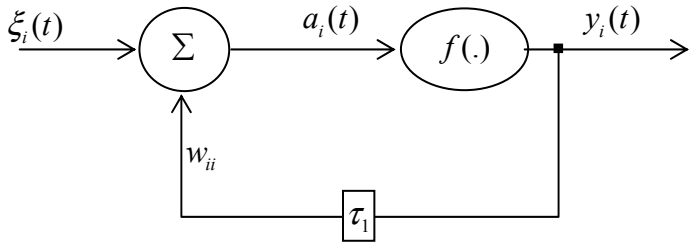


Figure 50. *Architecture simplifiée d'un Réseau LRGF avec retour de la sortie.*

Pour étudier l'impact du bouclage de la sortie sur le comportement dynamique du neurone, nous allons considérer son évolution en l'absence de toute excitation extérieure ($\xi_i(t) = 0 \quad \forall t > 0$) (Frasconi *et al.,* 1992), (Bernauer, 1996). La fonction d'activation $f(.)$ du neurone bouclé est la sigmoïde[24] (Figure 51) définie par :

$$f(x) = \frac{1 - \exp^{-bx}}{1 + \exp^{-bx}} \qquad [134]$$

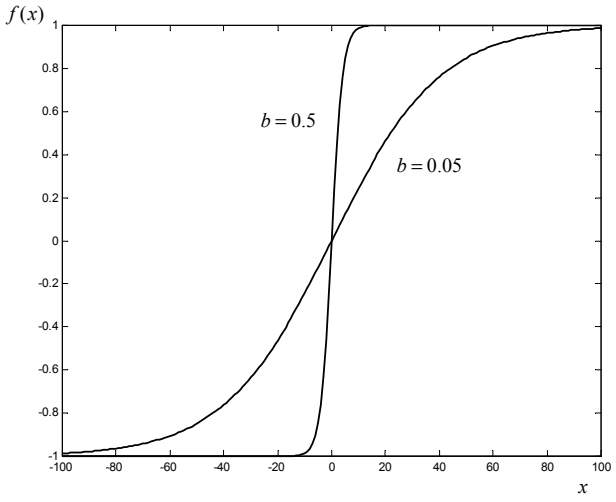


Figure 51. *Allure de la sigmoïde en fonction du paramètre b.*

a) Points d'équilibre

L'activation du neurone bouclé en l'absence de toute excitation extérieure est définie par l'équation suivante :

[24] Tangente Hyperbolique. Nous supposons que $b > 0$ ce qui rend cette fonction strictement croissante.

$$a_i(t) = w_{ii} f\left(a_i(t-1)\right) \qquad [135]$$

La recherche des points d'équilibre de cette équation, c'est-à-dire de l'ensemble des points a^* tels que $a^* = w_{ii} f(a^*)$, revient à chercher les racines de la fonction g définie par[25] :

$$g(a) = w_{ii} f(a) - a \qquad [136]$$

D'après la définition de la fonction $f(x)$, $f(0) = 0$. Le point $a_0 = 0$ est donc une solution évidente de $g(a) = 0$. La recherche des autres racines de $g(a)$ se fait par l'étude de ses variations. Soit $g'(a)$ la fonction dérivée de $g(a)$ définie par :

$$g'(a) = -\frac{(\exp^{-ba})^2 + 2\exp^{-ba}(1 - bw_{ii}) + 1}{(1 + \exp^{-ba})^2} \qquad [137]$$

Pour trouver le signe de $g'(a)$ dans son domaine de définition qui est l'ensemble des réels ($\Re$), nous cherchons les racines de $g'(a) = 0$. En posant $X = \exp^{-ba}$ on obtient l'équation du second degré suivante :

$$X^2 + 2(1 - bw_{ii})X + 1 = 0 \qquad [138]$$

Ces solutions sont fonction du déterminant $\Delta = 4bw_{ii}(bw_{ii} - 2)$. Deux cas sont alors possibles.

- Si $bw_{ii} > 2$, l'équation [138] possède les deux solutions positives ($X_1 X_2 = 1$)

$$\begin{aligned} X_1 &= (bw_{ii} - 1) + \sqrt{bw_{ii}(bw_{ii} - 2)} \Rightarrow X_1 > 1 \\ X_2 &= (bw_{ii} - 1) - \sqrt{bw_{ii}(bw_{ii} - 2)} \Rightarrow X_2 < 1 \end{aligned} \qquad [139]$$

Dans ce cas, $g'(a) = 0$ possède deux solutions :

$$\begin{aligned} a_1 &= -\frac{\log X_1}{b} \Rightarrow a_1 < 0 \\ a_2 &= -\frac{\log X_2}{b} \Rightarrow a_2 > 0 \end{aligned} \qquad [140]$$

A partir du calcul des limites de $g(a)$ qui donne $\lim_{a \to -\infty} g(a) = +\infty$ et $\lim_{a \to +\infty} g(a) = -\infty$ on obtient le tableau des variations suivant :

[25] Nous considérons bien évidemment dans tout ce chapitre que l'auto-connexion $w_{ii} \neq 0$, sinon on ne peut pas parler de réseaux *LRGF*.

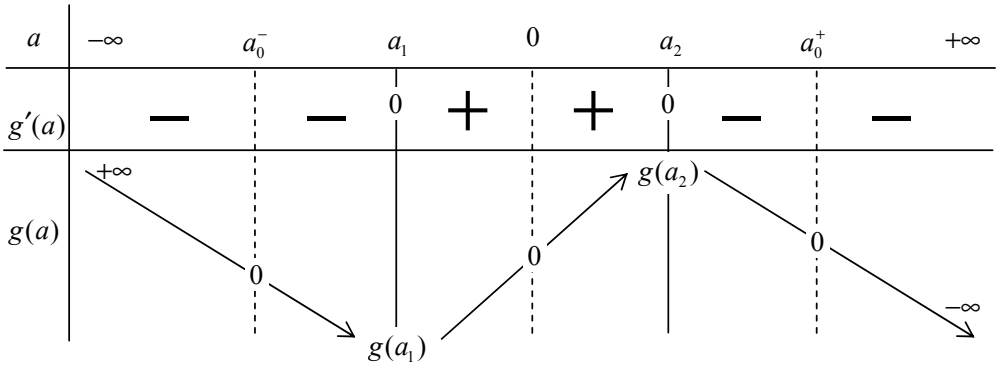
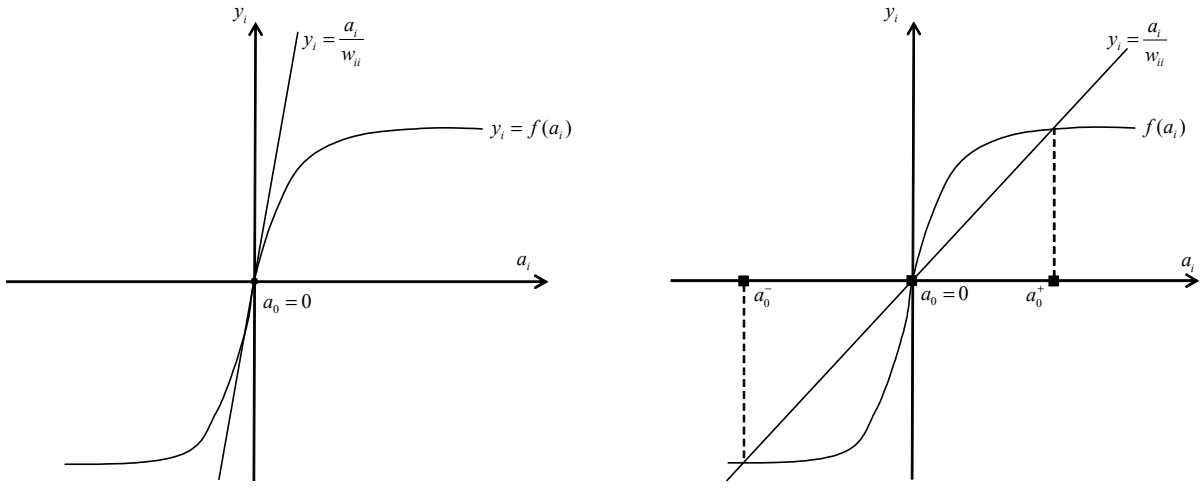


Tableau 2. *Tableau de variation de la fonction g(a).*

La fonction $g(a)$ s'annule donc sur $[-\infty, a_1]$ au point a_0^- et sur $[a_2, +\infty]$ au point a_0^+. Le neurone bouclé possède alors trois points d'équilibre a_0^-, a_0 et a_0^+ pour le cas où $bw_{ii} > 2$.

- Si $bw_{ii} \leq 2$ la fonction $g(a)$ est toujours décroissante et donc l'équation $g(a) = 0$ ne possède qu'une seule solution. Le neurone bouclé ne possède alors qu'un seul point d'équilibre a_0.



$bw_{ii} \leq 2$ $bw_{ii} > 2$

Figure 52. *Points d'équilibre du neurone bouclé en fonction du produit* bw_{ii}.

b) Stabilité des points d'équilibre

Commençons par étudier le cas où le neurone bouclé ne possède qu'un seul point d'équilibre $a_0 = 0$ ($bw_{ii} \leq 2$). Nous allons utiliser pour ce cas, la fonction de Lyapunov définie par $V(a) = a^2$ pour étudier la stabilité du point d'équilibre. D'après l'équation [135]

$$\Delta V = \left(w_{ii} f(a)\right)^2 - a^2 = g(a)\left(w_{ii} f(a) + a\right) \qquad [141]$$

Si $a > a_0 \Leftrightarrow a > 0$, alors $f(a) > 0$. D'après les variations de la fonction $g(a)$ qui est toujours décroissante dans ce cas, on a bien $g(a) < 0$. Si $w_{ii} > 0$ on a bien $\Delta V < 0$. Inversement, si $a < a_0 \Leftrightarrow a < 0$, alors $f(a) < 0$ et $g(a) > 0$. Si $w_{ii} > 0$ on a bien $\Delta V < 0$. D'après la théorie de la stabilité de Lyapunov, le point $a_0 = 0$ est donc un point d'équilibre stable si $bw_{ii} \leq 2$ avec $w_{ii} > 0$.

Dans le deuxième cas où $bw_{ii} > 2$, le neurone bouclé possède trois points d'équilibre a_0^-, a_0 et a_0^+. Si l'on étudie la stabilité du point a_0^+, la fonction de Lyapunov est définie par $V(a) = (a - a_0^+)^2$. D'après l'équation [135], nous pouvons écrire :

$$\Delta V = \left(w_{ii} f(a) - a_0^+\right)^2 - (a - a_0^+)^2 = g(a)\left(g(a) + 2(a - a_0^+)\right) \qquad [142]$$

Si $a > a_0^+$, d'après le Tableau 2, $g(a) < 0$. Puisque $f(.)$ est croissante, $w_{ii} > 0$, et comme a_0^+ est un point d'équilibre pour lequel $w_{ii} f(a_0^+) = a_0^+$, on peut écrire :

$$a > a_0^+ \Leftrightarrow w_{ii} f(a) > a_0^+ \Leftrightarrow g(a) + 2(a - a_0^+) > (a - a_0^+) > 0 \qquad [143]$$

alors $\Delta V < 0$. Si $a < a_0^+$, d'après le Tableau 2, $g(a) > 0$ et $\left(g(a) + 2(a - a_0^+)\right) < 0$, on a bien $\Delta V < 0$.

Le même raisonnement peut être fait pour le point a_0^-.

Résultat :

Le neurone bouclé de Frasconi-Gori-Soda possède donc

- *un seul point d'équilibre stable* $a_0 = 0$ si $bw_{ii} \leq 2$ *avec* $w_{ii} > 0$,
- *deux points d'équilibre stables* a_0^- *et* a_0^+*, et un point d'équilibre instable* $a_0 = 0$ *si* $bw_{ii} > 2$.

c) Comportement d'oubli

Considérons le neurone bouclé de la Figure 50 dont l'évolution est donnée par l'équation suivante :

$$\begin{aligned} y_i(t) &= f\left(a_i(t)\right) \\ a_i(t) &= \xi_i(t) + w_{ii} y_i(t-1) \end{aligned} \qquad [144]$$

avec $f(.)$ la fonction d'activation sigmoïde du neurone bouclé (équation [134]) et $w_{ii} > 0$.

Propriété

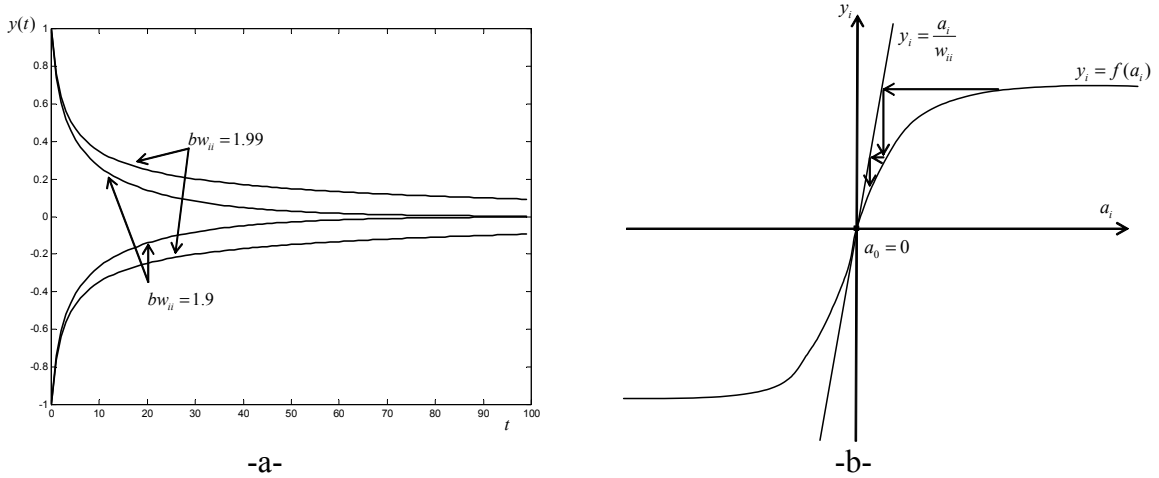
On dira qu'un neurone bouclé possède un comportement d'oubli si pour un instant donné t_0 et une activation $a_i(t_0)$ quelconque,

$$\exists q > 0,\ \forall \varepsilon > 0,\ \forall t > t_0 + q,\ \forall \theta > 0$$

telles que les relations suivantes soient respectées :

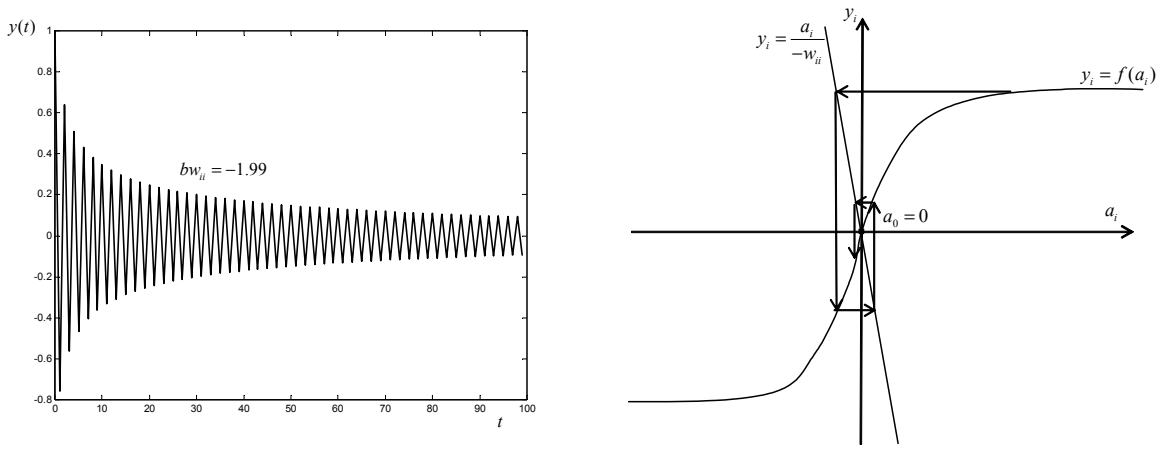
$$\begin{cases} si\ \ a_i(t_0) > \theta \ \ \Rightarrow \ \ \varepsilon > a_i(t) \geq 0 \\ si\ \ a_i(t_0) < -\theta \Rightarrow -\varepsilon < a_i(t) \leq 0 \end{cases} \qquad [145]$$

Lors de l'étude de la stabilité des points d'équilibre (paragraphe précédent) nous avons constaté que le neurone bouclé ne possède qu'un seul point d'équilibre stable $a_0 = 0$ si $bw_{ii} \leq 2$ avec $w_{ii} > 0$ et en l'absence de toute excitation extérieure $\xi_i(t) = 0 \quad \forall t > 0$. Dans ce cas, pour n'importe quelle activation $a_i(t) \neq 0$ du neurone bouclé, l'activation du neurone tend vers $a_0 = 0$. Par conséquent nous obtenons $\lim_{t\to\infty} a_i(t) = a_O = 0$. Le neurone bouclé possède bien un comportement d'oubli si $bw_{ii} \leq 2$ et $w_{ii} > 0$. La figure ci-dessous illustre bien ce comportement.



-a- -b-

Figure 53. *Comportement d'oubli : a) Convergence de la sortie du neurone bouclé vers le point d'équilibre stable $a = a_0$ en fonction de bw_{ii} avec une excitation initiale $\xi(0) = \pm 1$ et $w_{ii} > 0$. b) Etapes du comportement d'oubli entre la sortie du neurone bouclé et son activation.*



-a- -b-

Figure 54. *Comportement d'oubli du neurone bouclé : a) oscillations dues à la valeur négative de l'auto-connexion* $w_{ii} < 0$. *b) Etapes du comportement d'oubli entre sortie du neurone bouclé et son activation.*

d) Comportement de mémorisation

Propriété

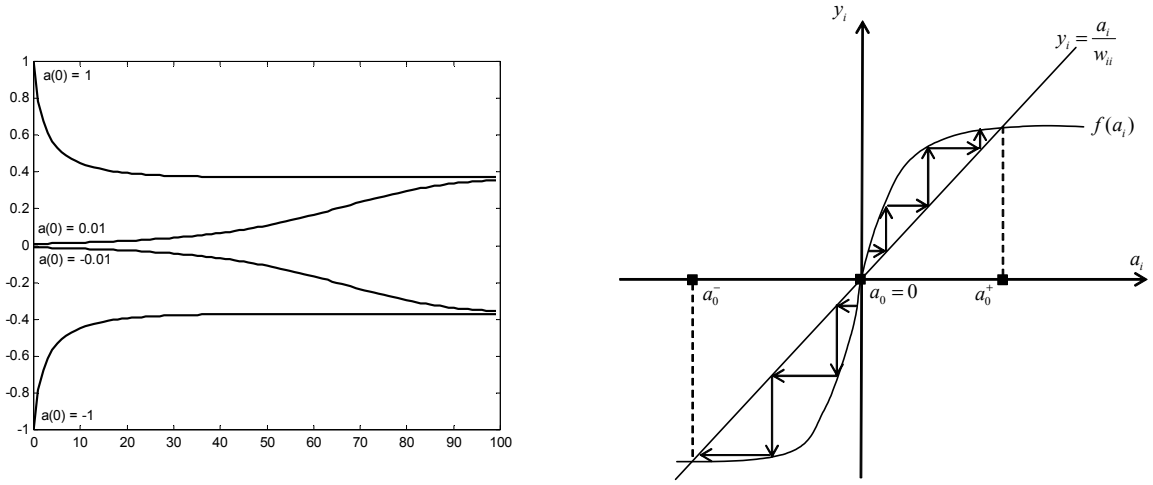
On dira qu'un neurone bouclé possède un comportement de mémorisation si pour un instant donné t_0 *et une activation* $a_i(t_0)$,

$$\exists \varepsilon > 0 : \forall t > t_0$$

telles que les relations suivantes soient respectées :

$$\begin{cases} si \;\; a_i(t_0) > 0 \Rightarrow a_i(t) \geq \varepsilon > 0 \\ si \;\; a_i(t_0) < 0 \Rightarrow a_i(t) \leq -\varepsilon < 0 \end{cases} \qquad [146]$$

Dans le paragraphe précédent, nous avons constaté que le neurone bouclé possède trois points d'équilibre $a_0^- < 0$, $a_0 = 0$ et $a_0^+ > 0$ si $bw_{ii} > 2$. D'après la théorie de Lyapunov, les deux points a_0^- et a_0^+ sont des points d'équilibre stables et $a_0 = 0$ est un point d'équilibre instable. Donc si la sortie du neurone bouclé s'éloigne du point $a_0 = 0$ du côté supérieur $a(t_0) > a_0$ ou inférieur $a(t_0) < a_0$ c'est pour tendre vers un point d'équilibre stable (vers a_0^+ pour le premier cas et a_0^- pour le deuxième). La figure suivante illustre bien ce comportement.



-a- -b-

Figure 55. *Comportement de mémorisation a) Convergence de la sortie du neurone bouclé vers le point d'équilibre stable* a_0^+ *pour* $a(0) > 0$ *et vers* a_0^- *pour* $a(0) < 0$ *avec* $bw_{ii} = 2.1$. *b) Etapes du comportement de mémorisation entre sortie du neurone bouclé et son activation.*

Ce comportement de mémorisation avec deux points d'équilibre stables reste valable même si on ajoute à l'activation du neurone un terme extérieur constant ξ_i tel que (Frasconi *et al.,* 1995) :

$$y_i(t) = f\left(\xi_i + w_{ii} y_i(t-1)\right) \text{ et } \left|\xi_i\right| < I^* \qquad [147]$$

avec I^* le point d'intersection de la tangente de $f(.)$ parallèle à la droite $y_i = a_i / w_{ii}$ avec la droite $y_i = 0$ (Figure 56). Dans le cas contraire, c'est-à-dire si $\left|\xi_i\right| > I^*$, il n'y aura plus qu'un seul point d'équilibre stable (positif si $\xi_i > I^*$ et négatif si $\xi_i < -I^*$), et donc la sortie du neurone bouclé va basculer en un nombre fini de transitions vers cet état d'équilibre proche de 1. On peut dire que ce comportement est analogue à celui d'une machine booléenne.

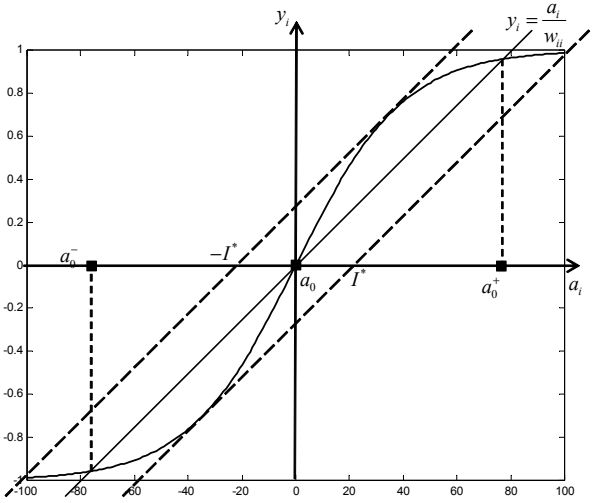


Figure 56. *Limites du comportement de mémorisation à deux points d'équilibre stable du neurone bouclé.*

e) Longueur de la mémoire dynamique du neurone bouclé

Propriété

La limite de la longueur de la mémoire dynamique d'un neurone bouclé peut être quantifiée par l'étude de l'influence d'une variation $\partial\xi_i(t)$ *de l'entrée à un instant* t *donné sur les variations de la sortie du neurone bouclé* $\partial y_i(t+q)$ *à un instant* $t+q$ *. On dira qu'un neurone bouclé a atteint sa limite maximum de mémorisation dynamique* $q_{\max}$ *si au bout d'un certain temps, la variation de l'entrée* $\partial\xi_i(t)$ *n'a aucune influence sur la variation de la sortie* $\partial y_i(t+q)$*. On peut exprimer ceci par l'équation suivante :*

$$\lim_{q\to\infty} \frac{\partial y_i(t)}{\partial \xi_i(t-q)} = 0 \qquad [148]$$

Cette équation peut être développée comme suit :

$$\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = \frac{\partial y_i(t)}{\partial a_i(t)} \frac{\partial a_i(t)}{\partial y_i(t-1)} \frac{\partial y_i(t-1)}{\partial \xi_i(t-q)} = f'\left(a_i(t)\right) w_{ii} \frac{\partial y_i(t-1)}{\partial \xi_i(t-q)} \qquad [149]$$

En développant davantage cette équation, on peut écrire :

$$\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = f'\left(a_i(t)\right) f'\left(a_i(t-1)\right) ..... f'\left(a_i(t-q+1)\right) {w_{ii}}^q \frac{\partial y_i(t-q)}{\partial \xi_i(t-q)} \qquad [150]$$

qui peut être reformulée par :

$$\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = \prod_{j=0}^{q} f'\left(a_i(t-j)\right) w_{ii}^{\,q} \qquad [151]$$

D'après les variations de la fonction $f'(.)$:

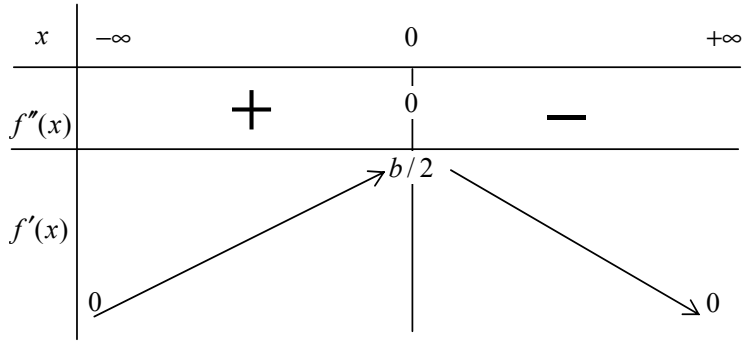


Tableau 3. *Tableau de variation de la fonction* $f'(x)$.

nous avons $0 < f'(.) \le b/2$ et, par conséquent :

$$0 < \frac{\partial y_i(t)}{\partial \xi_i(t-q)} \le (bw_{ii}/2)^q (b/2) \qquad [152]$$

On peut aisément démontrer que pour $bw_{ii} < 2$

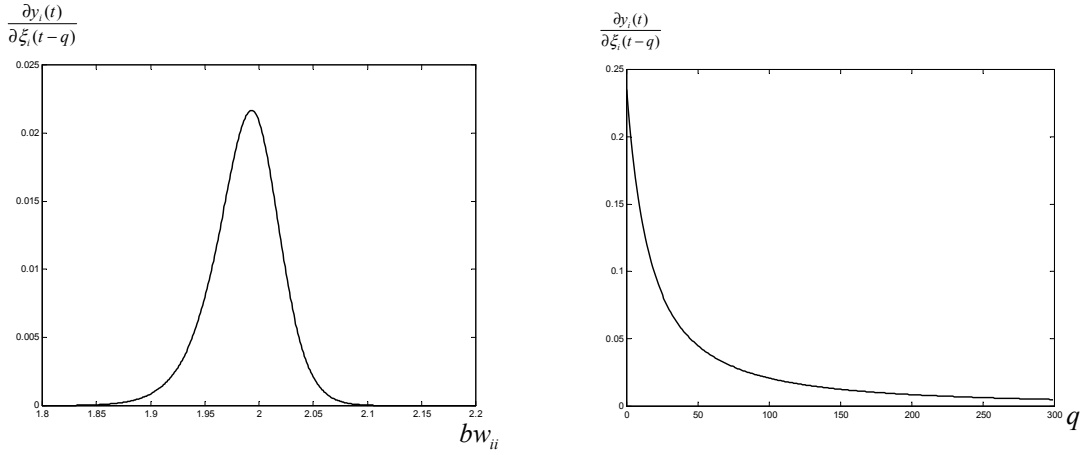
$$\lim_{q\to\infty} \frac{\partial y_i(t)}{\partial \xi_i(t-q)} \le \lim_{q\to\infty} (bw_{ii}/2)^q (b/2) = 0 \qquad [153]$$

Dans ce cas, le neurone bouclé possède bien une limite de mémorisation. Pour l'étude de la limite de mémorisation du neurone bouclé dans l'autre cas (c'est-à-dire $bw_{ii} \ge 2$), nous avons précédé par des simulations (sous MATLAB) de l'équation [151] en fonction de bw_{ii}. Les différentes étapes de la simulation sont les suivantes :

- on initialise la valeur de $\left.\frac{\partial y_i(t)}{\partial \xi_i(t-q)}\right|_{q=0} = f'\left(a_i(t)\right) w_{ii}^{\,0} = b/2$ (valeur qui correspond au maximum de $f'\left(a_i(t)\right)$ - voir Tableau 3 -, avec $b = 0.5$,

- on calcule $\left.\frac{\partial y_i(t)}{\partial \xi_i(t-q)}\right|_{q=0,,,,100}$, et on retient la valeur de $\left.\frac{\partial y_i(t)}{\partial \xi_i(t-q)}\right|_{q=100}$,
- on refait la même procédure pour plusieurs valeurs de bw_{ii} (plus précisément en faisant varier w_{ii} puisque $b = 0.5$) afin de comparer les valeurs retenues en fonction de bw_{ii}.

On obtient les résultats schématisés par la Figure 57.a. qui montre que le maximum $\frac{\partial y_i(t)}{\partial \xi_i(t-q)}$ est obtenu pour $bw_{ii} = 2$. Nous avons ensuite voulu quantifier cette valeur maximale de la longueur de la mémoire dynamique. Nous avons donc fixé $bw_{ii} = 2$ et calculé $\frac{\partial y_i(t)}{\partial \xi_i(t-q)}$ en fonction de q. On obtient une longueur maximale d'environ 300 unités de temps (Figure 57.b.).



-a- -b-

Figure 57. *Longueur de la mémoire dynamique du neurone bouclé avec les conditions initiales suivantes :* $\left.\frac{\partial y_i(t)}{\partial \xi_i(t-q)}\right|_{q=0} = f'\left(a_i(t)\right) w_{ii}^{\,0} = b/2$ *avec* $b = 0.5$ *a) en fonction du produit* bw_{ii} *pour un q donné (*$q = 100$*) b) en fonction de q avec une configuration de longueur de mémoire maximum (*$bw_{ii} = 2$*) obtenue à partir du premier graphe.*

IV.3.1.2. Le modèle de Poddar-Unnikrishnan

La deuxième représentation locale du temps par retour de la sortie est celle du modèle de *Poddar-Unnikrishnan* illustré par la Figure 58. La sortie du neurone est régie par les relations ci-dessous :

$$\begin{aligned} y_i(t) &= f\left(a_i(t)\right) \\ a_i(t) &= \sum_{j=1}^{n} w_{ij} e_j(t) + w_{ii} z_i(t) \\ z_i(t) &= \alpha y_i(t-1) + (1-\alpha) z_i(t-1) \end{aligned} \qquad [154]$$

On remarque que le neurone de *Frasconi-Gori-Soda* présenté précédemment représente un cas particulier du modèle de *Poddar-Unnikrishnan* où $\alpha = 1$.

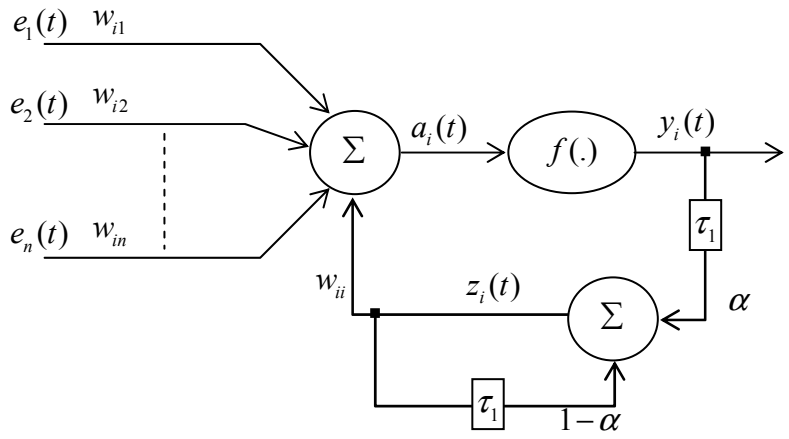


Figure 58. *Architecture générale d'un réseau LRGF avec bouclage de la sortie (modèle de Poddar-Unnikrishnan). Les τ_1 représentent des retards unitaires.*

Lorsqu'il s'agit d'un neurone d'entrée au réseau, son activation peut être ramenée à l'équation suivante :

$$a_i(t) = \xi_i(t) + w_{ii} z_i(t) \qquad [155]$$

Comme pour les cas précédents, nous allons étudier les propriétés de ce type de neurone à retour local de la sortie, en l'absence de toute excitation extérieure ($\xi_i(t) = 0 \quad \forall t > 0$).

a) Points d'équilibre

En l'absence de toute excitation extérieure, l'activation du neurone peut être réécrite de la façon suivante :

$$\begin{aligned} a_i(t) &= w_{ii} z_i(t) = w_{ii}\left(\alpha y_i(t-1) + (1-\alpha) z(t-1)\right) \\ a_i(t) &= w_{ii}\left(\alpha f\left(a_i(t-1)\right) + (1-\alpha)\frac{a_i(t-1)}{w_{ii}}\right) \\ a_i(t) &= w_{ii}\alpha f\left(a_i(t-1)\right) + (1-\alpha) a_i(t-1) \end{aligned} \qquad [156]$$

La recherche des points d'équilibre de cette équation revient à chercher l'ensemble des points a^* tels que :

$$a^* = w_{ii}\alpha f\left(a^*\right) + (1-\alpha)a^* \Rightarrow \alpha\left(w_{ii} f(a^*) - a^*\right) = 0 \qquad [157]$$

En d'autres termes, ceci revient à chercher les racines de la fonction g définie précédemment par l'équation [136]. Si on prend comme fonction d'activation la fonction tangente hyperbolique (sigmoïde) définie par l'équation [134], nous retrouvons les mêmes résultats que pour l'étude des points d'équilibre du neurone bouclé, c'est-à-dire :

- si $bw_{ii} \leq 2$ le neurone ne possède qu'un seul point d'équilibre $a_0 = 0$,
- si $bw_{ii} > 2$ le neurone possède trois points d'équilibre a_0^-, a_0 et a_0^+.

b) Stabilité des points d'équilibre

Comme pour le précédent neurone bouclé, nous étudions le premier cas où le neurone ne possède qu'un seul point d'équilibre $a_0 = 0$ ($bw_{ii} \leq 2$). La fonction de Lyapunov qui permet d'étudier la stabilité du point a_0 est définie par :

$$\Delta V = \left(a_i(t+1)\right)^2 - a_i(t)^2 = \left(w_{ii}\alpha f\left(a_i(t)\right) + (1-\alpha)a_i(t)\right)^2 - a_i(t)^2 \qquad [158]$$

En omettant volontairement l'indice *i* et la variable temporelle *t* et en développant l'équation précédente [158], on obtient les variations de la fonction de Lyapunov suivante :

$$\Delta V = \alpha g(a)(\alpha g(a) - 2a) = \alpha g(a)\left((\alpha-1)g(a) + w f(a) + a\right) \qquad [159]$$

avec $g(a)$ fonction définie précédemment (équation [136]). On peut alors se baser sur l'étude faite précédemment sur le neurone bouclé pour étudier la stabilité du point $a_0 = 0$. Si $a > a_0 \Leftrightarrow a > 0$, alors $f(a) > 0$. D'après les variations de la fonction $g(a)$ qui est toujours décroissante dans ce cas, on a bien $g(a) < 0$. Si $w_{ii} > 0$, $(\alpha - 1) < 0$ et $\alpha > 0$ on a bien $\Delta V < 0$. Inversement, si $a < a_0 \Leftrightarrow a < 0$, alors $f(a) < 0$ et $g(a) > 0$. Si $w_{ii} > 0$, $(\alpha - 1) < 0$ et $\alpha > 0$ on a bien $\Delta V < 0$. D'après la théorie de la stabilité de Lyapunov, le point $a_0 = 0$ est donc un point d'équilibre stable si $bw_{ii} \leq 2$ avec $w_{ii} > 0$ et $\alpha < 1$.

Dans le deuxième cas où $bw_{ii} > 2$, le neurone possède trois points d'équilibre a_0^-, a_0 et a_0^+. Si l'on étudie la stabilité du point a_0^+, la fonction de Lyapunov est définie par $V(a) = (a - a_0^+)^2$. Ses variations sont :

$$\Delta V = \left(a(t+1) - a_0^+\right)^2 - (a(t) - a_0^+)^2 = \alpha g(a)\left((\alpha - 1)g(a) + g(a) + 2(a - a_0^+)\right) \quad [160]$$

D'après l'étude faite pour la stabilité des points d'équilibre dans les mêmes conditions de $bw_{ii} > 2$ pour le neurone bouclé, on peut affirmer : pour $a > a_0^+$, d'après le Tableau 2 $g(a) < 0$, $\left(g(a) + 2(a - a_0^+)\right) > 0$, si $(\alpha - 1) < 0$ et $\alpha > 0$ alors $\Delta V < 0$. Inversement, pour $a < a_0^+$, d'après le Tableau 2, $g(a) > 0$ et $\left(g(a) + 2(a - a_0^+)\right) < 0$, si $(\alpha - 1) < 0$ et $\alpha > 0$ on a bien $\Delta V < 0$.

Résultat :

Le neurone bouclé de Poddar-Unnikrishnan possède donc

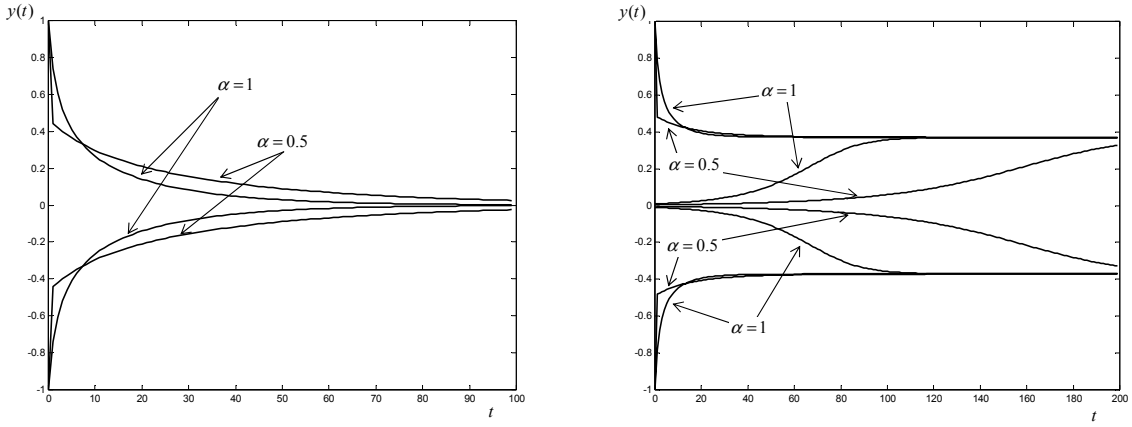
- *un seul point d'équilibre stable* $a_0 = 0$ *si* $bw_{ii} \leq 2$ *avec* $w_{ii} > 0$ *et* $0 < \alpha < 1$,
- *deux points d'équilibre stables* a_0^- *et* a_0^+, *et un point d'équilibre instable* $a_0 = 0$ *si* $bw_{ii} > 2$ *et* $0 < \alpha < 1$.

c) Comportement d'oubli et de mémorisation

En adoptant le même raisonnement que pour le neurone bouclé de Frasconi-Gori-Soda, le neurone de Poddar-Unnikrishnan possède bien deux comportements distincts en fonction de bw_{ii} pour $0 < \alpha < 1$:

- *Un comportement d'oubli si* $bw_{ii} \leq 2$ *avec* $w_{ii} > 0$,
- *Un comportement de mémorisation si* $bw_{ii} > 2$.

La figure suivante illustre ces deux comportements avec l'influence du paramètre α.



-a- -b-

Figure 59. *Influence des paramètres* α *et* bw_{ii} *sur le comportement du neurone bouclé de Poddar-Unnikrishnan a) Comportement d'oubli avec* $bw_{ii} = 1.9$ *(*$\xi(0) = \pm 1$*) b) comportement de mémorisation avec* $bw_{ii} = 2.1$ *(*$\xi(0) = \pm 1$ *et* $\xi(0) = \pm 0.01$*).*

d) Longueur de la mémoire dynamique du neurone de Poddar-Unnikrishnan

Comme pour le neurone bouclé de Frasconi-Gori-Soda, nous allons étudier la longueur de la mémoire dynamique du neurone de Poddar-Unnikrishnan à partir de la relation [148]. L'activation $a_i(t)$ du neurone de Poddar-Unnikrishnan (équation [154]) peut être réécrite de la façon suivante :

$$a_i(t) = \xi_i(t) - (1-\alpha)\xi_i(t-1) + \alpha w_{ii} y_i(t-1) + (1-\alpha)a_i(t-1) \qquad [161]$$

On peut alors développer l'étude des variations $\partial y_i(t)$ de la sortie par rapport à celles de l'entrée $\partial \xi_i(t-q)$ comme suit :

$$\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = \frac{\partial y_i(t)}{\partial a_i(t)} \frac{\partial a_i(t)}{\partial y_i(t-1)} \frac{\partial y_i(t-1)}{\partial \xi_i(t-q)} = f'\left(a_i(t)\right) \alpha w_{ii} \frac{\partial y_i(t-1)}{\partial \xi_i(t-q)} \qquad [162]$$

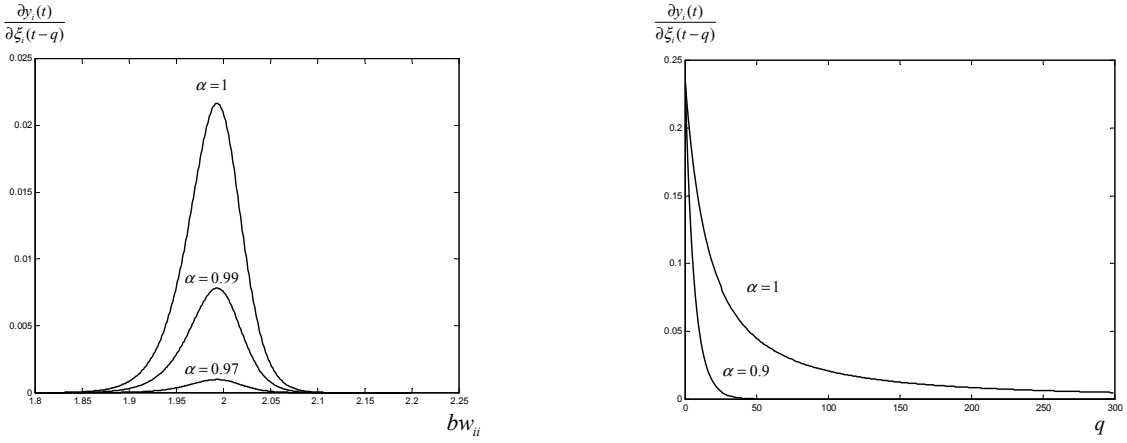
En développant davantage cette équation, on peut écrire :

$$\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = f'\left(a_i(t)\right) f'\left(a_i(t-1)\right) ..... f'\left(a_i(t-q+1)\right) (\alpha w_{ii})^q \frac{\partial y_i(t-q)}{\partial \xi_i(t-q)} \qquad [163]$$

qui peut être reformulée par :

$$\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = \prod_{j=0}^{q} f'\left(a_i(t-j)\right)(\alpha w_{ii})^q \qquad [164]$$

Comme pour le cas du neurone de Frasconi-Gori-Soda présenté précédemment, nous avons procédé par simulation pour quantifier la longueur de la mémoire dynamique du neurone de Poddar-Unnikrishnan. La figure ci-dessous montre que sa longueur maximum est obtenue pour $bw_{ii} = 2$ et pour $\alpha = 1$. Cette longueur maximale est de $q = 300$. On rappelle que le cas où $\alpha = 1$ représente le cas particulier du neurone bouclé de Frasconi-Gori-Soda présenté précédemment.



-a- -b-

Figure 60. *Longueur de la mémoire du neurone de Poddar-Unnikrishnan avec les conditions initiales suivante :* $\left.\frac{\partial y_i(t)}{\partial \xi_i(t-q)}\right|_{q=0} = f'(a_i(t))\, w_{ii}^{\,0} = b/2$ *avec* $b = 0.5$ *a) en fonction du produit* bw_{ii} *et du paramètre* α *pour un q donné (*$q = 100$*) b) en fonction de q et* α *avec une configuration de longueur de mémoire maximum (*$bw_{ii} = 2$*) obtenue à partir du premier graphe.*

IV.3.2. Architecture LRGF avec retour de l'activation

La deuxième représentation locale du temps implicitement au niveau du neurone est celle présentée pas la Figure 61. Cette architecture particulière de réseaux *LRGF* introduite par Frasconi-Gori-Soda (Gori, 1989) est appelée *local activation feedback network* ou réseau à retour local de l'activation.

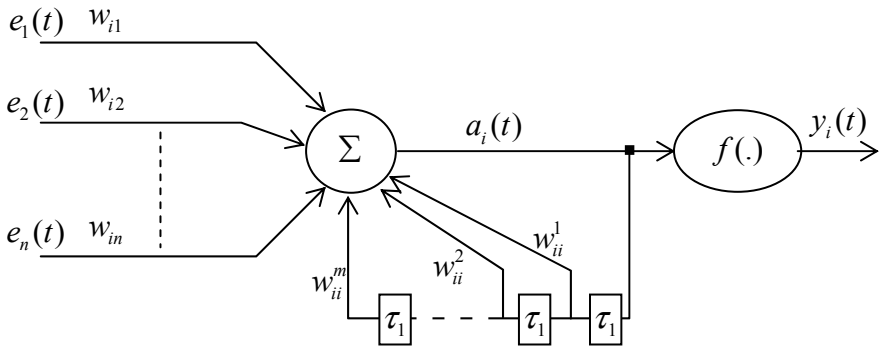


Figure 61. *Architecture générale d'un réseau LRGF avec bouclage de l'activation. Les* τ_1 *représentent des retards unitaires.*

L'évolution de la sortie de ce neurone dépend non seulement de ses entrées pondérées à l'instant *t*, mais également des valeurs retardées de son activation. Sa sortie $y_i(t)$ est définie par les équations :

$$\begin{aligned} y_i(t) &= f\left(a_i(t)\right) \\ a_i(t) &= \sum_{j=1}^{n} w_{ij} e_j(t) + \sum_{q=1}^{m} w_{ii}^q a_i(t-q) \end{aligned} \qquad [165]$$

où $a_i(t)$ représente l'activation du neurone à l'instant *t*, $f(.)$ sa fonction d'activation, $e_j(t)$ son entrée, w_{ij} la valeur du poids de la connexion reliant le neurone *i* au neurone amont *j* et w_{ii}^k représente le poids de l'auto-connexion du neurone *i* avec un retard égal à $q\tau_1$. Cette auto-connexion permet au neurone de garder en mémoire une trace d'un certain passé de ses entrées $e_j(t)$: le neurone est donc doté d'une mémoire dynamique. Ce réseau peut être simplifié en utilisant qu'un seul retard au niveau de l'auto-connexion (Figure 62). L'activation du neurone devient alors :

$$a_i(t) = \sum_{j=1}^{n} \left(w_{ij} e_j(t) \right) + w_{ii} a_i(t-1) \qquad [166]$$

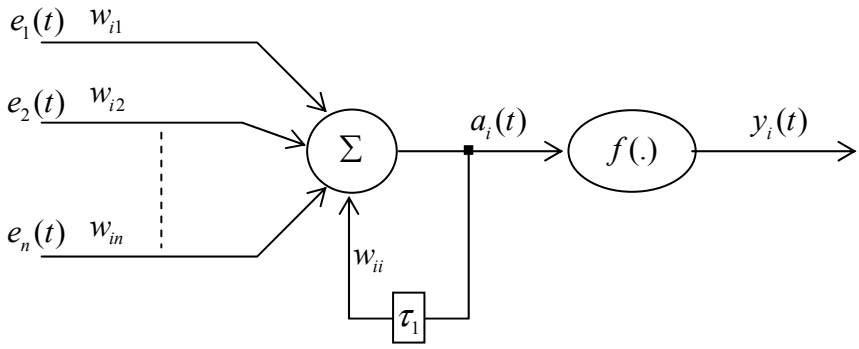


Figure 62. *Architecture simplifiée d'un Réseau LRGF avec retour de l'activation*

Lorsqu'il s'agit d'un neurone d'entrée au réseau, son activation peut être ramenée à l'équation suivante :

$$a_i(t) = \xi_i(t) + w_{ii} a_i(t-1) \qquad [167]$$

L'étude des propriétés de ce type de neurone à retour local de l'activation, revient à étudier son comportement en l'absence de toute excitation extérieure ($\xi_i(t) = 0 \quad \forall t > 0$). En d'autres termes, ceci revient à étudier l'influence du retour local de l'activation sur la mémoire dynamique du neurone.

IV.3.2.1. Points d'équilibre

La recherche des points d'équilibre de ce neurone se traduit par la recherche des racines a^* de l'équation :

$$g(a) = a.(w_{ii} - 1) \qquad [168]$$

Deux cas sont possibles en fonction de w_{ii} :

- si $w_{ii} \neq 1$, le seul point d'équilibre dans ce cas est $a_0 = 0$,
- si $w_{ii} = 1$, tous les points $a \in \Re$ sont des points d'équilibre.

IV.3.2.2. Stabilité des points d'équilibre

Si l'on prend le premier cas, c'est-à-dire le cas où $w_{ii} \neq 1$, l'équation [168] possède qu'une seule racine et donc le neurone ne possède qu'un seul point d'équilibre $a_0 = 0$. Pour étudier sa stabilité, on utilisera la fonction de Lyapunov définie par $V(a) = a^2$. Les variations de cette fonction autour du point d'équilibre $a_0 = 0$ s'expriment par

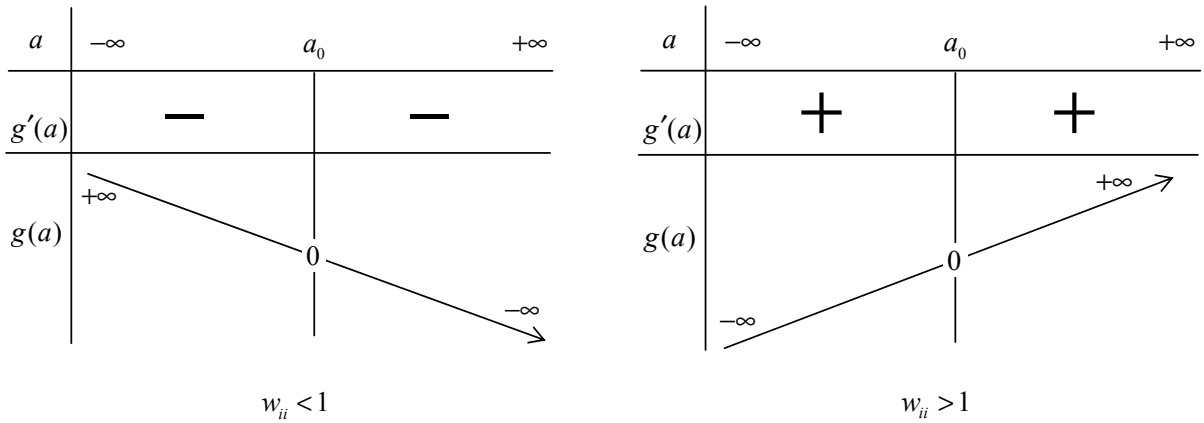
$$\Delta V = (w_{ii}a)^2 - a^2 = a.g(a).(w_{ii}+1) \qquad [169]$$

Si $a > a_0$ et $w_{ii} > 1$, d'après le tableau des variation de g(a) (Tableau 4), $g(a) > 0$ et $w_{ii}+1 > 0$ alors $\Delta V > 0$. Inversement, si $a < a_0$ et avec la même condition de $w_{ii} > 1$, on a bien $\Delta V > 0$. D'après la théorie de la stabilité de Lyapunov, le point $a = a_0$ est donc un point d'équilibre instable si $w_{ii} > 1$.

Plaçons nous maintenant dans le cas où $w_{ii} < 1$. Si $a > a_0$ et $-1 < w_{ii} < +1$, alors $g(a) < 0$ et $w_{ii}+1 > 0$ donc $\Delta V < 0$. Dans le cas où $a < a_0$ et avec les mêmes conditions de w_{ii}, $g(a) > 0$, alors $\Delta V < 0$. Dans ce cas où $-1 < w_{ii} < +1$, le point $a = a_0$ est un point d'équilibre stable.

Le dernier cas est celui où $w_{ii} < -1$, on obtient $w_{ii}+1 < 0$ et de ce fait $\Delta V > 0$. Le point $a = a_0$ est un point d'équilibre instable.

Dans les cas particuliers où $w_{ii} = -1$ ou $w_{ii} = +1$, on a $\Delta V = 0$. Le neurone ne possède aucune dynamique qui rapproche ou écarte l'activation du point d'équilibre $a = a_0$. Dans le cas où $w_{ii} = -1$, l'activation du neurone oscille entre les valeurs $\pm a$ (comportement astable), et reste stable en a pour le cas où $w_{ii} = +1$.



$w_{ii} < 1$ $\qquad\qquad$ $w_{ii} > 1$

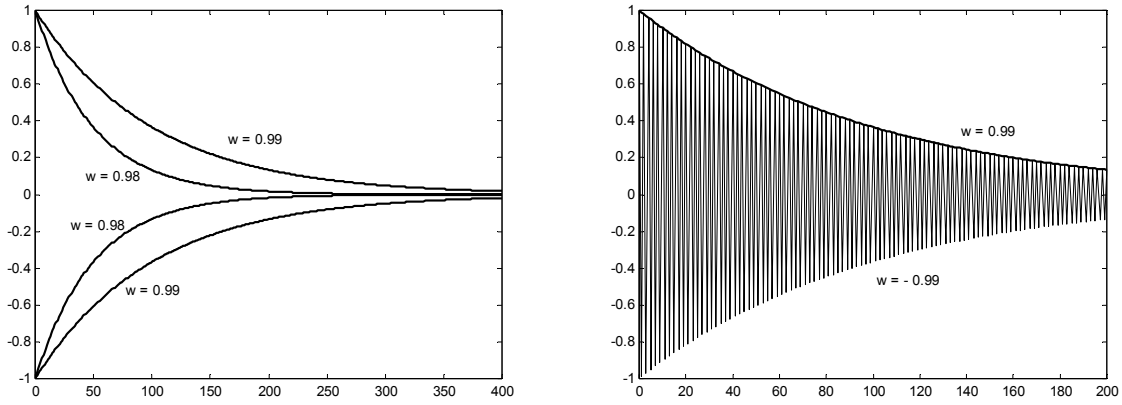
Tableau 4. *Tableau de variation de la fonction g(a).*

IV.3.2.3. Comportement d'oubli et de mémorisation

Comme pour les deux cas précédents, en se basant sur les propriétés des comportements d'oubli et de mémorisation évoquées précédemment, le neurone à retour local de l'activation possède deux comportements distincts en fonction de w_{ii} :

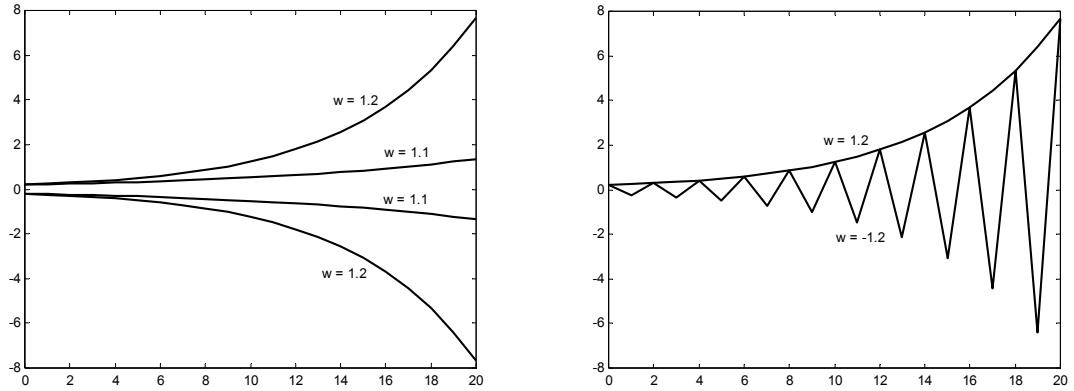
- *un comportement d'oubli si* $|w_{ii}| < 1$,
- *un comportement de mémorisation si* $|w_{ii}| \geq 1$.

Les figures suivantes illustrent ces deux comportements en fonction de w_{ii} :



-a- -b-

Figure 63. *Comportement d'oubli. Convergence de l'activation du neurone vers le point d'équilibre stable* $a = a_0$. *a) sans oscillations pour des valeurs positives de l'auto-connexion, b) avec oscillations pour des valeurs négatives de l'auto-connexion. La condition initiale est donnée par* $\xi(0) = \pm 1$.



-a- -b-

Figure 64. *Comportement de mémorisation. Divergence de l'activation du neurone vers une valeur infinie a) sans oscillations, b) avec oscillations. La condition initiale est donnée par* $\xi(0) = \pm 0.2$.

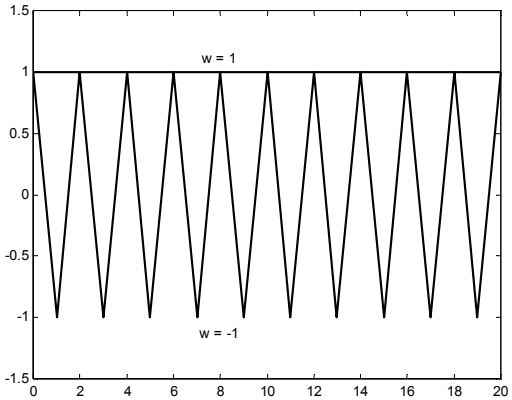


Figure 65. *Stabilisation de l'activation aux conditions d'activation initiales (*$\xi(0) = +1$*) pour* $w = 1$. *Oscillation de cette activation entre* $\pm\xi(0)$ *pour* $w = -1$.

IV.3.2.4. Longueur de la mémoire dynamique du neurone

Comme pour les cas précédents, nous allons utiliser l'équation [148] pour l'étude de la longueur de la mémoire dynamique du neurone à retour local de l'activation. L'étude des variations $\partial y_i(t)$ de la sortie par rapport à celles de l'entrée $\partial \xi_i(t-q)$ s'écrit comme suit :

$$\begin{aligned}\frac{\partial y_i(t)}{\partial \xi_i(t-q)} &= \frac{\partial y_i(t)}{\partial a_i(t)}\frac{\partial a_i(t)}{\partial \xi_i(t-q)} \\ &= f'\left(a_i(t)\right)\frac{\partial a_i(t)}{\partial a_i(t-1)}\frac{\partial a_i(t-1)}{\partial \xi_i(t-q)} \\ &= f'\left(a_i(t)\right)w_{ii}\frac{\partial a_i(t-1)}{\partial a_i(t-2)}\frac{\partial a_i(t-2)}{\partial \xi_i(t-q)} = f'\left(a_i(t)\right)w_{ii}{}^2\frac{\partial a_i(t-2)}{\partial \xi_i(t-q)}\end{aligned}$$

En développant davantage cette équation, on peut écrire :

$$\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = f'\left(a_i(t)\right)w_{ii}{}^q\frac{\partial a_i(t-q)}{\partial \xi_i(t-q)} = f'\left(a_i(t)\right)w_{ii}{}^q \qquad [170]$$

D'après cette relation, on peut conclure que :

- le neurone à retour local de l'activation possède une mémoire de longueur finie si $|w_{ii}|<1$:

$$\lim_{q\to\infty}\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = \lim_{q\to\infty} f'\left(a_i(t)\right)w_{ii}{}^q = 0 \qquad [171]$$

- une mémoire à longueur infinie si $|w_{ii}|>1$

$$\lim_{q\to+\infty}\frac{\partial y_i(t)}{\partial \xi_i(t-q)} = \lim_{q\to+\infty} f'\left(a_i(t)\right)w_{ii}{}^q = +\infty \qquad [172]$$

IV.4. Analyse comparative entre les trois types de mémoires dynamiques du réseau RRFR

Les architectures *LRGF* peuvent être divisées en trois représentations majeures : deux représentations à retour local de la sortie (le neurone de *Frasconi-Gori-Soda* et le neurone de *Poddar-Unnikrishnan*) et une représentation à retour local de l'activation (le neurone de *Frasconi-Gori-Soda*). Ces trois neurones récurrents présentent des comportements quasi-semblables :

- les trois neurones possèdent un comportement d'oubli,
- les trois neurones possèdent un comportement de mémorisation,
- par contre, seul le neurone à retour local de l'activation possède dans certaines conditions une mémoire à longueur infinie. Les deux autres neurones à retour local de la sortie (neurone de *Frasconi-Gori-Soda* et celui de *Poddar-Unnikrishnan*) possèdent une mémoire à longueur limitée (environ 300 unités de temps).

Le modèle du neurone bouclé[26] de *Frasconi-Gori-Soda* représente un cas particulier du modèle de *Poddar-Unnikrishnan* (cas où $\alpha = 1$). D'après l'étude faite précédemment, nous pouvons conclure que le neurone bouclé *Frasconi-Gori-Soda* possède des performances meilleures que celles du neurone de *Poddar-Unnikrishnan*. En effet, la longueur de la mémoire du neurone bouclé de *Frasconi-Gori-Soda* est plus importante que celle du modèle de *Poddar-Unnikrishnan* (voir Figure 59). D'un autre côté, le neurone de *Poddar-Unnikrishnan* possède un paramètre supplémentaire (le paramètre α) à ajuster, ce qui peut dans certains cas, compliquer la procédure de réglage des paramètres optimum du neurone bouclé. On peut dire alors que le modèle de *Frasconi-Gori-Soda* est plus performant que le modèle *Poddar-Unnikrishnan*.

Par contre, la comparaison entre les performances du neurone bouclé de *Frasconi-Gori-Soda* et le neurone à retour local de l'activation, n'est pas évidente. La Figure 66 présente une comparaison des longueurs de la mémoire des trois types de neurones localement récurrents. On peut voir que le neurone à retour local de l'activation possède la mémoire la plus longue pour des valeurs du poids de l'auto-connexion proches de 1.

Néanmoins, une différence majeure existe entre ces deux neurones localement récurrents : il s'agit de l'emplacement du point de retour de la récurrence. En effet, au niveau du neurone bouclé (retour local de la sortie) le point de retour se situe après la non-linéarité du neurone (après la fonction d'activation sigmoïde) ; par contre, au niveau du neurone à retour local de l'activation, le point de retour se situe avant la non-linéarité (avant la fonction d'activation sigmoïde). Ce détail comporte certainement des conséquences considérables aux niveaux des différences de comportements entre les deux mémoires dynamiques. Une question se pose alors à nous : laquelle des deux architectures est plus performante que l'autre dans la prise en compte de la dimension temporelle des systèmes non-linéaires ? Intuitivement, nous dirons que le neurone bouclé de *Frasconi-Gori-Soda* est probablement plus performant que le neurone à retour local de l'activation car la non-linéarité se situe à l'intérieur de la mémoire dynamique.

Pour appuyer notre hypothèse, nous pouvons extrapoler sur l'un des critères de choix de la fonction d'activation des neurones pour avoir un bon approximateur universel. Ce critère est justement d'avoir une fonction d'activation non-linéaire (la sigmoïde pour le *PMC* et la gaussienne pour les réseaux *RFR*). Les réseaux de neurones sont alors capables d'approximer n'importe quelle fonction non-linéaire (relation non-linéaire entrée-sortie) (Tsoi *et al.,* 1994).

[26] Nous rappelons que le neurone bouclé représente le neurone à retour local de la sortie de Frasconi-Gori-Soda.

En effet, dans (Tsoi *et al.,* 1994), les auteurs insistent sur l'avantage d'utiliser des fonctions d'activation non-linéaires pour la prise en compte de la non-linéarité des données d'entrée. L'emplacement du bouclage : avant la non-linéarité ou après le passage à la non-linéarité peut donc être un critère de choix décisif pour le type de mémoire dynamique.

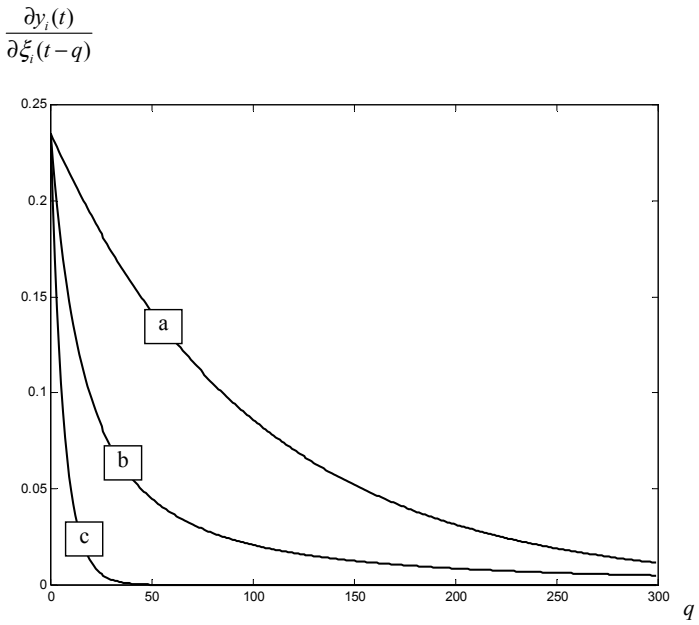


Figure 66. *Comparaison de la longueur de la mémoire des trois architectures LRGF présentées précédemment. Le graphe (a) représente le neurone à retour local de l'activation avec* $w_{ii} = 0.99$*, le graphe (b) représente le neurone bouclé de Frasconi-Gori-Soda avec* $bw_{ii} = 2$ *et la graphe (c) représente le neurone de Poddar-Unnikrishnan avec* $bw_{ii} = 2$ *et* $\alpha = 0.9$*. Pour les trois graphes le paramètre* $b = 0.5$*.*

IV.5. Conclusion

Ce quatrième chapitre constitue la première étape de notre contribution. Il a été dédié à la présentation d'un nouveau réseau de neurones dynamique : le *Réseau Récurrent à base de Fonctions Radiales* (*RRFR*) doté de ses deux mémoires internes (mémoire dynamique et mémoire statique). La mémoire dynamique interne du réseau *RRFR* est obtenue grâce aux représentations *LRGF*, c'est-à-dire des neurones à récurrences locales. Nous avons recensé trois modèles de neurones localement récurrents pouvant être exploités comme mémoire dynamique interne au réseau *RRFR* : deux neurones à retour local de la sortie et un neurone à retour local de l'activation. Nous avons ainsi effectué une étude comparative approfondie des

performances dynamiques de ces trois mémoires. Nous avons également appuyé les différents développements mathématiques par des simulations de performances.

Pour le choix de la mémoire dynamique du *RRFR*, cette étude nous a conduit à éliminer un des deux modèles à retour local de la sortie : le modèle de *Poddar-Unnikrishnan.* Ce neurone localement récurrent possède des performances au mieux identiques au neurone bouclé de *Frasconi-Gori-Soda*. Par contre, les deux autres modèles restant possèdent des comportements dynamiques comparables. La différence entre ces deux modèles concerne le point de retour de la récurrence : l'un se situe avant la non-linéarité du neurone et l'autre se situe après cette non-linéarité. Cette différence structurelle entre les deux neurones récurrents peut avoir des conséquences dans la prise en compte de la non-linéarité des données d'entrée.

Cette étude nous a permis d'identifier deux comportements dynamiques différents pour les deux types de mémoire dynamique du réseau *RRFR* : un comportement de *mémorisation* et un comportement *d'oubli*. Ces deux comportements sont obtenus grâce aux variations des deux paramètres de cette mémoire dynamique : la valeur du poids de l'auto-connexion du neurone récurrent et le paramètre de la fonction d'activation (la sigmoïde). Ces deux paramètres sont donc définis *a priori* par l'expert en fonction du comportement souhaité. La phase d'apprentissage du réseau *RFR* dynamique que nous proposons garde ainsi toute la simplicité de paramétrage des réseaux *RFR* statiques.

La mémoire dynamique basée sur des architectures *LRGF* se prête bien pour le développement d'un outil de surveillance dynamique facile à paramétrer à distance par connexion *TCP/IP* (via le réseau Web). Des récurrences locales au niveau de la couche d'entrée du réseau *RFR* n'ajoutent aucune complexité supplémentaire au réseau de neurones. Nous gagnons ainsi l'aspect dynamique sans pour autant perdre la simplicité d'utilisation des réseaux *RFR*.

Dans le chapitre suivant, nous allons évaluer les performances du réseau *RRFR* avec les deux types de mémoires dynamiques comparables, en l'appliquant à des problèmes type de surveillance dynamique. Nous testerons le réseau *RRFR* sur les trois problématiques d'application des réseaux de neurones temporels en surveillance.

Chapitre V

Evaluation des performances du réseau RRFR

Résumé : *Après avoir présenté au chapitre précédent une étude sur les réseaux LRGF, nous allons tester dans ce chapitre le réseau RRFR avec les deux types de mémoire dynamique à savoir le neurone à retour local de l'activation et le neurone à retour local de la sortie. Les tests et validations du réseau RRFR se feront sur trois types d'applications : Reconnaissance de séquences booléennes et réelles, prédiction de séries temporelles et enfin la reproduction de séries temporelles. Nous verrons que l'architecture LRGF adoptée permet au réseau RRFR d'acquérir des propriétés dynamiques tout en gardant la simplicité et l'efficacité des réseaux RFR. Nous proposons une version améliorée de l'algorithme d'apprentissage des k-moyennes. En effet, la version simple présente quelques faiblesses qui seront mises en évidence à travers quelques tests. La version proposée procure à la phase d'apprentissage une certaine stabilité du résultat avec une plus grande robustesse par rapport à la phase de paramétrage de l'algorithme.*

Abstract : *after having presented in the previous chapter a study on LRGF network, we will test in this fifth chapter the RRBF network with two type of dynamic memory: the local activation feedback and the local output feedback. The tests and the validation of the RRBF network will be made on three types of applications: sequence recognition (Boolean and real), temporal series prediction and finally the temporal series reproduction. We will see that the adopted LRGF architecture allows the RRBF network to have dynamic properties with the simplicity and the efficiency of RBF networks. We also propose an improved version of the k-means clustering algorithm. Indeed, the simple version presents some weaknesses which will be put in evidence through some tests. The proposed k-means algorithm is more stable and has low dependences to the parameter setting phase.*

Chapitre V

Evaluation des performances du réseau RRFR

V.1. Introduction

Le chapitre précédent était consacré à la présentation du réseau *RRFR* avec ses deux mémoires : une mémoire ***dynamique*** basée sur les architectures Localement Récurrente Globalement Feedforward et une mémoire ***statique*** basée sur les neurones gaussiens. Nous avons pour cela effectué une étude comparative approfondie des trois architectures existantes de réseaux de neurones localement récurrents. Une étude formelle de chaque architecture avec des tests de simulation nous a ainsi permis de comparer les performances de ces trois architectures *LRGF*. Parmi ces trois architectures récurrentes, deux possèdent des comportements dynamiques quasi-semblables (comparables) : le neurone à retour local de la sortie et le neurone à retour local de l'activation.

Dans ce cinquième chapitre, nous allons évaluer les performances du réseau *RRFR* avec ses deux mémoires dynamiques sur les trois types d'applications des réseaux de neurones temporels en surveillance dynamique. Nous mettons ainsi en évidence la simplicité d'apprentissage du réseau *RRFR* avec l'efficacité de sa mémoire dynamique sur des problématiques de reconnaissance de séquences temporelles (booléennes et réelles), de prédiction de séquences temporelles pour le pronostic et enfin de reconstitution de séquences temporelles.

Le paramétrage de la mémoire dynamique du réseau *RRFR* est ainsi effectué *a priori*. L'étude faite au chapitre précédent permet de déterminer ces paramètres en fonction du comportement souhaité. Généralement, on définit ces paramètres de telle sorte à avoir une mémoire dynamique la plus longue possible. Par contre, les paramètres de la mémoire statique sont déterminés *a posteriori*, c'est-à-dire à partir des valeurs enregistrées dans une base de données. L'un des intérêts à utiliser les réseaux de neurones artificiels est leur capacité de *généralisation* des connaissances apprises. On comprend alors que si l'on possède une base de données assez importante (en terme de nombre de données), il faudrait mémoriser les données les plus représentatives de cette base et non l'ensemble des données, afin de garantir une bonne généralisation au réseau de neurones. Ce point représente une réelle problématique de

la phase d'apprentissage des réseaux de neurones artificiels. Nous proposons pour cela une version améliorée de la technique des *k-moyennes* qui garantit au réseau *RRFR* de converger vers la zone de bonne généralisation.

Ce chapitre est organisé en trois grandes parties. Chaque partie représente l'évaluation des performances du *RRFR* sur une des trois problématiques d'application des réseaux de neurones artificiels en surveillance dynamique qui sont :

Reconnaissance de séquences temporelles

Le premier type d'application concerne la reconnaissance de séquences temporelles qui peuvent être divisées en deux catégories : séquences temporelles booléennes (pour la surveillance des systèmes à événements discrets) et séquences temporelles réelles (pour la surveillance d'un paramètre de type réel d'un équipement).

Un système à événements discrets est caractérisé par plusieurs séquences. Chaque séquence est à son tour caractérisée par plusieurs événements discrets. Une séquence de bon fonctionnement est donc associée à une succession d'événements et chaque événement se produit à des instants bien connus. Une séquence différente de la séquence d'un bon fonctionnement caractérise une défaillance d'une partie du système. Cette séquence de mauvais fonctionnement peut servir à diagnostiquer l'origine de la défaillance. A travers un exemple simple d'un système à événements discrets, nous verrons que la mémoire dynamique du réseau *RRFR* permet de caractériser toute une séquence temporelle d'événements discrets. Les techniques classiques utilisées pour déterminer les paramètres de la couche des fonctions de base peuvent alors être adoptées pour mémoriser le vecteur séquence.

En surveillance de paramètres de type réel, l'évolution d'un signal capteur représente une succession de paramètres de type réel. Un palier de dégradation d'un signal capteur peut représenter une séquence réelle bien particulière qui sera mémorisée par le réseau *RRFR*. La reconnaissance d'une telle dégradation par le réseau de neurones permet de déclencher une pré-alarme. L'expert peut localiser une dégradation précoce d'un paramètre de l'équipement et prendre ainsi des décisions pour des actions de maintenance préventive. On verra que la mémoire dynamique du réseau *RRFR* est capable de distinguer un palier de dégradation particulier parmi d'autres paliers de dégradation et aussi d'éliminer des pics caractérisant une fausse alarme.

Prédiction de séquences temporelles

Le deuxième type d'application du réseau *RRFR* sera celui de la prédiction temporelle pour le pronostic. Cette problématique représente un vrai défi dans plusieurs domaines (prévisions météorologiques, financières,…). En surveillance, la prédiction permet de connaître les

évolutions futures d'un paramètre de surveillance afin d'anticiper sur les actions à entreprendre.

Reproduction de séquences temporelles

Le dernier type d'application du réseau de neurones *RRFR* en surveillance dynamique concerne la reproduction de séquences. En surveillance industrielle, l'évolution d'un paramètre donné d'un équipement pendant son régime transitoire (soit au démarrage de l'équipement soit pendant un changement de mode de fonctionnement) est une information très utile pour le diagnostic des défaillances. Cette évolution représente alors une signature de bon fonctionnement qui sera mémorisée par le réseau *RRFR*. Le réseau *RRFR* apprendra à reproduire cette signature qui peut être par la suite comparée à l'évolution réelle du paramètre de l'équipement. Le résidu ainsi obtenu permet de caractériser le fonctionnement de l'équipement. Ce type d'application est très complexe avec les réseaux globalement récurrents. Nous verrons comment la récurrence locale de la couche d'entrée permet de transformer le problème de reproduction de séquences en un simple problème d'interpolation.

V.2. La reconnaissance de séquences temporelles

V.2.1. Apprentissage de séquences booléennes d'un Système à Evénements Discrets (SED)

D'après l'étude faite au chapitre précédent sur les comportements dynamiques du neurone récurrent, une occurrence d'un événement externe peut être gardée en mémoire soit temporairement pour un comportement d'oubli, soit indéfiniment pour un comportement de mémorisation. Ces deux types de comportements peuvent être exploités différemment : le comportement d'oubli peut nous renseigner sur l'instant exact d'occurrence de l'événement, à condition de se trouver dans sa plage de mémorisation ($T<300$). Au delà de cette plage, le neurone perd complètement l'effet de l'événement externe. Par contre, le comportement de mémorisation ne donne aucune information concernant l'instant d'occurrence de l'événement (une fois le régime permanent atteint), mais l'on saura « *éternellement* » que l'événement s'est produit. Ces deux comportements offrent deux possibilités d'utilisation différentes, selon la problématique à résoudre. Pour une application d'apprentissage de séquences temporelles, où l'instant d'occurrence d'un événement représente une donnée importante du problème, on exploitera donc le comportement d'oubli du neurone récurrent.

Une séquence S_k est caractérisée par une succession d'événements booléens $\mathbf{Z}_l$ selon un ordre bien précis et à des instants d'occurrence bien précis pour chaque événement. Une séquence S_k a donc un nombre N d'événements ($S_k = \{Z_1, Z_2, ..., Z_l, .., Z_N\}$), et une longueur T qui dépend du nombre N d'événements et de leur instant d'occurrence.

Un événement $\mathbf{Z}_l$ est caractérisé par son instant d'occurrence t_l. En prenant un neurone récurrent *i* avec un comportement d'oubli pour caractériser l'instant d'occurrence de l'événement t_l, on considère alors les conditions d'excitation $\xi_i^l(t)$ ci-dessous :

$$\begin{cases} \xi_i^l(t) = 1 \quad \textit{à l'occurence de l'événement } \mathbf{Z}_l \quad (t = t_l) \\ \xi_i^l(t) = 0 \quad \textit{sinon } (t \neq t_l) \end{cases} \qquad [173]$$

avec la condition initiale de la sortie du neurone bouclé : $y_i(t) = 0 \ \ (t < t_l)$.

On peut démontrer, en exploitant les propriétés de bijection de la fonction d'activation sigmoïde du neurone récurrent, que pour deux instants d'excitation différents (t_a et t_b) on obtient deux évolutions temporelles différentes du neurone récurrent. En d'autres termes :

$$\begin{aligned} &\xi_i^a(t) \neq \xi_i^b(t) \Leftrightarrow y_i^a(t) \neq y_i^b(t) \qquad \forall t > \min(t_a, t_b) \\ &\xi_i^a(t) = \xi_i^b(t) \Leftrightarrow y_i^a(t) = y_i^b(t) \qquad \forall t \end{aligned} \qquad [174]$$

avec $y_i^a(t)$ et $y_i^b(t)$ représentant les sorties du neurone récurrent *i* pour respectivement les entrées $\xi_i^a(t)$ et $\xi_i^b(t)$. Si l'on prend le cas du neurone bouclé, sa sortie pour l'excitation $\xi_i^a(t)$ est $y_i^a(t) = f\left(w_{ii} y_i^a(t-1) + \xi_i^a(t)\right)$ et $y_i^b(t) = f\left(w_{ii} y_i^b(t-1) + \xi_i^b(t)\right)$ pour $\xi_i^b(t)$. Si $\xi_i^a(t) = \xi_i^b(t)$, l'excitation du neurone récurrent est la même. Avec les conditions initiales $y_i^a(t) = 0$ pour $(t < t_a)$ et $y_i^b(t) = 0$ pour $(t < t_b)$, la propriété de bijection de la fonction sigmoïde $f(.)$ fait que l'évolution de la sortie du neurone bouclé soit la même $y_i^a(t) = y_i^b(t)$. Inversement, si l'on considère à un instant donnée t_m que $y_i^a(t_m) = y_i^b(t_m)$, par propriété de bijection de la fonction $f(.)$, on obtient $y_i^a(t_m - 1) = y_i^b(t_m - 1)$. On peut étendre ce raisonnement jusqu'à l'instant d'excitation du neurone bouclé par l'événement qui sera donc le même $\xi_i^a(t) = \xi_i^b(t)$ (voir Figure 67 pour plus de précisions). Le même raisonnement peut être utilisé pour le neurone à retour local de l'activation. On peut donc affirmer que l'évolution temporelle $y_i^a(t)$ d'un neurone récurrent ayant un comportement d'oubli dépend de l'instant exact t_a qui caractérise l'occurrence de l'événement $\mathbf{Z}_a$.

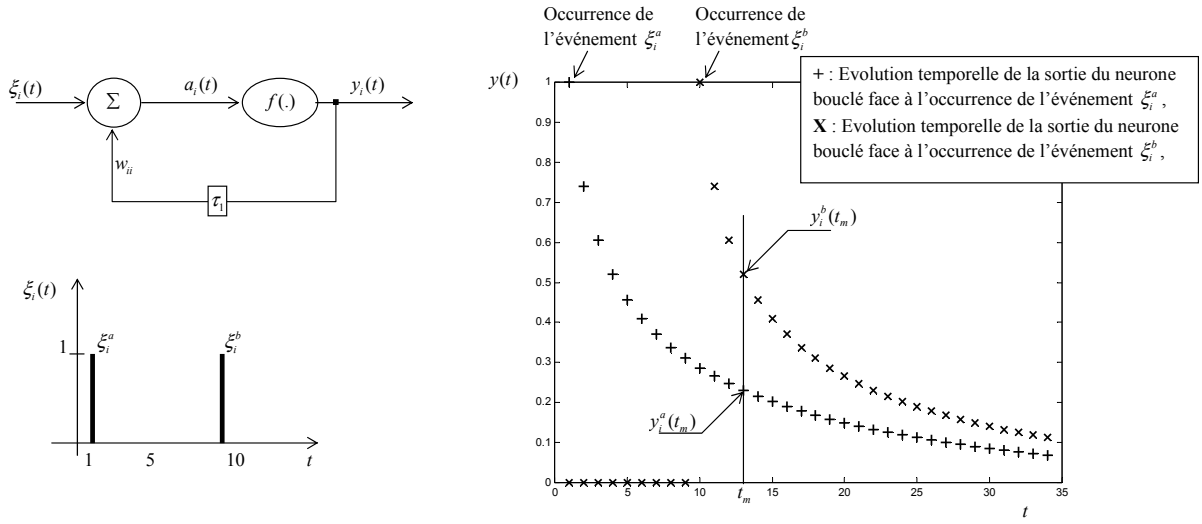


Figure 67. *Evolution du neurone bouclé par rapport à deux événements distincts (se produisant à des instants différents).*

V.2.1.1. Apprentissage de séquences booléennes simples

Nous considérons une séquence booléenne simple comme une séquence S_k où les événements $\mathbf{Z}_l$ ne se produisent qu'une seule fois dans la séquence. Chaque neurone récurrent *i* de la mémoire dynamique ℓ_1 du réseau *RRFR* est dédié à un événement $\mathbf{Z}_i$ (i=l) de la séquence S_k. Le nombre de neurones récurrents de la mémoire dynamique est donc égal au nombre N d'événements de la séquence S_k (Figure 68). La sortie de chaque neurone récurrent *i* est représentative de l'instant d'occurrence de l'événement $\mathbf{Z}_i$.

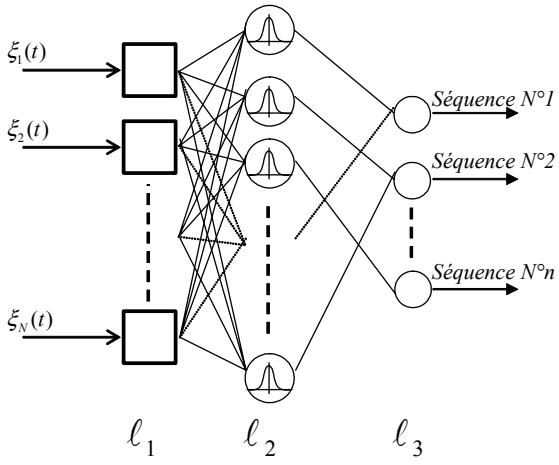


Figure 68. *Utilisation du réseau RRFR pour l'apprentissage de séquence d'événements discrets.*

A la fin de la séquence S_k de longueur T on obtient le vecteur[27] $\mathbf{y}_k = (y_1(T), y_2(T), ..., y_N(T))$. Pour deux séquences différentes, on obtient deux vecteurs différents. Le vecteur $\mathbf{y}_k$ est donc représentatif de la séquence S_k. Ce dernier sera mémorisé en tant que prototype dans la mémoire statique ℓ_2.

Nous allons tester le réseau *RRFR* sur un problème de surveillance d'un système à événements discrets. Le système étudié (Figure 69) représente une petite chaîne de production avec deux machines M_1, M_2. Chaque machine possède un temps de traitement nominal T_{M1} respectivement T_{M2}. Après traitement sur la machine M_1, les pièces sont mises sur une file d'attente. Au bout d'un temps d'attente T_{at1}, celles-ci sont mises sur le convoyeur pour être acheminées vers M_2 pendant un temps de convoyage T_{conv}. Avant d'être traitées sur la machine M_2, les pièces attendent pendant T_{at2} sur la deuxième file d'attente. Le système est muni de six capteurs qui indiquent le passage des pièces à chaque phase. On peut imaginer un système d'identification magnétique sur chaque palette qui permet de suivre ses différents passages dès que celle-ci rentre dans le système.

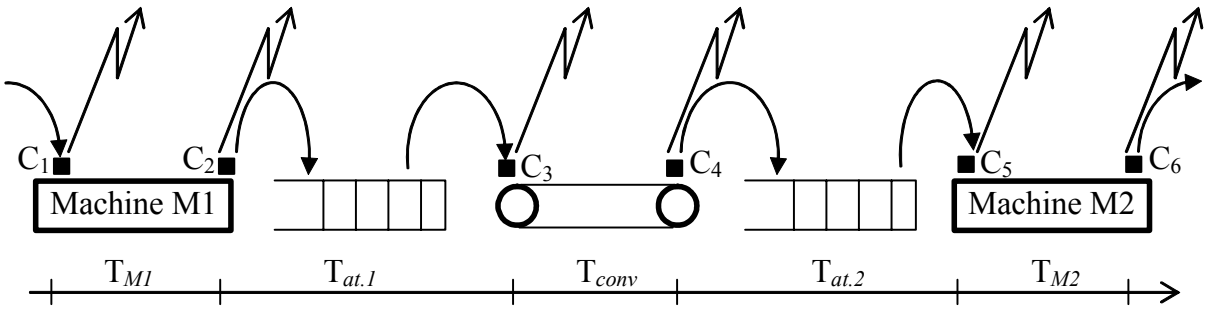
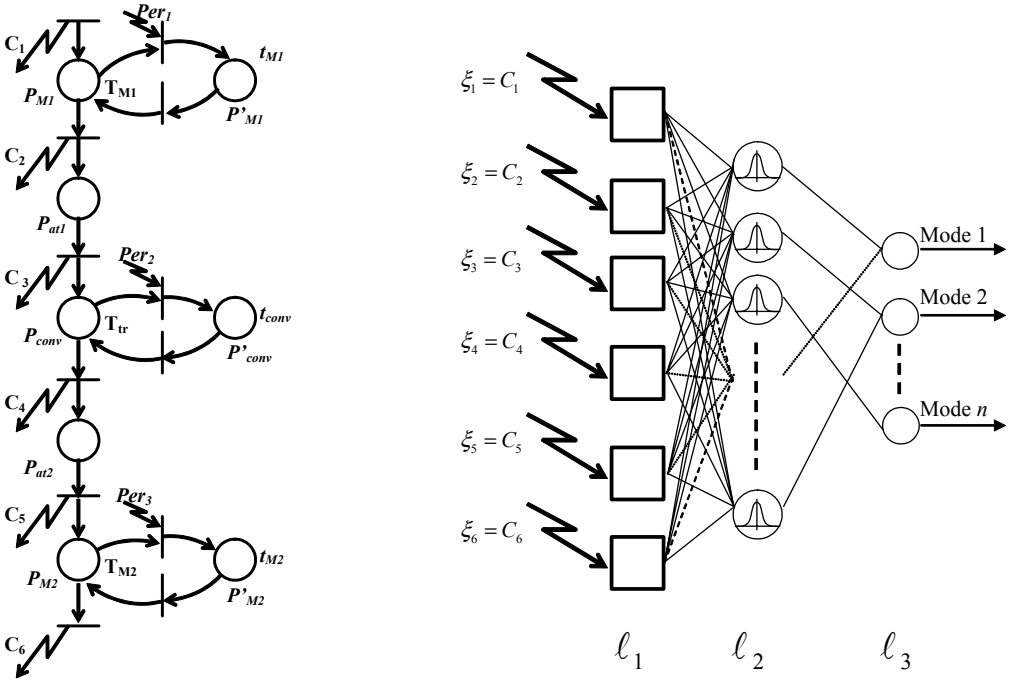


Figure 69. *Exemple d'un système d'événements discrets.*

Chaque machine possède son temps de traitement nominal ainsi qu'un temps nominal de convoyage pour le convoyeur. On peut alors imaginer toutes sortes de perturbations sur le système et tester les capacités du réseau *RRFR* à apprendre la reconnaissance de ces situations de dysfonctionnement. Nous pouvons représenter ce système de production avec ses perturbations par un réseau de Petri temporisé (Daniel, 1995), (Racoceanu *et al.*, 2002) (Figure 70 -a-). Les signaux émis par les passages des différentes transitions représentent les entrées ξ_i du réseau de neurones correspondant aux événements C_i. Chaque séquence S_k sera caractérisée par un vecteur $\mathbf{y}_k$ composé des valeurs de sorties $y_i(t)$ de chaque mémoire dynamique à la fin de la séquence S_k.

[27] En considérant l'origine des temps à l'instant d'occurrence du premier événement de la séquence S_k.



-a- -b-

Figure 70. *a) Représentation du système à événements discrets par un réseau de Petri temporisé, b) Architecture du RRFR pour l'apprentissage des séquences du système.*

Nous avons appliqué le réseau *RRFR* avec les deux types de mémoires dynamiques sur un ensemble de données d'apprentissage de séquences correspondantes à différents modes de fonctionnement du système de la Figure 69. Cet ensemble d'apprentissage comporte une séquence de fonctionnement pour le mode nominal ($N1$), six séquences pour six modes dégradés ($D1,...,D6$) et trois séquences pour trois modes de pannes ($P1, P2, P3$). Nous avons testé les capacités de généralisation du réseau *RRFR* sur des séquences proches de celles qui ont été apprises ainsi que sur des séquences différentes de celles apprises. Les résultats de cette application avec le *RRFR* à mémoire dynamique composée de neurones bouclés (retour local de la sortie) sont présentés sur le Tableau 5 et une mémoire dynamique avec des neurones à retour local de l'activation, sur le Tableau 6. Chaque vecteur caractéristique d'une séquence de la base d'apprentissage est mémorisé par les neurones gaussiens de la mémoire statique à la fin de la séquence. Nous avons appliqué l'algorithme *RCE*[28] pour le calcul des paramètres des neurones gaussiens. L'avantage majeur de cet algorithme d'apprentissage est sa rapidité de convergence.

[28] Décrit au chapitre II

La Figure 71 montre un exemple d'évolution de la mémoire dynamique ℓ_1 du réseau *RRFR* pour la première séquence du Tableau 5. Le vecteur qui sera mémorisé par la mémoire statique (ℓ_2) est composé par les valeurs de sortie des neurones récurrents à la fin de la séquence. Chaque séquence est donc représentée par un prototype avec son champ d'influence. La propriété de généralisation locale du neurone gaussien lui permet de reconnaître des séquences proches de celles apprises.

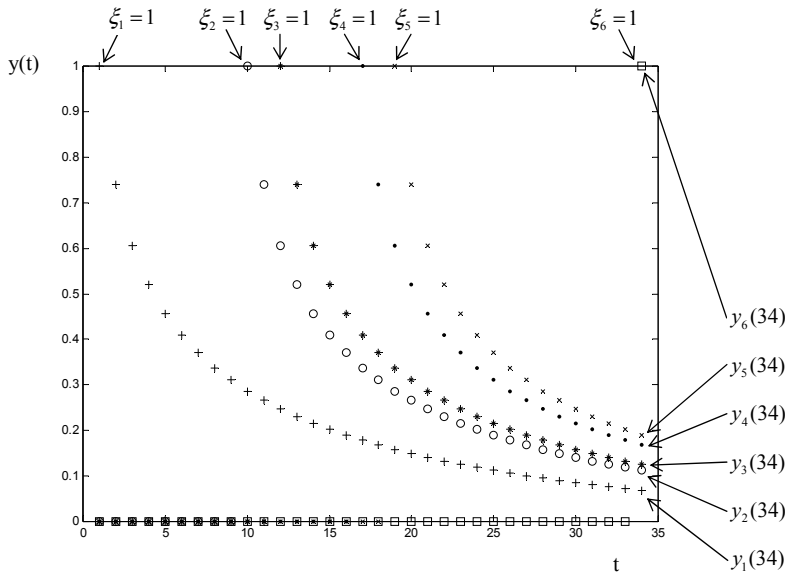


Figure 71. *Représentation de l'évolution temporelle des sorties de chaque neurone récurrent face aux événements associés à chaque neurone. Les valeurs de fin de séquence seront mémorisées par la mémoire statique.*

	Paramètres du système					***Perturbations***			***Réponse du Réseau de neurones***									
	T_{M1}	T_{at1}	T_{conv}	T_{at2}	T_{M2}	t_{M1}	t_{conv}	t_{M2}	N1	D1	D2	D3	D4	D5	D6	P1	P2	P3
Apprentissage	10	2	5	2	15	-	-	-	1	-	-	-	-	-	-	-	-	-
	10	4	5	2	15	-	-	-	-	1	-	-	-	-	-	-	-	-
	10	2	5	4	15	-	-	-	-	-	1	-	-	-	-	-	-	-
	10	4	5	4	15	-	-	-	-	-	-	1	-	-	-	-	-	-
	10	2	5	2	15	5	-	-	-	-	-	-	1	-	-	-	-	-
	10	2	5	2	15	-	-	5	-	-	-	-	-	1	-	-	-	-
	10	2	5	2	15	-	5	-	-	-	-	-	-	-	1	-	-	-
	10	2	5	2	15	15	-	-	-	-	-	-	-	-	-	1	-	-
	10	2	5	2	15	-	-	10	-	-	-	-	-	-	-	-	1	
	10	2	5	2	15	-	10	-	-	-	-	-	-	-	-	-	-	1
Test et validation sur des modes connus	10	2	5	2	15	1	-	-	0.92	0.38	-	-	0.43	-	-	-	-	-
	10	2	5	2	15	4	-	-	0.32	0.32	-	-	0.95	-	-	-	-	-
	10	2	5	2	15	6	-	-	-	-	-	-	0.96	-	-	0.39	-	-
	10	2	5	2	15	11	-	-	-	-	-	-	0.32	-	-	0.87	-	-
	10	2	5	2	15	14	-	-	-	-	-	-	-	-	-	0.99	-	-
	10	2	5	2	15	-	1	-	0.53	0.55	-	-	-	-	-	-	-	-
	10	2	5	2	15	-	4	-	-	-	-	-	-	-	0.9	-	-	-
	10	2	5	2	15	-	6	-	-	-	-	-	-	-	0.91	-	-	0.48
	10	2	5	2	15	-	8	-	-	-	-	-	-	-	0.47	-	-	0.85
	10	2	5	2	15	-	9	-	-	-	-	-	-	-	-	-	-	0.96
	10	2	5	2	15	-	-	2	-	-	-	-	-	0.48	-	-	-	-
	10	2	5	2	15	-	-	4	-	-	-	-	-	0.93	-	-	-	-
	10	2	5	2	15	-	-	6	-	-	-	-	-	0.94	-	-	0.48	-
	10	2	5	2	15	-	-	8	-	-	-	-	-	0.61	-	-	0.85	-
	10	3	5	2	15	-	-	-	0.72	0.75	-	-	0.35	-	-	-	-	-
	10	3	5	3	15	-	-	-	-	0.39	0.36	-	-	-	-	-	-	-
	10	3	5	4	15	-	-	-	-	-	0.72	0.75	-	-	-	-	-	-
Détection de nouveaux modes	10	2	5	2	15	-10	-	-	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	-	-	-15	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	-10	-	-15	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	-10	+10	-	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	-10	+25	-15	-	-	-	-	-	-	-	-	-	-
	10	4	5	0	15	-	-	-	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	15	5	-	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	15	10	10	-	-	-	-	-	-	-	-	0.39	-
	10	2	5	2	15	5	5	5	-	-	-	-	-	0.45	-	-	0.37	-

Tableau 5. *Résultats du test du RRFR sur l'apprentissage des séquences simples du système à événements discrets. La mémoire dynamique du réseau est constituée par les neurones bouclés (retour local de la sortie) qui ont un comportement d'oubli avec* $b = 0.5, w = 1.9$. *Les paramètres des neurones gaussiens ont été calculés selon l'algorithme RCE avec* $\theta = 0,3$.

	Paramètres du système					Perturbations			Réponse du Réseau de neurones									
	T_{M1}	T_{at1}	T_{conv}	T_{at2}	T_{M2}	t_{M1}	t_{conv}	t_{M2}	N1	D1	D2	D3	D4	D5	D6	P1	P2	P3
Apprentissage	10	2	5	2	15	-	-	-	1	-	-	-	-	-	-	-	-	-
	10	4	5	2	15	-	-	-	-	1	-	-	-	-	-	-	-	-
	10	2	5	4	15	-	-	-	-	-	1	-	-	-	-	-	-	-
	10	4	5	4	15	-	-	-	-	-	-	1	-	-	-	-	-	-
	10	2	5	2	15	5	-	-	-	-	-	-	1	-	-	-	-	-
	10	2	5	2	15	-	-	5	-	-	-	-	-	1	-	-	-	-
	10	2	5	2	15	-	5	-	-	-	-	-	-	-	1	-	-	-
	10	2	5	2	15	15	-	-	-	-	-	-	-	-	-	1	-	-
	10	2	5	2	15	-	-	10	-	-	-	-	-	-	-	-	1	
	10	2	5	2	15	-	10	-	-	-	-	-	-	-	-	-	-	1
Test et validation sur des modes connus	10	2	5	2	15	1	-	-	0.87	0.45	-	-	-	-	-	-	-	-
	10	2	5	2	15	4	-	-	-	0.3	-	-	0.92	-	-	-	-	-
	10	2	5	2	15	6	-	-	-	-	-	-	0.92	-	-	0.38	-	-
	10	2	5	2	15	11	-	-	-	-	-	-	-	-	-	0.83	-	-
	10	2	5	2	15	14	-	-	-	-	-	-	-	-	-	0.98	-	-
	10	2	5	2	15	-	1	-	0.62	0.62	-	-	-	-	-	-	-	-
	10	2	5	2	15	-	4	-	-	-	-	-	-	-	0.8	-	-	-
	10	2	5	2	15	-	6	-	-	-	-	-	-	-	0.81	-	-	0.46
	10	2	5	2	15	-	8	-	-	-	-	-	-	-	-	-	-	0.82
	10	2	5	2	15	-	9	-	-	-	-	-	-	-	-	-	-	0.95
	10	2	5	2	15	-	-	2	-	-	0.45	-	-	0.33	-	-	-	-
	10	2	5	2	15	-	-	4	-	-	-	-	-	0.89	-	-	-	-
	10	2	5	2	15	-	-	6	-	-	-	-	-	0.89	-	-	0.46	-
	10	2	5	2	15	-	-	8	-	-	-	-	-	0.35	-	-	0.82	-
	10	3	5	2	15	-	-	-	0.73	0.74	-	-	-	-	-	-	-	-
	10	3	5	3	15	-	-	-	-	0.69	0.69	-	-	-	-	-	-	-
	10	3	5	4	15	-	-	-	-	-	0.73	0.74	-	-	-		-	-
Détection de nouveaux modes	10	2	5	2	15	-10	-	-	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	-	-	-15	-	-	-	-	-	-	-	0.74	-	-
	10	2	5	2	15	-10	-	-15	-	-	-	-	-	-	-	-	0.49	-
	10	2	5	2	15	-10	+10	-	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	-10	+25	-15	-	-	-	-	-	-	-	-	-	-
	10	4	5	0	15	-	-	-	-	-	-	-	-	-	-	-	0.66	-
	10	2	5	2	15	15	5	-	-	-	-	-	-	-	-	0.33	-	-
	10	2	5	2	15	15	10	10	-	-	-	-	-	-	-	-	-	-
	10	2	5	2	15	5	5	5	-	-	-	-	-	-	-	-	0.48	-

Tableau 6. *Résultats du test du RRFR sur l'apprentissage des séquences simples du système à événements discrets. La mémoire dynamique du réseau est constituée par les neurones à retour local de l'activation qui ont un comportement d'oubli avec* $b = 0.5, w = 0.99$. *Les paramètres des neurones gaussiens ont été calculés selon l'algorithme RCE avec* $\theta = 0,3$.

Après les résultats obtenus par les deux types de mémoires dynamiques sur le problème d'apprentissage de séquences booléennes simples, on peut faire un certain nombre de remarques:

- le réseau *RRFR* a appris correctement les séquences de la base d'apprentissage avec les deux types de mémoire dynamique,
- le test du réseau, sur l'ensemble des séquences proches de celles apprises de la base d'apprentissage, a été satisfaisant. Toutes les séquences de la base de test ont été correctement reconnues par le réseau.
- la différence entre les deux types de mémoires dynamiques (retour local de la sortie et retour local de l'activation) se situe au niveau du test sur la reconnaissance de nouvelles séquences. Le réseau *RRFR* avec neurones à retour local de l'activation a tendance à donner de fausses réponses, contrairement au réseau avec neurone bouclé. Ce dernier ne donne pas de réponses pour les séquences qui sont différentes de celles apprises et est plus apte à détecter des séquences différentes de celles rencontrées lors de l'apprentissage.

V.2.1.2. Apprentissage de séquences booléennes complexes

Contrairement aux séquences simples, dans une séquence complexe, un événement peut apparaître plusieurs fois. Cette caractéristique représente une limite du réseau *RRFR*. En effet, le réseau *RRFR* est incapable d'apprendre des séquences complexes car, à chaque nouvelle occurrence d'un événement, le réseau a tendance à oublier les précédentes occurrences de l'événement même. La Figure 72 schématise clairement ce phénomène d'oubli dû à l'apprentissage d'une séquence complexe. Nous considérons quatre événements ***A***, ***B***, ***C*** et ***D***, un neurone récurrent par événement. En essayant de faire apprendre au réseau la séquence complexe ***ABCADB***, on se rend compte que le vecteur caractéristique de cette séquence (prototype) est identique à celui de la séquence ***CADB***. Les deux premières occurrences des événements ***A*** et ***B*** ont été oubliés par leur deuxième occurrence. Le réseau de neurones ne fait aucune différence entre la séquence complexe ***ABCADB*** et simple ***CADB***. Le réseau *RRFR* est donc incapable d'apprendre des séquences complexes.

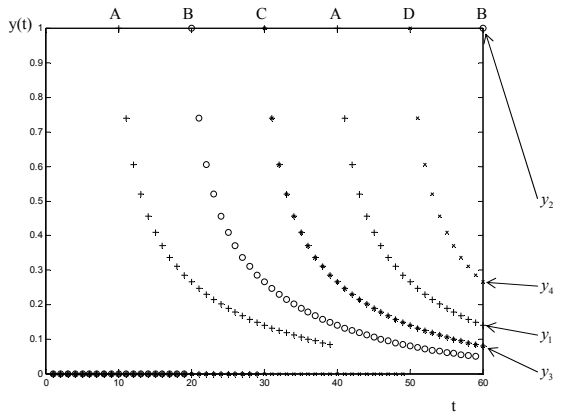


Figure 72. *Séquence ABCADB, prototype de fin de séquence identique à celui de la séquence CADB. Impossibilité d'apprentissage d'une séquence complexe.*

V.2.2. Reconnaissance de séquences réelles

V.2.2.1. Détection précoce d'un palier de dégradation et élimination des fausses alarmes

Contrairement aux séquences booléennes caractérisées par une succession d'événements discrets et finis, une séquence réelle est plutôt vue comme une série temporelle à paramètres réels. Dans une séquence booléenne, on s'intéresse plutôt à l'occurrence ou pas d'un événement avec son instant d'occurrence. Les données d'entrée du réseau sont donc de type Booléen (tout ou rien), alors que dans une séquence réelle, on s'intéresse plutôt à une succession de valeurs de type réel à chaque période d'échantillonnage (séquence réelle discrétisée) :

$$S_k = \{u(t), u(t+1), u(t+2), ..., u(t+n)\} \mid n \in \mathbb{N}, u(t) \in \mathbb{R} \qquad [175]$$

Comme pour l'apprentissage des séquences booléennes, la mémoire dynamique du réseau *RRFR* devra caractériser cette séquence réelle discrétisée par un vecteur représentatif. Ce dernier sera mémorisé par la mémoire statique. Nous allons dans cette partie, mettre en évidence les capacités du réseau *RRFR* à apprendre à distinguer entre des pics de fausses alarmes et des paliers de dégradation. Un palier de dégradation est considéré comme une série temporelle de type réel qui sera caractérisée par la mémoire dynamique. Les deux types de mémoire dynamique seront comparés. Nous présentons sur la Figure 73 un exemple de plusieurs paliers de dégradation.

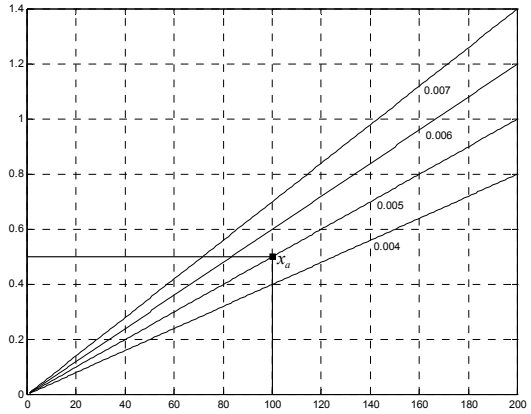


Figure 73. *Apprentissage à la reconnaissance d'un palier de dégradation.*

Afin de mesurer l'effet de la mémoire dynamique sur la caractérisation du vecteur qui sera mémorisé par la mémoire statique, on considère un seul point x_a du palier d'apprentissage (Figure 73). On testera alors les capacités du réseau *RRFR* à distinguer entre ce point appris et d'autres points faisant partie des paliers de test. La reconnaissance du palier se fera à travers la sortie de la fonction gaussienne mémorisant le prototype. Cette sortie dépend du calcul de la distance euclidienne entre ce prototype mémorisé et le vecteur de sortie de la mémoire dynamique. Ce calcul se fera tout au long de la présentation des points du palier de dégradation à l'entrée du réseau. La réussite d'une telle classification dynamique est donc tributaire de la qualité du vecteur de sortie de la mémoire dynamique. Un bon vecteur caractérisant une séquence réelle S_k est un vecteur dont la distance euclidienne doit être faible pour des séquences proches de la séquence apprise S_k, et a tendance à croître pour des séquences différentes de S_k. Nous allons de ce fait établir des tests de comparaison entre les deux types de mémoires dynamiques du réseau *RRFR*. Nous essayerons de voir celle qui donnera un bon vecteur caractéristique d'une séquence réelle.

Les tableaux 7 et 8 montrent les résultats obtenus du calcul des distances euclidiennes entre le point mémorisé pris de la droite de pente 0.005 (Figure 73) et d'autres droites avec différentes pentes. Le calcul des distances se fait en continu, c'est-à-dire au fur et à mesure que les points de la droite sont présentés à l'entrée du réseau de neurones, et ceci pour différentes longueurs de la mémoire dynamique. Cette longueur varie entre une mémoire ayant qu'un seul neurone bouclé (ou un neurone à retour local de l'activation) à 1 neurone linéaire et une cascade de 6 neurones bouclés (ou 1 neurone linéaire et une cascade de 6 neurones à retour local de l'activation) (Figure 74). Pour chaque dimension de la mémoire et chaque valeur de la pente, nous donnons la distance minimum calculée tout au long de la présentation de la droite à l'entrée du réseau de neurones.

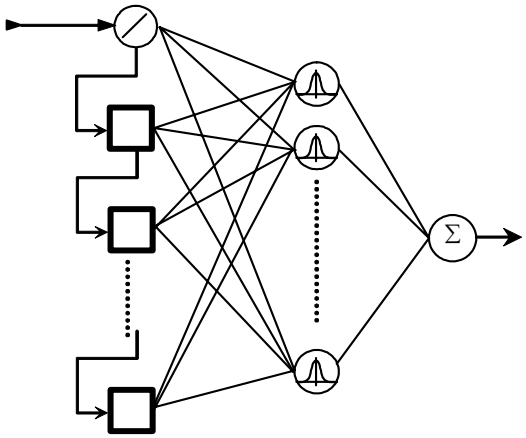


Figure 74. *Réseau RRFR avec mémoire dynamique à un neurone linéaire et une cascade de neurones récurrents.*

	Valeurs des pentes des paliers de dégradation									
	0.002	**0.003**	**0.004**	**0.005**	**0.006**	**0.007**	**0.008**	**0.009**	**0.01**	**0.1**
1NB	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
1NL + 1NB	*0.0102*	*0.0001*	*0*	*0*	*0*	*0.0002*	*0.0006*	*0.0013*	*0.0021*	*0.0699*
1NL + 2NB	*0.0106*	*0.0008*	*0.0003*	*0*	*0.0006*	*0.0023*	*0.0050*	*0.0084*	*0.0122*	*0.1231*
1NL + 3NB	*0.0159*	*0.0057*	*0.0019*	*0*	*0.0017*	*0.0059*	*0.0116*	*0.0180*	*0.0246*	*0.1520*
1NL + 4NB	*0.0343*	*0.0173*	*0.0036*	*0*	*0.0025*	*0.0083*	*0.0156*	*0.0234*	*0.0311*	*0.1623*
1NL + 5NB	*0.0596*	*0.0210*	*0.0042*	*0*	*0.0028*	*0.0090*	*0.0168*	*0.0249*	*0.0329*	*0.1646*
1NL + 6NB	*0.0619*	*0.0219*	*0.0044*	*0*	*0.0028*	*0.0092*	*0 .0170*	*0.0252*	*0.0332*	*0.1649*

Tableau 7. *Calcul de la distance euclidienne entre le vecteur de sortie de la mémoire dynamique et le prototype mémorisé correspondant à une droite d'une pente égale à 0.005. Comparaison de plusieurs dimensions de mémoires dynamiques constituées d'un neurone linéaire (NL) et de plusieurs neurones bouclés (NB). Les paramètres de la fonction d'activation sigmoïde sont b = 0.05 avec bw = 1.99.*

	Valeurs des pentes des paliers de dégradation									
	0.002	**0.003**	**0.004**	**0.005**	**0.006**	**0.007**	**0.008**	**0.009**	**0.01**	**0.1**
1RA	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0.0003*
1NL + 1RA	*0.0164*	*0.0060*	*0.0013*	*0*	*0.0009*	*0.0033*	*0.0066*	*0.0105*	*0.0149*	*0.1584*
1NL + 2RA	*0.0393*	*0.0140*	*0.0029*	*0*	*0.0022*	*0.0075*	*0.0148*	*0.0234*	*0.0327*	*0.2540*
1NL + 3RA	*0.0560*	*0.0198*	*0.0041*	*0*	*0.0030*	*0.0103*	*0.0201*	*0.0315*	*0.0436*	*0.2882*
1NL + 4RA	*0.0649*	*0.0228*	*0.0047*	*0*	*0.0033*	*0.0114*	*0.0223*	*0.0346*	*0.0476*	*0.2965*
1NL + 5RA	*0.0686*	*0.0239*	*0.0049*	*0*	*0.0034*	*0.0117*	*0.0228*	*0.0353*	*0.0485*	*0.2980*
1NL + 6RA	*0.0698*	*0.0242*	*0.0049*	*0*	*0.0034*	*0.0118*	*0.0229*	*0.0354*	*0.0486*	*0.2982*

Tableau 8. *Calcul de la distance euclidienne entre le vecteur de sortie de la mémoire dynamique, et le prototype mémorisé correspondant à une droite d'une pente égale à 0.005. Comparaison de plusieurs dimensions de mémoires dynamiques constituées d'un neurone linéaire (NL) et de plusieurs neurones à retour local de l'activation (RA). Les paramètres de la fonction d'activation sigmoïde sont b = 0.05 avec w = 0.99.*

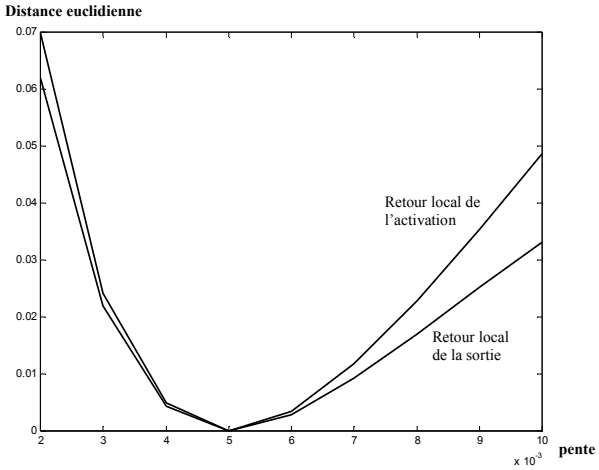


Figure 75. *Comparaison entre la sortie de la mémoire dynamique avec neurones bouclés et celle avec neurone à retour local de l'activation. Les deux mémoires dynamiques sont constituées d'un neurone linéaire et une cascade de six neurones récurrents. En abscisse nous avons les pentes des droites présentées au réseau et en ordonnée, la distance euclidienne avec le prototype de la droite de pente égale à 0.005.*

La Figure 75 résume les résultats des calculs de distance obtenus avec les deux types de mémoires dynamiques (neurone bouclé et neurone à retour local de l'activation). Pour ces deux types de mémoires, la distance euclidienne est bien proche de zéro pour les droites proches de celle apprise et tend à augmenter pour celles qui s'éloignent de la droite apprise. La mémoire dynamique du réseau est donc bien capable de caractériser une séquence temporelle réelle. Ce vecteur caractéristique est ensuite mémorisé par la mémoire statique.

Le dimensionnement de la mémoire dynamique est un paramètre important pour la caractérisation de la séquence réelle (Zemouri *et al.,* 2003 -b-). Les deux tableaux précédents montrent que plus on augmente la cascade de neurones bouclés, plus le vecteur de sortie de la mémoire dynamique est mieux représentatif de la séquence réelle. Néanmoins, on commence à obtenir de bons résultats discriminatoires à partir d'une mémoire dynamique à un neurone linéaire et un neurone récurrent. Les résultats obtenus avec les deux types de mémoire dynamique sont quasiment identiques (voir Figure 75). On peut ainsi détecter une dégradation, aussi minime soit elle, alors que l'équipement est toujours dans sa zone de bon fonctionnement.

Après avoir testé les capacités de la mémoire dynamique à caractériser un palier de dégradation, nous allons montrer que le neurone récurrent peut jouer le rôle d'un filtre passe bas. En d'autres termes, la mémoire dynamique du réseau RRFR est capable de réagir différemment face à un pic de fausse alarme et à un palier de dégradation (Zemouri *et al.,* 2002 -c-). Comme pour le cas précédent du palier de dégradation, nous allons faire des tests de calcul de distance euclidienne entre le point mémorisé précédemment, en l'occurrence le point de la droite de pente 0.005 (palier de dégradation de la Figure 73) et le même point, mais cette fois-ci faisant partie d'un pic de fausse alarme (voir Figure 76). Le tableau ci-

dessous caractérise ce calcul de distance pour chacune des deux mémoires dynamiques (retour local de la sortie et retour local de l'activation).

	Type de mémoire dynamique	
	Retour Local de la Sortie	**Retour Local Activation**
1 N Récurrent	*0.0821*	*0.1780*
1NL + 1NR	*0.0821*	*0.1780*
1NL + 2NR	*0.1361*	*0.2754*
1NL + 3NR	*0.1650*	*0.3096*
1NL + 4NR	*0.1753*	*0.3179*
1NL + 5NR	*0.1776*	*0.3194*
1NL + 6NR	*0.1780*	*0.3196*

Tableau 9. *Calcul de distance euclidienne entre un palier de dégradation et un pic de fausse alarme (NL = Neurone Linéaire, NR = Neurone Récurrent).*

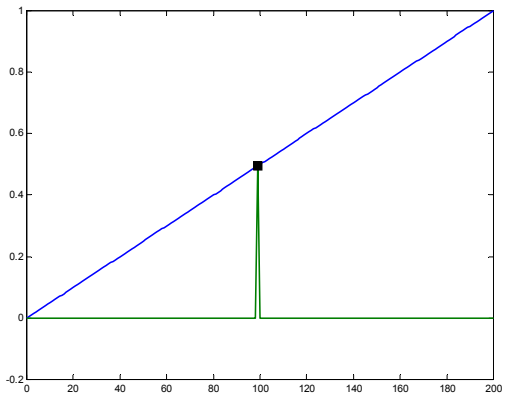


Figure 76. *Dissociation entre un changement brusque et un palier de dégradation.*

La réponse du neurone récurrent est bien différente pour le même point physique appartenant à un palier de dégradation et à un pic de fausse alarme. Le neurone récurrent agit donc comme un filtre passe bas. Il élimine implicitement les hautes fréquences. On peut formaliser ce comportement en étudiant sa sortie pour un changement brusque et pour un palier de dégradation. Les développements mathématiques que nous allons donner concernent le neurone bouclé. L'analogie avec le neurone à retour local de l'activation peut ainsi être faite facilement.

Soit $\tilde{y}$ le régime permanent de la sortie du neurone bouclé correspondant au régime permanent du signal d'entrée $\tilde{\xi}$. On définit un changement brusque du signal d'entrée par un passage de $\tilde{\xi}$ à ξ^* en un laps de temps relativement nul. On peut exprimer ce changement par l'expression suivante :

$$\left|\frac{\xi^* - \tilde{\xi}}{\Delta t}\right| \approx +\infty \qquad [176]$$

Soit la réponse du neurone bouclé pour un pic de changement brusque y^* ([176]) :

$$y^* = \frac{1-\exp(-b(w_{ii}\tilde{y} + \xi^*))}{1+\exp(-b(w_{ii}\tilde{y} + \xi^*))} \qquad [177]$$

A la différence d'un changement brusque, on peut définir un palier de dégradation par l'existence d'une valeur intermédiaire ξ^ε entre $\tilde{\xi}$ et ξ^* tel que :

$$\tilde{\xi} < \xi^\varepsilon < \xi^* \text{ ou bien } \exists \eta > 0 \ / \ \left|\frac{\xi^* - \tilde{\xi}}{\Delta t}\right| = \eta \qquad [178]$$

Pour étudier la sortie du neurone bouclé face à un pic de fausse alarme [176] et un palier de dégradation [178], on compare l'expression [177] et la sortie y^{**} pour ξ^* de la relation [178] (Figure 77).

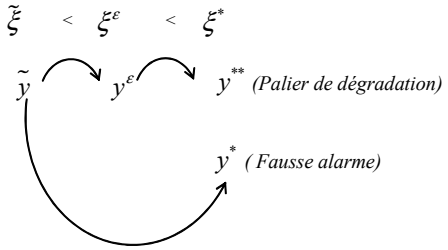


Figure 77. *Principe de calcul des sorties du neurone bouclé face à un pic de fausse alarme et un palier de dégradation*

Pour la valeur intermédiaire ξ^ε du signal d'entrée, la sortie du neurone bouclé présente la forme suivante :

$$y^\varepsilon = \frac{1-\exp(-b(w_{ii}\tilde{y} + \xi^\varepsilon))}{1+\exp(-b(w_{ii}\tilde{y} + \xi^\varepsilon))} \qquad [179]$$

Comme la fonction sigmoïde est strictement croissante et que $w_{ii} > 0$, on obtient la relation suivante :

$$y^\varepsilon > \tilde{y} \qquad [180]$$

La sortie du neurone bouclé pour la valeur ξ^* devient par conséquent :

$$y^{**} = \frac{1-\exp(-b(w_{ii}y^{\varepsilon} + \xi^*))}{1+\exp(-b(w_{ii}y^{\varepsilon} + \xi^*))} \qquad [181]$$

Si l'on considère que $w_{ii} > 0$, on obtient par la suite :

$$w_{ii}y^{\varepsilon} + \xi^* > w_{ii}\tilde{y} + \xi^* \qquad [182]$$

donc :

$$y^{**} > y^* \qquad [183]$$

La sortie du neurone bouclé, de fonction d'activation sigmoïde, est donc différente dans le cas où on aurait un changement brusque du signal d'entrée et dans le cas où il s'agirait d'un palier de dégradation. La réponse du neurone bouclé est plus importante dans le deuxième cas.

Nous avons validé cette propriété du neurone récurrent sur le filtrage des bruits d'acquisition de la vitesse de rotation d'un moteur électrique (Figure 78). Nous avons provoqué des perturbations au niveau du capteur de vitesse d'une part, et des frottements continus au niveau de l'axe de rotation du moteur. La Figure 79 montre d'un côté l'acquisition de vitesse (entrée du neurone récurrent) et d'un autre côté le filtrage effectué par le neurone récurrent (sa sortie).

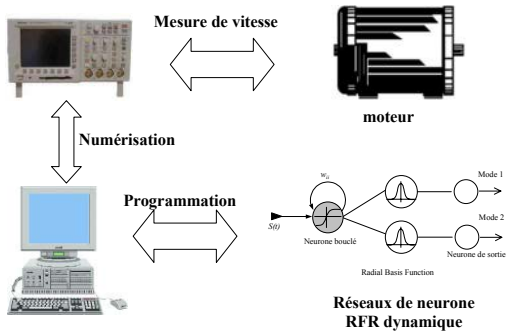
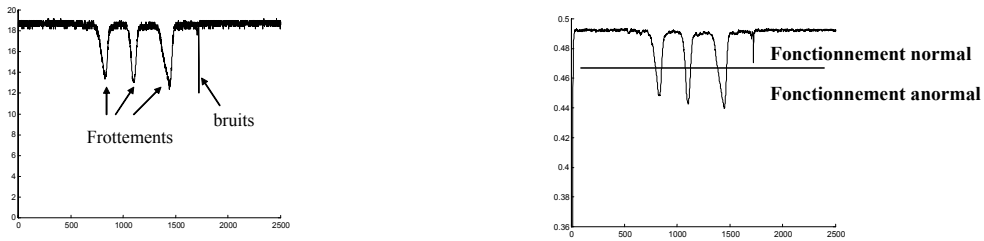


Figure 78. *Schéma d'application de la surveillance d'un moteur.*



- a - *mesure de vitesse* - b - *réponse du neurone bouclé*

Figure 79. *Mesure de la vitesse de rotation et réponse du neurone bouclé avec les deux types de perturbations : frottements et bruits de mesures.*

V.2.2.2. Détection du type de collision d'un bras de robot

Nous présentons une autre application qui met en évidence l'apport de la mémoire dynamique du *RRFR* sur la reconnaissance dynamique de signaux capteurs (Zemouri *et al.,* 2003 -a-). Cette application a été construite à partir des données réelles du benchmark « *Robot Execution Failures*» disponibles sur le serveur de l'université de Californie[29] (Camarinha-Matos *et al.,* 1996). Le but de cette application est de reconnaître le type de collision d'un bras de robot à partir d'une acquisition de signaux capteurs (Figure 80). Le bras du robot est muni de trois capteurs de force (F_x, F_y, F_z). Les réponses données par ces trois capteurs nous renseignerons sur l'existence ou pas de contact brusque du bras du robot avec un obstacle. Quatre types de collisions sont susceptibles de se produire (Figure 81) : *collision frontale, collision par derrière, collision à gauche et collision à droite*. Chaque type de collision est caractérisé par une évolution temporelle des trois signaux de mesures. La Figure 82 présente un échantillon de mesure pour les quatre types de collisions. Les réponses des trois capteurs de force constituent les trois entrées du réseau *RRFR.* Après une phase d'apprentissage, le réseau *RRFR* devra reconnaître le type de collision à partir des entrées des trois capteurs (Tableau 10).

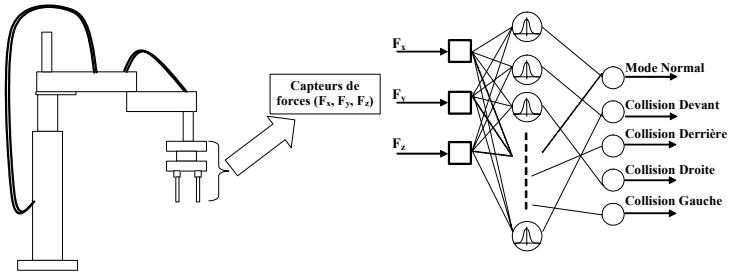


Figure 80. *Application du réseau RRFR pour la surveillance d'un bras de robot. Les sorties des trois capteurs de force constituent les entrées du réseau RRFR. L'apprentissage permet de définir le nombre ainsi que les paramètres des neurones gaussiens.*

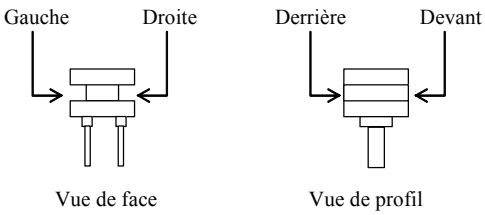
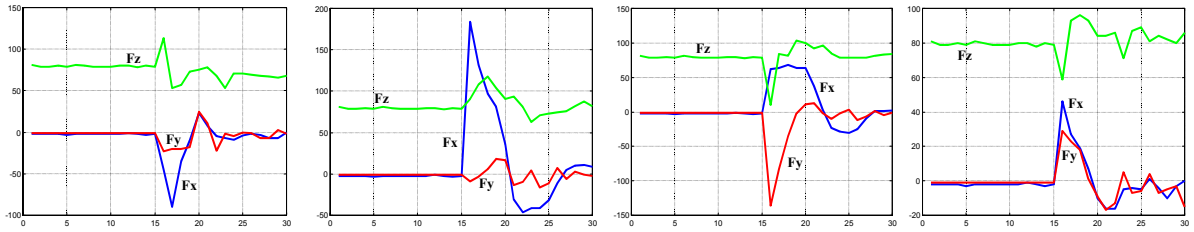


Figure 81. *Différents types de collisions possibles du robot lors de l'exécution d'une tache : collision frontale, par derrière, à gauche ou par la droite.*

[29] *The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science*



-a- Collision frontale | -b- Collision par derrière | -c- Collision à droite | -d- Collision à gauche

Figure 82. *Réponses des capteurs de force (F_x, F_y et F_z) pour chaque type de collision.*

Chaque type de collision est donc caractérisé par une signature des trois mesures de force. La base de données de cette application contient 47 échantillons qui sont répartis avec la distribution suivante : 43% mode normal, 13% mode collision frontale, 15% mode collision par derrière, 11% mode collision à droite et 19% collision à gauche. Chaque échantillon est composé de 15 valeurs (mesures) effectuées juste au moment de la collision du bras du robot avec un obstacle. Nous avons pris un seul échantillon par type de collision pour le processus d'apprentissage du réseau *RRFR*. Le reste des échantillons sera utilisé pour le test. La dynamique de chaque signal est prise en compte par la mémoire dynamique du *RRFR*. Nous avons donc voulu comparer les performances du réseau *RRFR* avec les deux types de mémoires dynamiques et, pour mettre en évidence l'apport de la mémoire dynamique, nous avons également comparé le *RRFR* avec le réseau *RFR* sans mémoire dynamique (statique). La taille ainsi que les paramètres de la mémoire statique (neurones gaussiens) ont été déterminés par l'algorithme d'apprentissage *DDA*[30] qui permet d'établir un traitement incrémental. Les paramètres du réseau sont ainsi déterminés tout au long de la présentation des 15 valeurs de l'échantillon d'apprentissage. Les seuils utilisés pour le calcul sont $\theta^+ = 0,4$ et $\theta^- = 0,1$. Concernant la phase de reconnaissance, nous avons utilisé une couche d'intégration[31] qui permet de prendre en compte les réponses des neurones gaussiens par rapport à toute la séquence. Cette couche d'intégration effectue une sommation des sorties de chaque neurone gaussien tout au long de la présentation du signal au réseau de neurones :

$$\phi_i = \sum_{t=1}^{15} \phi_i(t) \qquad [184]$$

avec $\phi_i(t)$ la sortie du $i^{ème}$ neurone gaussien du réseau *RRFR* à l'instant *t*. La réponse du réseau est donnée par le principe du maximum d'appartenance. Les résultats des tests obtenus par le réseau *RRFR* avec les deux types de mémoires dynamiques et du *RFR* sans mémoire dynamique (statique) sont donnés par le tableau ci-dessous :

[30] *Dynamic Decay Adjustment présenté au chapitre II*
[31] *Voir réseau TDNN et TDRBF au Chapitre III*

Types de collisions	**Taux de reconnaissance**		
	Mémoire dynamique avec Neurone bouclé	*Mémoire dynamique avec neurone à retour de l'activation*	*Absence de mémoire dynamique (RFR classique)*
Collision Frontale	60 %	40 %	20 %
Collision par derrière	16,5 %	50 %	0 %
Collision à droite	75 %	50 %	0 %
Collision à gauche	25 %	50 %	0 %

Tableau 10. *Résultats obtenus par les trois réseaux (RRFR avec les deux types de mémoire, et RFR).*

Chaque colonne du Tableau 10 représente les taux de reconnaissance obtenus par les trois réseaux testés (RRFR avec les deux types de mémoires dynamiques et le réseau RFR statique). Le pourcentage est donné par rapport aux échantillons de test pour chaque type de collision. Les performances du *RFR* statique sont nettement inférieures au *RRFR*. L'absence de mémoire dynamique le rend incapable de prendre en compte l'évolution des signaux. Les prototypes mémorisés par la mémoire statique du *RFR* sont complètement indépendants de l'évolution dynamique des signaux d'entrée. Par contre, grâce au traitement effectué par la mémoire dynamique du réseau *RRFR,* les prototypes mémorisés par la mémoire statique dépendent de l'évolution du signal. Le réseau est ainsi capable de caractériser la dynamique de chaque type de collision. Une moyenne globale des taux de reconnaissance obtenus par le *RRFR* est de *44%* avec une mémoire dynamique à neurones bouclés, *47,5%* avec mémoire dynamique à retour local de l'activation contre *5%* avec le réseau *RFR* statique. Les temps d'apprentissage et de traitement sont quasiment identiques pour les trois réseaux (≈ 0.04 secondes[32]).

V.3. La prédiction temporelle pour le pronostic

Après avoir testé les performances du *RRFR* sur une problématique d'apprentissage et de reconnaissance de séquences temporelles (booléennes et réelles), nous allons tester le réseau *RRFR* sur un autre type d'application où la dynamique des données d'entrée est tout aussi importante que précédemment. Cette application concerne la prédiction de séries temporelles pour des applications de pronostic industriel. Nous allons tester les propriétés dynamiques du réseau *RRFR* sur deux types de données temporelles : la série temporelle *MackeyGlass* et une

[32] *Traitement effectué sur un processeur de 1.2Ghz de fréquence d'horloge.*

application de prédiction d'une concentration de sortie en CO_2 d'un four à gaz (*the Box and Jenkins gas furnace database)*[33].

Une application de prédiction temporelle est complètement différente de la problématique précédente, en l'occurrence la reconnaissance de séquences temporelles. La prédiction temporelle est considérée comme une application d'approximation de fonctions et non de classification. Les variables de sortie du réseau de neurones sont donc des variables de type réel et non catégoriel et la couche de sortie du réseau est constituée par des neurones linéaires. La réponse du réseau *RRFR* est donnée par l'expression ci-dessous :

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \phi(\|\mathbf{x} - \boldsymbol{\mu}_n\|) \qquad [185]$$

avec $h(\mathbf{x}) \in \Re$ qui représente la sortie du réseau pour le vecteur d'entrée $\mathbf{x}$. On remarque clairement que la réponse du réseau de neurones est fortement liée aux valeurs des poids des connexions de sortie. Le vecteur des pondérations est, dans ce type d'application, un des paramètres à prendre en considération avec plus de rigueur que dans une application en classification. Le processus d'apprentissage du *RRFR* pour une application de type approximation de fonction comporte alors deux phases :

- *une première phase* dite non supervisée, pour le calcul des paramètres des neurones gaussiens (centres ou prototypes $\boldsymbol{\mu}$ et rayons d'influence σ),
- *une deuxième phase* dite supervisée, pour le calcul du vecteur de pondération de la couche de sortie. Nous avons adopté la méthode de l'inversion matricielle qui, comme nous le verrons, donne d'assez bons résultats avec pratiquement un temps de calcul négligeable. Une écriture sous forme matricielle nous permet donc de déduire aisément le vecteur de pondération par simple inversion matricielle :

$$\mathbf{w}=\boldsymbol{\Phi}^{-1}.\boldsymbol{\zeta} \qquad [186]$$

$\boldsymbol{\zeta}$ représente le vecteur de sortie désiré. On comprend alors que cette phase de calcul du vecteur $\mathbf{w}$ dépend d'une première phase qui est celle du calcul des paramètres des gaussiennes (centres $\boldsymbol{\mu}$ et rayons d'influence σ) pour avoir la matrice $\boldsymbol{\Phi}$.

Nous nous intéresserons donc plus en détail à la première phase de l'apprentissage. Plusieurs techniques existent pour cette première phase du calcul. Nous allons voir que les algorithmes d'apprentissage de type heuristique[34] *RCE* et *DDA*, qui ont été présentés au chapitre II, peuvent présenter, sous certaines conditions, une bonne solution pour calculer la matrice des gaussiennes $\boldsymbol{\Phi}$. Ces techniques présentent certains désavantages comme leur sensibilité aux paramètres de l'apprentissage. Les centres des gaussiennes ne sont pas

[33]. *Cette base est disponible sur le serveur du groupe de travail IEEE Working Group on Data Modeling Benchmarks , http://neural.cs.nthu.edu.tw/jang/benchmark/*

[34] *dites aussi techniques incrémentales.*

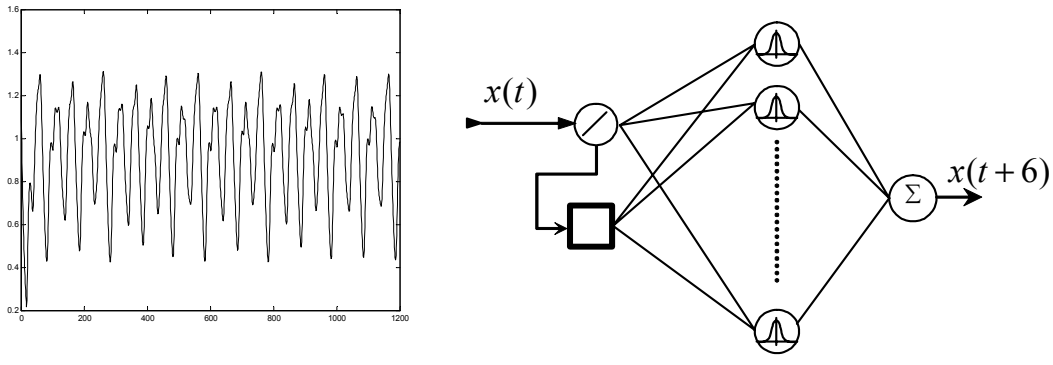
déterminés par rapport à un critère à minimiser, comme pour la technique des *k-moyennes*. En effet, l'algorithme des *k-moyennes* détermine les centres des gaussiennes en minimisant les moyennes des distances quadratiques. Cette technique offre de meilleurs résultats que les techniques heuristiques mais présente également quelques inconvénients. Nous proposons dans cette partie une version améliorée de l'algorithme des *k-moyennes* qui pallie les faiblesses de la version classique. Tous ces points seront traités et mis en évidence dans les paragraphes suivants à travers deux exemples types.

V.3.1. La série temporelle de Mackey-Glass

Soit la série temporelle de Mackey-Glass définie par l'équation différentielle à délai (Mackey *et al.,* 1977):

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-\Delta)}{1+x(t-\Delta)^{10}} \qquad [187]$$

La technique du Runge-Kutta du 4ème ordre est employée pour simuler le système. Deux valeurs de Δ se distinguent conduisant à un comportement chaotique quasi-périodique : $\Delta = 17$ et $\Delta = 30$. Les données que nous allons utiliser dans cette partie ont été obtenues avec les conditions suivantes : $\Delta = 17$, $x(0) = 1.2$ *et* $x(t-\Delta) = 0$ *pour* $0 \le t < \Delta$ avec une fréquence d'échantillonnage de 1. La base de données contient 1200 points : les 500 premiers points à partir du 118ème point sont utilisés pour la phase d'apprentissage, le reste des points (à partir du 618ème point) pour le test du réseau. L'apprentissage ainsi que le test de prédiction du réseau de neurones se fera sur un horizon de six pas : $x(t+6)$ (Figure 83). Le réseau RRFR testé dans cette partie comporte une cascade d'un neurone linéaire et un neurone récurrent.



-a- -b-

Figure 83. *a) Série Mackey-Glass, b) RRFR pour la prédiction de x(t+6)*

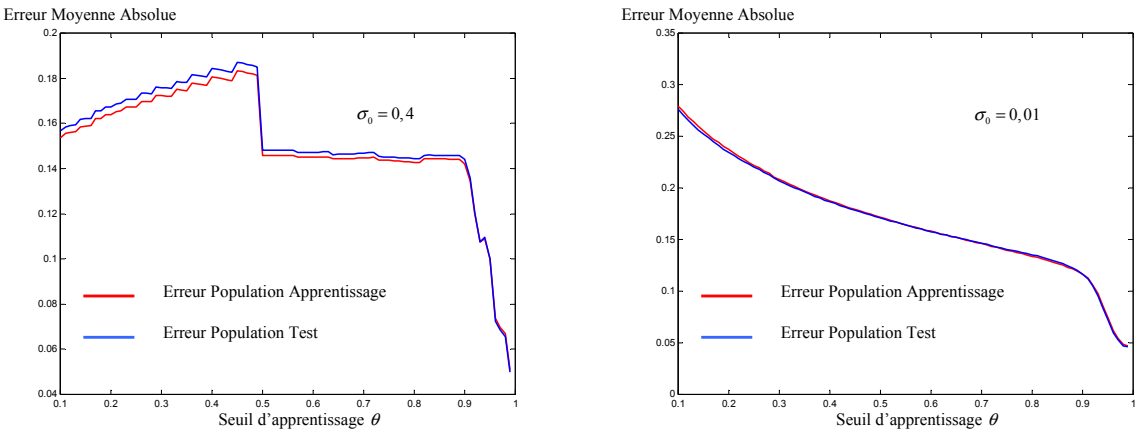
V.3.1.1. Techniques RCE et DDA

Comme nous l'avons présenté au chapitre II, ces deux techniques sont généralement dédiées à des problématiques de classification. On peut néanmoins, sous certaines hypothèses, les appliquer à l'approximation de fonction. En effet, dans ce cas, la notion de *classe d'appartenance des données* n'existe pas. Pour pouvoir utiliser ces deux techniques heuristiques, il faudrait supposer deux cas possibles :

- *Même groupe de centres* : tous les points de la base d'apprentissage font partie du même groupe de centres (équivalant à la même classe en faisant l'analogie avec les problématiques de classification). Le nombre de prototypes mémorisés serait alors inférieur au nombre de population d'apprentissage contenue dans la base de données,
- *Différents groupes de centres* : chaque point d'apprentissage appartient à un groupe différent (équivalant à des classes différentes pour chaque point, par analogie aux problèmes de classification). Tous les points de la base d'apprentissage seraient dans ce cas mémorisés en tant que centre (prototype). Il est évident que cette situation provoquera un phénomène de sur-apprentissage.

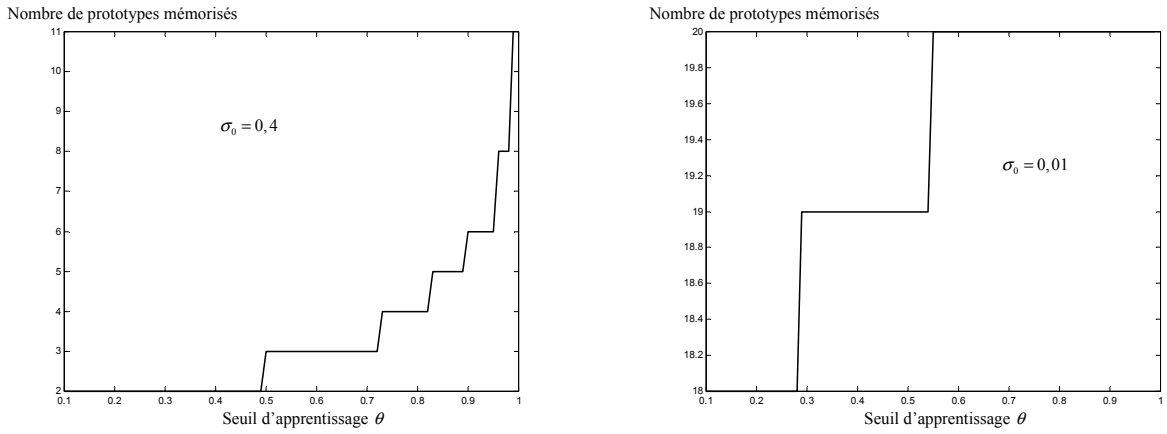
a) Appartenance au même groupe de centres

En considérant ce premier cas, en appliquant l'algorithme *RCE* et en faisant varier le seuil d'apprentissage θ entre $[0.1, 0.99]$, on obtient les résultats de la Figure 84 avec deux valeurs initiales du rayon d'influence $\sigma_0 = 0.4$ et $\sigma_0 = 0.01$. La Figure 85 présente le nombre de prototypes créés lors de cette phase d'apprentissage.



-a- -b-

Figure 84. *Application de l'algorithme RCE sur le problème de la série Mackey-Glass : a) avec un rayon initial de 0.4, b) rayon initial de 0.01.*

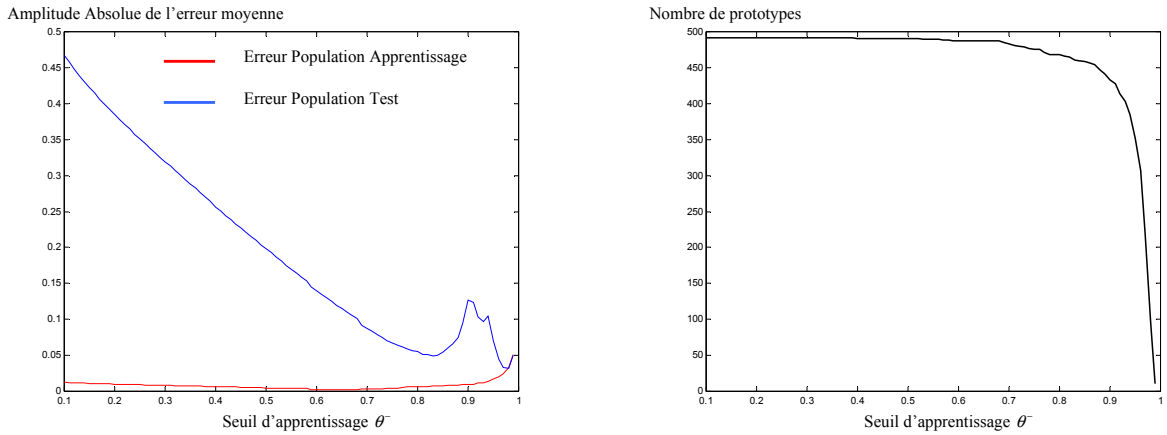


-a- -b-

Figure 85. *Nombre de prototypes créés avec l'algorithme RCE en fonction du seuil d'apprentissage : a) avec un rayon initial de 0.4, b) avec un rayon initial de 0.01.*

La première remarque que nous pouvons faire est que l'erreur sur la population de test est pratiquement identique à celle obtenue sur la population d'apprentissage, ceci pour les deux valeurs initiales du rayon d'influence. La technique *RCE* appliquée sous cette condition procure au réseau de neurones une bonne capacité de généralisation. Les meilleures performances sont obtenues pour des valeurs du seuil proches de 1 ($\theta \approx 1$) avec une erreur moyenne proche de 0.05 pour les deux populations (test et apprentissage) (Figure 84). Par contre, d'après les graphes des Figure 84 et Figure 85, les résultats obtenus sont étroitement dépendants de la valeur initiale du rayon d'influence donnée au premier prototype mémorisé. On peut se poser alors la question suivante : quels sont les critères à prendre en compte pour définir cette valeur initiale σ_0 ? Aucune technique d'initialisation n'est disponible actuellement en littérature.

Dans la deuxième partie des tests, nous avons appliqué l'algorithme *DDA* en essayant d'exploiter les résultats obtenus avec la technique *RCE*. La différence entre ces deux techniques est l'ajout d'un seuil θ^+ supplémentaire pour la technique *DDA*. En effet, cet algorithme possède deux seuils θ^- et θ^+ : le premier a pour rôle de réajuster les rayons d'influence alors que le deuxième contrôle l'ajout de nouveaux prototypes (voir chapitre II pour plus de détails). On a établi les mêmes tests que pour la technique *RCE*. On a fixé la valeur de $\theta^+ = 0.99$ et on a fait varier θ^- entre $[0.1, 0.99]$.



-a- -b-

Figure 86. *Application de l'algorithme DDA sur le problème de la série MackeyGlass : a) erreur moyenne sur population d'apprentissage et sur population de test, b) nombre de prototypes mémorisés en fonction du seuil* θ^- .

La Figure 86.a montre que les performances du réseau *RRFR* sont identiques à celles obtenues avec la technique *RCE* pour les valeurs de $\theta^- \approx \theta^+$ (erreur moyenne de test et apprentissage proche de 0.05). Dans les autres cas ($\theta^- < 0.99$), l'erreur sur la population de test est plus importante, par contre celle sur la population d'apprentissage est quasiment nulle : le réseau souffre alors d'un sur-apprentissage. Ceci est révélateur par le nombre de prototypes créés tout au long de l'apprentissage (Figure 86.b). Le réseau a tendance à apprendre parfaitement les données de la population d'apprentissage, par contre, ses capacités de généralisation sont médiocres.

b) Appartenance à des groupes différents

Dans le cas de groupe différent pour chaque point, tous les points de la base d'apprentissage seront mémorisés en tant que prototypes (centres). Il est évident que le réseau souffrirait alors de sur-apprentissage. Les performances du réseau ne seront pas meilleures que dans le cas précédent (Figure 87).

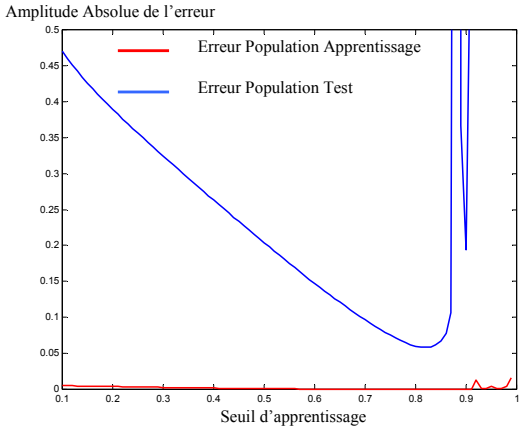
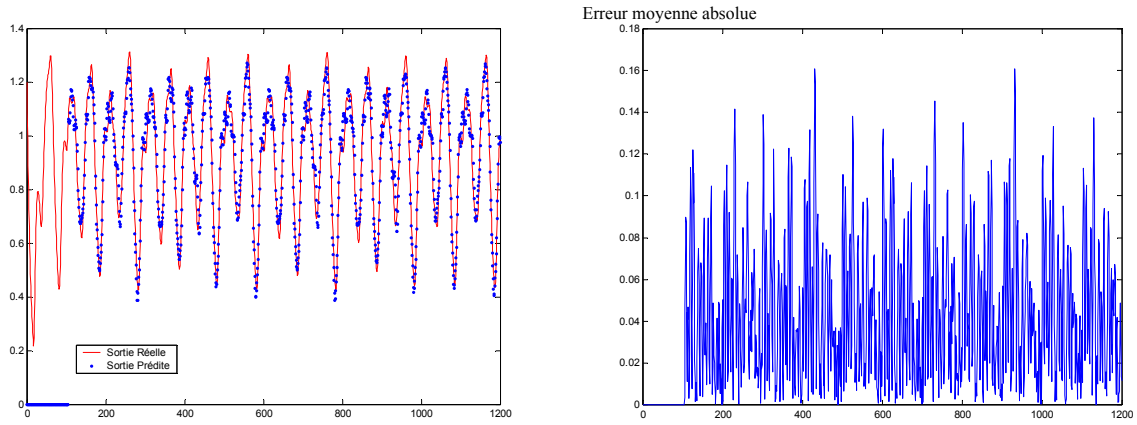


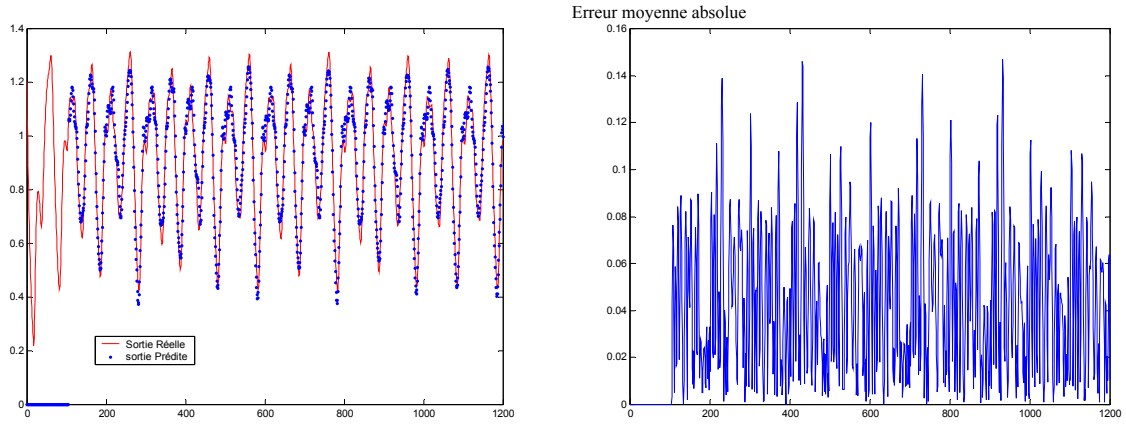
Figure 87. *Test du RRFR avec les techniques heuristiques en faisant l'hypothèse que les points de la base d'apprentissage appartiennent à différents groupes de centres (l'ensemble des points de la base d'apprentissage est mémorisé en tant que prototypes).*

Finalement, les meilleures performances sont obtenues par l'algorithme *RCE* avec la valeur du seuil égale à $\theta = 0,99$. Nous avons donc établi des tests de comparaison entre le réseau *RRFR* avec les deux types de mémoires dynamiques et le réseau *RFR* statique. Ces résultats sont présentés par les figures ci-dessous.



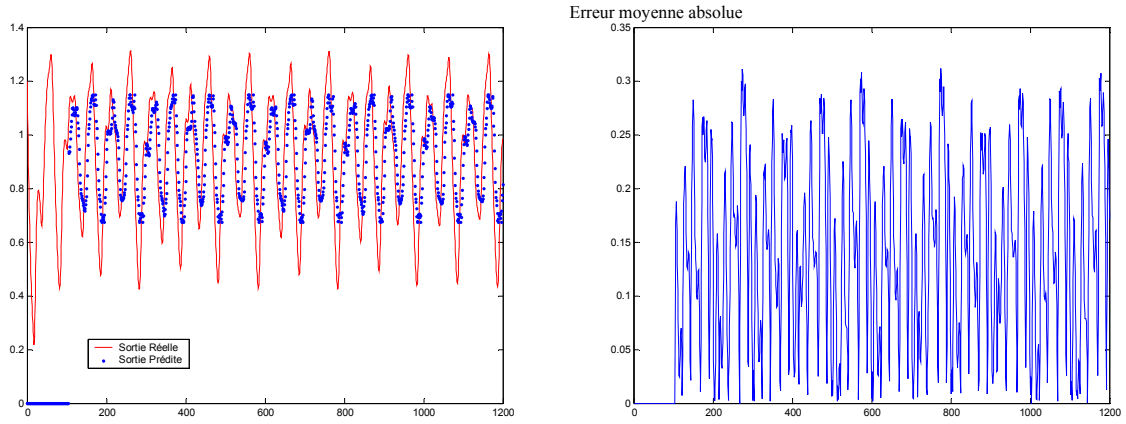
-a- -b-

Figure 88. *Résultat du RRFR avec la mémoire dynamique à neurone bouclé : a) comparaison de la sortie réelle avec celle donnée par le réseau RRFR, b) erreur moyenne absolue.*



-a- -b-

Figure 89. *Résultat du RRFR avec la mémoire dynamique à retour local de l'activation : a) comparaison de la sortie réelle avec celle donnée par le réseau RRFR, b) erreur moyenne absolue.*



-a- -b-

Figure 90. *Résultat du RFR statique (sans mémoire dynamique) : a) comparaison de la sortie réelle avec celle donnée par le réseau RFR, b) erreur moyenne absolue.*

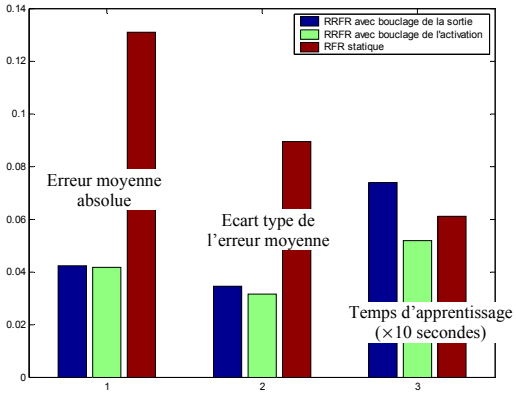


Figure 91. *Comparaison des trois réseaux après apprentissage par l'algorithme RCE avec le seuil de* $\theta = 0{,}99$ *sur le problème de prédiction de la série MackeyGlass.*

Malgré les résultats obtenus, les deux techniques d'apprentissage heuristiques (*RCE* et *DDA*) présentent néanmoins quelques faiblesses. La plus importante est que leurs performances sont étroitement liées aux paramètres que l'expert doit définir. Deux valeurs initiales du rayon d'influence ont donné deux résultats complètement différents, en terme d'erreur de prédiction et également en terme de nombre de prototypes crées (voir Figure 84 et Figure 85). Comment définir la valeur initiale du rayon d'influence afin de garantir un résultat optimal ? Aucune technique répondant à cette question n'existe en littérature mise à part la technique du tâtonnement.

Un autre paramètre qui influe grandement sur la qualité des résultats obtenus, est le seuil θ (où θ^- et θ^+ pour la technique *DDA*). Il a fallu, dans ce cas également, simuler le réseau *RRFR* sur toute la plage de variation des seuils afin d'aboutir au meilleur résultat. Ces deux techniques souffrent aussi d'une certaine dépendance à l'ordre dans lequel sont présentées les données pour l'apprentissage. Nous ne pouvons donc pas garantir l'optimalité des résultats obtenus avec ces deux techniques. On peut néanmoins attribuer deux avantages non négligeables à la technique *RCE* : une bonne capacité de généralisation procurée au réseau *RRFR* (erreur sur population de test quasi égale à celle sur la population d'apprentissage) et également la rapidité de convergence de l'algorithme. Dans la partie suivante, nous allons tester les capacités de prédiction du RRFR avec la technique des *k-moyennes*.

V.3.1.2. Algorithme des k-moyennes

Les faiblesses des deux techniques abordées précédemment peuvent être contournées avec d'autres techniques plus robustes. Nous avons présenté au chapitre II une technique de partitionnement appelée technique des *k-moyennes*[35]. Cette technique a pour principe de déterminer le centre de tout un nuage de points :

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \chi_j} \mathbf{x} \qquad [188]$$

Ce centre représente le point qui minimisera la distance quadratique entre ce dernier ($\boldsymbol{\mu}_j$) et tous les autres points ($\mathbf{x}$) du nuage (χ_j). Cette distance quadratique minimum sera la variance du prototype (*carré du rayon d'influence*):

$$\sigma^2{}_j = \frac{1}{N} \sum_{\mathbf{x} \in \chi_j} (\mathbf{x} - \boldsymbol{\mu}_j)(\mathbf{x} - \boldsymbol{\mu}_j)^t \qquad [189]$$

Cet algorithme a donc l'avantage sur les techniques incrémentales précédemment présentées de déterminer les prototypes les plus représentatifs d'un nuage de points. Comme

[35] Appelée également technique des centres mobiles. Son appellation anglophone est « *k-means clustering algorithm* »

pour les techniques *RCE* et *DDA*, l'algorithme des *k-moyennes* est généralement utilisé dans des problématiques de classification. Pour pouvoir l'utiliser en approximation de fonctions, il faudrait supposer que tous les points de la base d'apprentissage appartiennent au même groupe de centres (même classe en faisant l'analogie avec une application de reconnaissance des formes). On peut résumer le principe de fonctionnement de cet algorithme par les étapes suivantes :

- *1. initialisation du nombre de centres k,*
- *2. initialisation aléatoire des k centres* $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k\}$,
- *3. faire jusqu'à pas de changements :*
  - *3.1. affectation de chaque point* $\mathbf{x} \in \chi$ *au centre* $\boldsymbol{\mu}_i$ *le plus proche,*
  - *3.2. calcul des nouveaux centres* $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k\}$ *de chaque nuage de points avec l'équation [188],*
- *4. calcul du rayon d'influence de chaque centre avec l'équation [189].*

Tableau 11. *Etapes de l'algorithme k-moyennes*

Après quelques tests, nous avons confirmé le fait que l'algorithme des *k-moyennes* donne de meilleures prédictions que les deux techniques heuristiques (*RCE* et *DDA*). Ce résultat est pleinement justifié par le critère de calcul des prototypes qui est celui de minimiser la distance quadratique entre l'ensemble des prototypes et tous les points de la base d'apprentissage. Par contre, comme nous l'avons cité au chapitre II, cet algorithme possède quelques faiblesses qui sont :

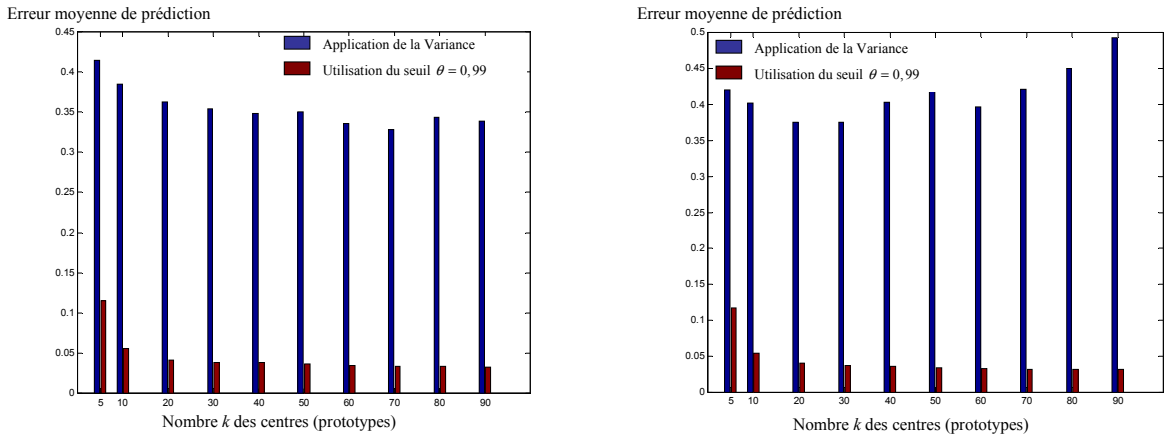
Problème des nuages vides :

En faisant quelques tests avec cette version de l'algorithme, nous avons remarqué qu'on peut avoir des situations où le nuage de points associé à un prototype soit vide. Ceci engendre un gros souci pour calculer le rayon d'influence du prototype (division par zéro de l'équation [189]). Trois solutions sont alors possibles pour résoudre ce problème :

- soit que l'on décide d'affecter une valeur donnée au rayon du nuage vide, et dans ce cas la même question se pose : sur quel critère se baser pour définir cette valeur ?
- soit que l'on élimine ce point de la liste des *k* centres,
- soit de calculer le rayon d'influence en essayant d'exploiter les résultats précédents, c'est-à-dire utiliser le seuil de la technique *RCE*.

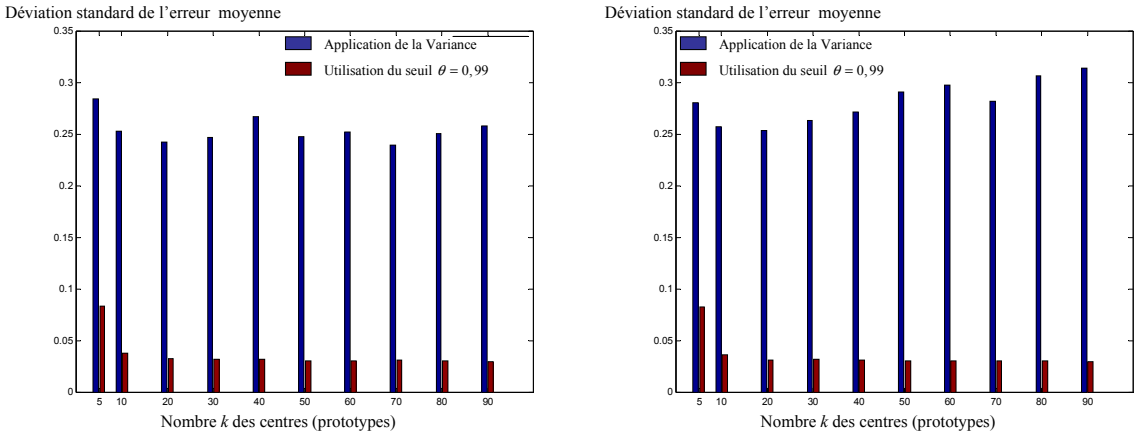
Nous avons fait des tests de comparaison, au niveau de l'étape 4 de l'algorithme des *k-moyennes* (la phase de calcul du rayon d'influence) entre la technique de la variance (équation

[189]) et la technique du seuil θ de l'algorithme *RCE*. Toutes les autres étapes de l'algorithme des k-moyennes (étapes 1, 2 et 3) n'ont pas été modifiées. Les Figure 92 et Figure 93 montrent les résultats obtenus par ces tests de comparaison. On remarque que le calcul des rayons d'influence par la technique du seuil θ de l'algorithme RCE donne, de meilleures performances. Nous avons donc opté pour cette procédure pour calculer les rayons d'influence de chaque prototype (étape 4 de l'algorithme des k-moyennes). C'est cette méthode de calcul que nous avons utilisée dans ce que nous appellerons la version simple de l'algorithme des *k-moyennes*.



-a- -b-

Figure 92. *Comparaison des performances de l'algorithme des k-moyennes avec les deux techniques de calcul du rayon d'influence : application de la variance et utilisation du seuil θ de l'algorithme RCE (avec la valeur du seuil $\theta = 0,99$): a) erreur moyenne sur la population d'apprentissage en fonction du paramètre k, b) sur la population de test.*

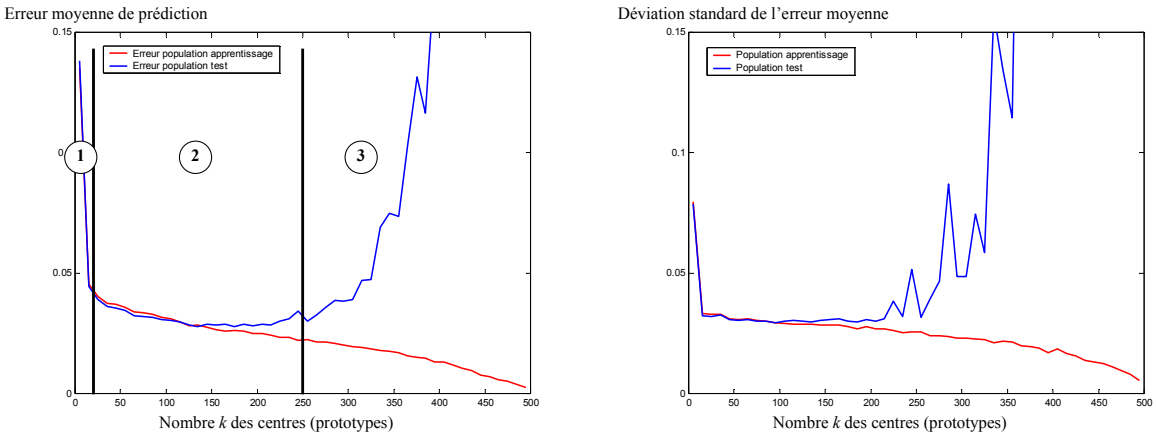


-a- -b

Figure 93. *Déviation standard des erreurs moyennes de prédiction : a) sur population d'apprentissage, b) sur population de test.*

Problème du choix du nombre de prototypes k :

Le deuxième point de l'algorithme qui influe sur les performances du réseau de neurones est le choix du nombre *k* de centres (prototypes) que le réseau devra mémoriser (étape 1 de l'algorithme des *k-moyennes*). La Figure 94 montre l'influence du choix de ce paramètre sur les performances de prédiction du réseau *RRFR*. Trois zones sont mises en évidence : une zone de sous-apprentissage (1), une zone de bonne généralisation (2) et une zone de sur apprentissage (3). Il faudrait donc initialiser le nombre *k* pour que le réseau de neurones se trouve dans la zone (2). Il n'existe pas de méthodes formelles pour initialiser a priori le nombre de centres au début de l'algorithme (Chang *et al.,* 2001).



- a - - b -

Figure 94. *Influence du paramètre k sur les performances de prédiction du réseau RRFR : a) erreur moyenne de prédiction, b) déviation standard de l'erreur moyenne de prédiction.*

Problème d'initialisation des k centres $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k\}$ *:*

Au tout début de l'algorithme d'apprentissage, les *k* centres sont initialisés (étape 2 de l'algorithme des *k-moyennes*) généralement d'une façon aléatoire. Cette initialisation aléatoire rend l'algorithme très instable : il faudrait alors plusieurs tentatives d'exécution de l'algorithme pour obtenir un bon résultat, comme le montre la figure suivante :

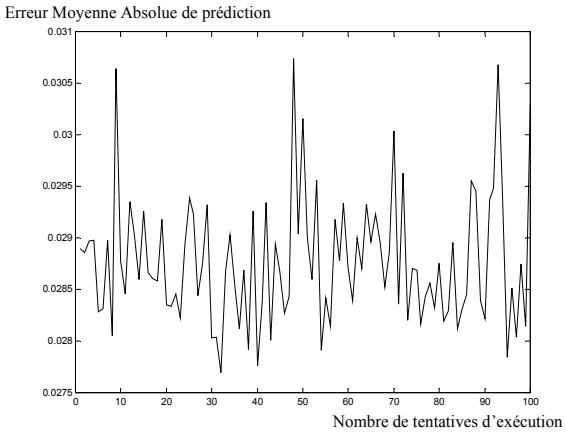


Figure 95. *Instabilité de la technique par rapport à l'initialisation aléatoire des centres.*

Cette partie montre que les performances obtenues avec l'algorithme des *k-moyennes* sont meilleures que celles obtenues avec les techniques heuristiques. Les *k* centres obtenus à la convergence de l'algorithme, sont déterminés de façon à minimiser la distance quadratique moyenne de chaque nuage. Celle-ci représente la variance du centre du nuage. Ces *k* centres sont donc mieux répartis sur l'ensemble des données comparé aux techniques heuristiques. Par contre, la technique des *k-moyennes* souffre de plusieurs désavantages. Nous proposons dans la partie suivante une version améliorée de l'algorithme des *k-moyennes* qui tente de contourner les désavantages de la version classique.

V.3.1.3. Proposition d'un algorithme d'apprentissage

Ce problème d'initialisation des centres peut être résolu par une technique appelée *Fuzzy Min-Max* (Simpson, 1992), (Simpson, 1993), (Chang *et al.,* 2001). Cette technique permet de déterminer le nombre *k* des centres et leur valeur initiale d'une manière itérative. L'algorithme des *k-moyennes* peut être ainsi *« boosté »* pour converger vers le minimum de la somme des erreurs quadratiques entre les vecteurs d'entrée et les *k* centres. Durant cette phase d'initialisation, des hyper-cube à *n* dimensions sont créés. Les limites d'un hyper-cube sont définies par les coordonnées maximales et minimales de chaque dimension des points appartenant à cet hyper-cube. Un degré d'appartenance d'un point à chaque hyper-cube est déterminé par la fonction d'appartenance ci-dessous :

$$H_j(x, v_j, u_j) = \frac{1}{n}\sum_{i=1}^{n}\left[1 - f(x_i - u_{ji}) - f(v_{ji} - x_i)\right] \qquad [190]$$

où la fonction floue f est définie par l'expression suivante :

$$f(x)=\begin{cases}1, & x>\eta \\ x/\eta, & si\ 0\le x\le\eta \\ 0, & x<0\end{cases} \qquad [191]$$

avec :

H_j le degré d'appartenance d'un point x à l'hyper-cube j. Ce degré d'appartenance est compris dans l'intervalle $[0,1]$;

x_i la $i^{ème}$ dimension du vecteur d'entrée x ;

u_{ji} et v_{ji} la valeur de la $i^{ème}$ dimension des points maximums et minimums respectivement du $j^{ème}$ hyper-cube.

Le paramètre η est appelé sensibilité de l'hyper-cube. Sa valeur détermine la pente de la décroissance du degré d'appartenance H_j (équation [190]) d'un point en fonction de son éloignement par rapport à l'hyper-cube j. Pour de petites valeurs de η, on obtient une fonction d'appartenance H_j plutôt raide (on perd l'aspect flou). Les recouvrements entre hyper-cubes sont dans ce cas réduits. Inversement, pour de grandes valeurs de η, on obtient une fonction d'appartenance H_j très floue, et les recouvrements entres hyper-cubes sont plutôt importants. Les auteurs de cet algorithme ne donnent aucune méthode formelle pour initialiser ce paramètre de sensibilité du degré d'appartenance. Le seul critère donné est d'initialiser ce paramètre de telle sorte à minimiser les recouvrements entre hyper-cubes (c'est-à-dire donner de petites valeurs à η). Afin de respecter ce critère et de vouloir garder un certain aspect flou au degré d'appartenance, nous proposons l'expression suivante qui permet d'initialiser la valeur de η :

$$\eta=\min_i \frac{\max_{\mathbf{x}_j\in\chi}(x_{ji})-\min_{\mathbf{x}_j\in\chi}(x_{ji})}{2\times(N-1)} \qquad [192]$$

avec

x_{ji} la $i^{ème}$ dimension du vecteur d'entrée $\mathbf{x}_j$ appartenant au nuage de points χ ;

N le nombre total de points du nuage χ ;

La Figure 96 permet d'illustrer un exemple de calcul de la sensibilité η pour un problème à deux dimensions et un nuage χ contenant deux points $\{\mathbf{x}_1,\mathbf{x}_2\}$.

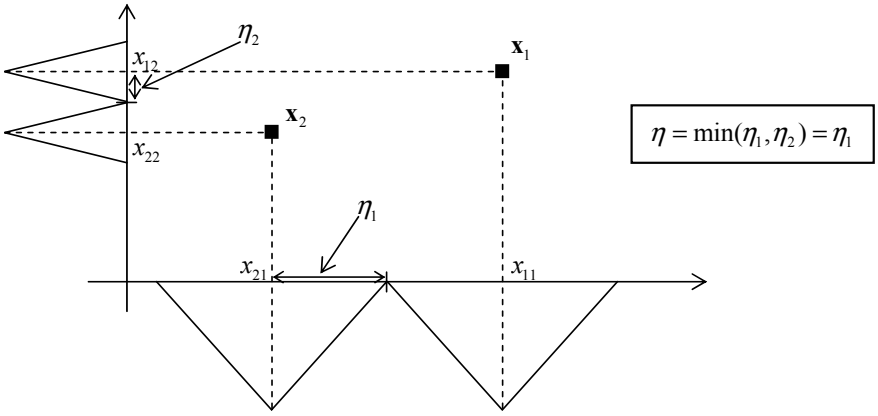


Figure 96. *Calcul du paramètre* η.

L'algorithme *Fuzzy Min-Max* possède trois phases : extension de l'hyper-cube, test de recouvrement et phase de re-dimensionnement de l'hyper-cube. Pour la phase d'initialisation des *k* centres, nous n'avons utilisé que la partie extension pour former les différents nuages de points. Les différentes étapes de l'algorithme sont les suivantes :

- 1. Initialisation des valeurs maximales et minimales du premier hyper-cube par le premier point présenté au réseau.
- 2. Calcul du degré d'appartenance de chaque point d'entrée par l'équation [190].
- 3. L'extension de l'hyper-cube ayant la plus grande fonction d'appartenance se fait selon la condition suivante :

$$\sum_{i=1}^{n} (max(u_{ji}, x_i) - min(v_{ji}, x_i)) \leq n\theta \qquad [193]$$

où θ représente un paramètre de l'algorithme qui contrôle la création des nouveaux hyper-cubes. De petites valeurs de θ donnent un nombre important d'hyper-cubes, et inversement, de grandes valeurs de θ donnent un nombre faible d'hyper-cubes. Après cette phase d'extension, les anciens points minimums et maximums de l'hyper-cube sont remplacés par les nouvelles valeurs minimales et maximales.

- 4. Si aucun hyper-cube ne peut être élargi (condition de l'équation [193] n'est pas respectée), un nouvel hyper-cube contenant le nouveau point est créé.

Après avoir présenté au réseau l'ensemble des données d'apprentissage, un certain nombre d'hyper-cubes sont créés en fonction de la valeur du paramètre θ. Ce dernier est le seul paramètre que l'expert doit ajuster. On calcule alors les centres de chaque hyper-cube. Les *k* centres de l'algorithme des *k-moyennes* sont ainsi initialisés. La Figure 97 montre des résultats comparatifs entre une initialisation des *k* centres avec la technique classique (c'est-à-dire initialisation aléatoire) et avec l'algorithme *Fuzzy Min-Max*.

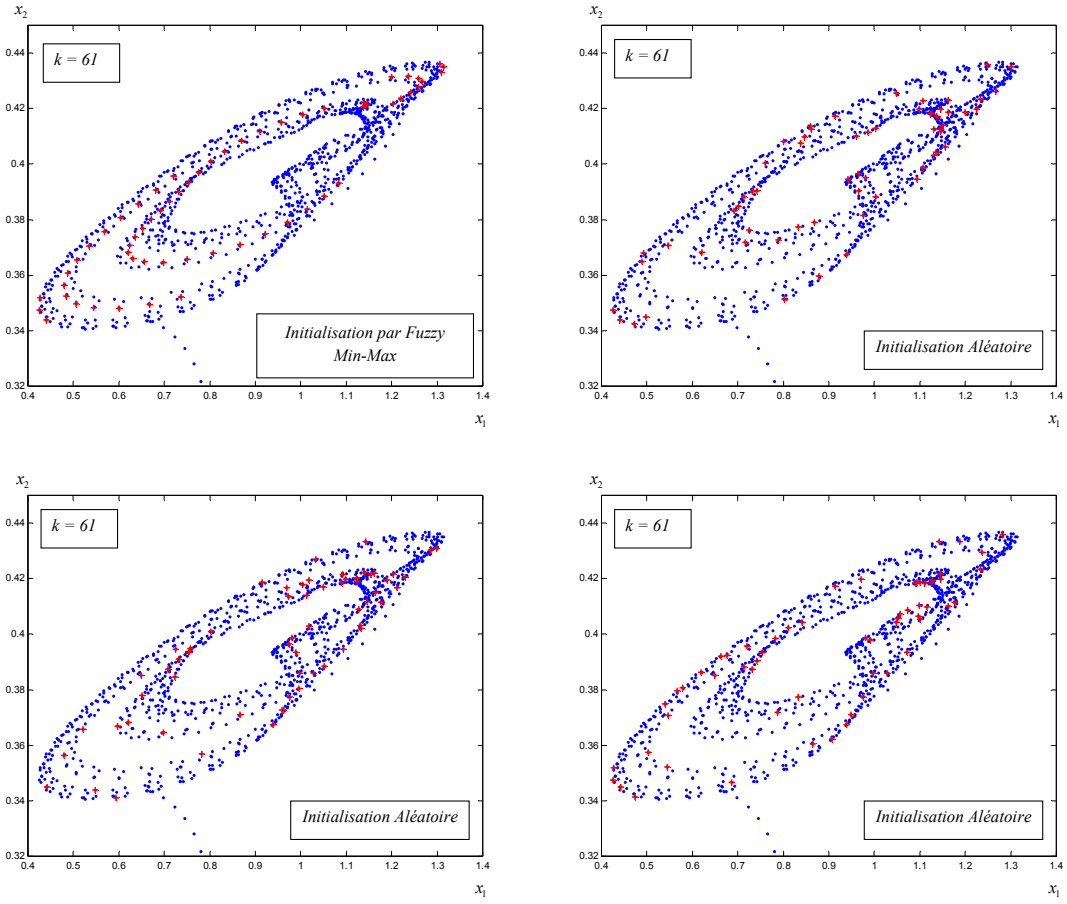


Figure 97. *Différence d'initialisation entre les deux techniques (Fuzzy Min-Max et aléatoire)*

La différence entre les deux façons d'initialiser les centres est clairement identifiée. La technique *Fuzzy Min-Max* permet d'avoir une certaine uniformité des *k* centres par rapport à la densité du nuage de points. Cette initialisation permet alors de booster l'algorithme des *k-moyennes*. On remarque également que l'initialisation aléatoire est instable, ce qui provoque l'instabilité du résultat final (résultat de la Figure 95). Après cette phase d'initialisation des *k* centres, on applique alors l'algorithme des *k*-moyennes pour trouver le minimum des sommes des erreurs quadratiques. Après avoir fait quelques tests de la technique des *k-moyennes* avec cette technique d'initialisation des centres, nous avons remarqué, après convergence de l'algorithme des *k-moyennes,* qu'on obtient plein de nuages vides et d'autres très denses. L'initialisation avec le *Fuzzy Min-Max* provoque une forte disparité de la densité des nuages, alors qu'une initialisation aléatoire fait que la densité des nuages de points soit plutôt équilibrée. Nous illustrons ce résultat sur la Figure 98 :

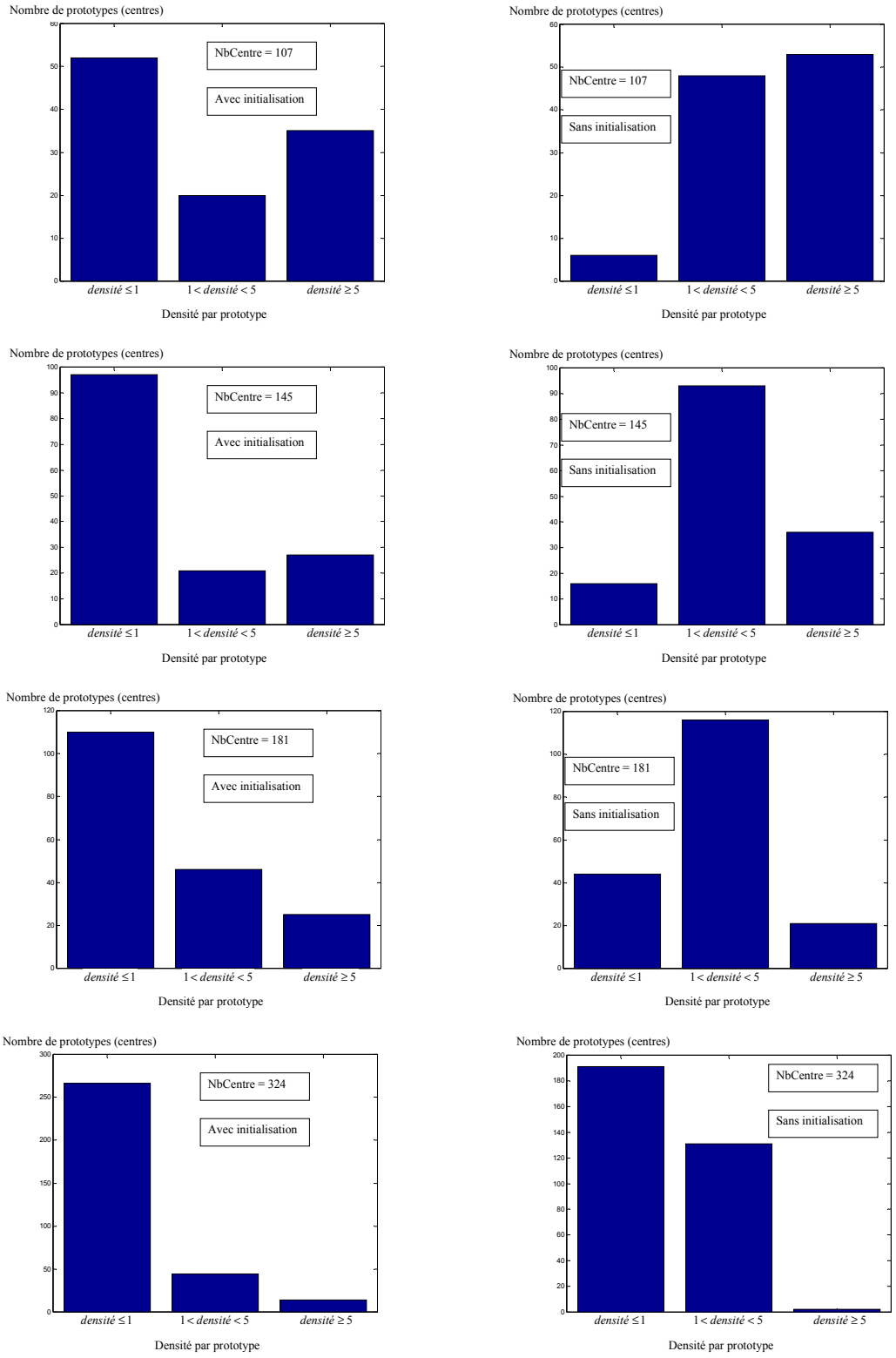


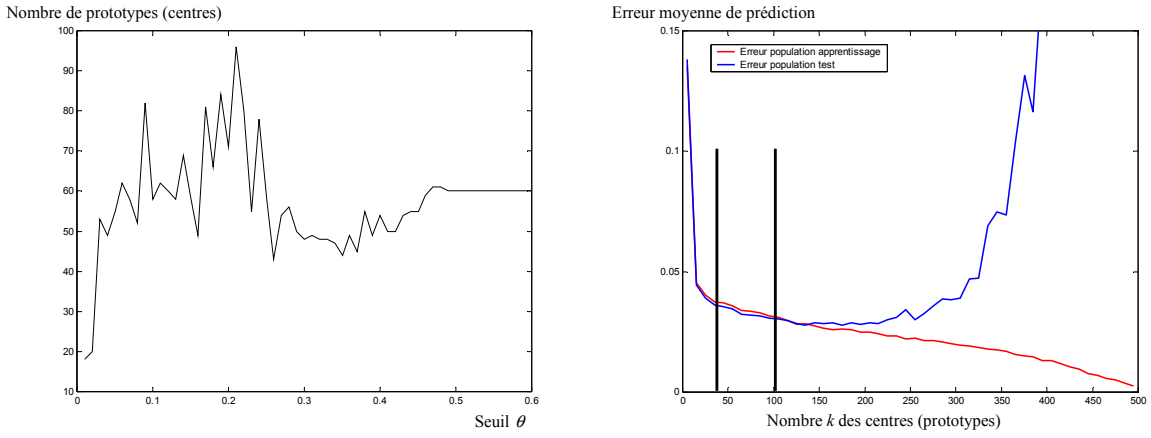
Figure 98. *Comparaison des performances de l'algorithme des k-moyennes avec initialisation en utilisant l'algorithme Fuzzy Min-Max et initialisation aléatoire.*

A partir de ces résultats, nous proposons la version boostée de l'algorithme d'apprentissage des *k-moyennes* suivant :

- *1. Initialisation du paramètre θ de l'algorithme Fuzzy Min-Max* [193],
- *2. Initialisation des k centres avec une itération de l'algorithme Fuzzy Min-Max,*
- *3. Faire jusqu'à pas de changement :*
  - *3.1. Faire jusqu'à pas de changement :*
    - *3.1.1. affectation de chaque point $\mathbf{x} \in \chi$ au centre $\boldsymbol{\mu}_i$ le plus proche,*
    - *3.1.2. calcul des nouveaux centres $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k\}$ de chaque nuage de points avec l'équation [188],*
  - *3.2. Eliminer les centres des nuages vides*
- *4. Calcul des rayons de chaque prototype avec la technique RCE en utilisant la valeur de 0.99 pour le seuil.*

Tableau 12. *Etapes de l'algorithme proposé.*

Le seul paramètre qu'il faut donc ajuster avec cette version boostée est le paramètre θ de l'algorithme *Fuzzy Min-Max* (équation [193]). Pour étudier la sensibilité de cette version par rapport aux variations de ce paramètre, nous avons testé son influence sur la détermination du nombre de prototypes créés et mémorisés par le réseau de neurones. On obtient alors les résultats présentés sur le graphe de la Figure 99 :



-a- -b-

Figure 99. *Influence du paramètre θ sur les performances de l'algorithme : a) Nombre de prototypes crées en fonction des variations du seuil θ, b) limites de la zone des prototypes créés après convergence de l'algorithme des k-moyennes modifié.*

La Figure 99.a montre que quelles que soient les variations du seuil θ, le nombre de prototypes crées se situe dans une zone délimitée par les deux bornes supérieure et inférieure (Figure 99.b). Cette zone se situe bien dans la zone de bonne généralisation. L'algorithme force le réseau à converger vers une zone de bonne généralisation et évite ainsi les zones de sur-apprentissage et de sous apprentissage et ceci quelles que soient les variations du seuil θ. Cet algorithme est donc moins sensible à la phase délicate de paramétrage.

Nous avons par ailleurs établi une série de tests comparatifs entre les performances obtenues avec l'algorithme proposé et la version simple des *k-moyennes*. Pour chaque test, nous avons déduit un nombre *k* de centres à partir d'une valeur du paramètre θ, après convergence de l'algorithme des *k-moyennes* modifié. Nous avons ensuite imposé ce nombre de centres à l'algorithme des *k-moyennes* simple. Chaque test comporte 100 itérations complètes des deux algorithmes. Nous présentons sur la Figure 100 les résultats de ce test de comparaison.

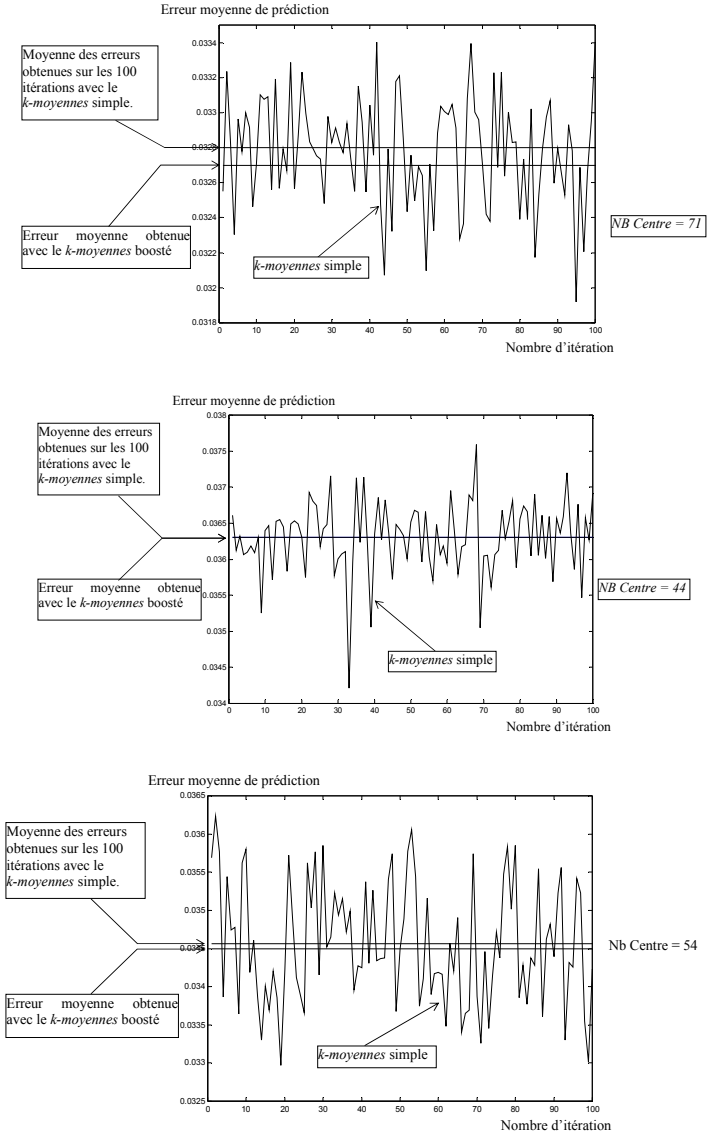


Figure 100. *Stabilité de l'algorithme en fonction des itérations.*

Pour mieux expliquer les résultats obtenus et ainsi mettre davantage en évidence la différence entre la version simple de l'algorithme des *k-moyennes* et la version améliorée que nous proposons, nous avons fait des tests de comparaison sur un exemple assez illustratif présenté sur la Figure 101. La figure (a) montre une initialisation des centres avec une itération de l'algorithme *Fuzzy Min-Max*. Cette initialisation a permis de créer un hyper-cube pour chacun des 5 nuages de points et un hyper-cube contenant un seul point pour chacun des 8 points isolés. Les centres crées sont schématisés par des croix rouges. Les centres de nuages vides seront éliminés par notre algorithme. En fin de calcul, les centres obtenus sont ceux représentés par la figure (b). L'algorithme ne garde ainsi que les centres les plus représentatifs d'une population importante de points. Les centres vides représentent des points isolés et il serait donc plus judicieux de les éliminer. Notons que le résultat obtenu est le même à chaque exécution de l'algorithme (stabilité de l'algorithme), contrairement à la version simple de l'algorithme des k-moyennes qui est assez instable. En effet, les figures (c) et (d) représentent les résultats obtenus avec la version simple des *k-moyennes*. Non seulement cette version est instable (deux résultats différents pour deux exécutions différentes de l'algorithme), mais les centres obtenus ne sont pas du tout représentatifs de la population d'apprentissage.

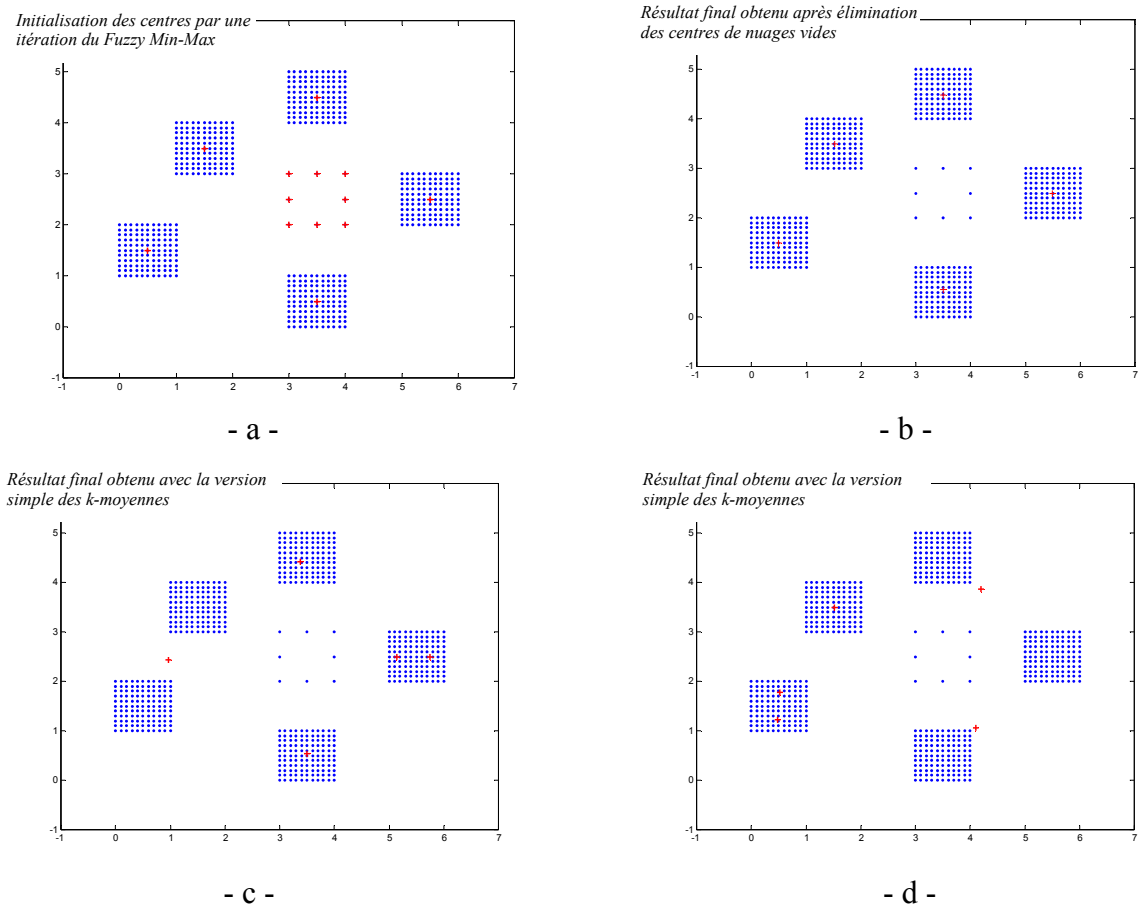


Figure 101. *Exemple de calcul des centres avec la version améliorée des k-moyennes (a) et (b) et avec deux exécutions de la version simple de l'algorithme (c) et (d). Les résultats obtenus avec la version améliorée sont nettement meilleurs et surtout stables par rapport à la version simple.*

Ces tests nous permettent de conclure sur trois remarques :

- La première concerne la stabilité des résultats obtenus avec l'algorithme des *k-moyennes* modifié. En effet, contrairement à la version simple de l'algorithme des *k-moyennes,* la version boostée permet de converger toujours vers le même résultat.
- La deuxième remarque concerne le nombre *k* calculé après la convergence de l'algorithme des *k-moyennes* modifié. Ce nombre *k* permet au réseau *RRFR* de se trouver dans la zone de bonne généralisation en évitant les deux zones de sous-apprentissage et de sur-apprentissage (voir Figure 99).
- La troisième remarque concerne le résultat vers lequel converge le réseau *RRFR*. Ce résultat se trouve parmi les meilleurs résultats pouvant être obtenus par la version simple des *k-moyennes*. Cette troisième remarque est le résultat de la comparaison des performances obtenues avec la technique des *k-moyennes boostée*, et une moyenne des performances de 100 résultats différents obtenus avec la version simple des *k-moyennes* (voir Figure 100).

A partir de tous les résultats obtenus, on peut affirmer que l'algorithme des *k-moyennes* est *boosté* pour converger vers un des meilleurs résultats.

Nous présentons sur la Figure 102 les résultats obtenus en comparant le *RRFR* avec ses deux types de mémoires dynamiques et le *RFR* statique sur le problème de prédiction de la série *MackeyGlass*. Les performances du *RRFR* sont largement meilleures que celles du *RFR* statique. Par contre, les performances des deux mémoires dynamiques sont quasi semblables. Les temps de convergence de la version améliorée des *k-moyennes* sont supérieurs à ceux de la version simple des *k-moyennes*. Ce constat était bien évidemment prévisible puisque la version modifiée des *k-moyennes* possède des opérations supplémentaires par rapport à la version simple de l'algorithme.

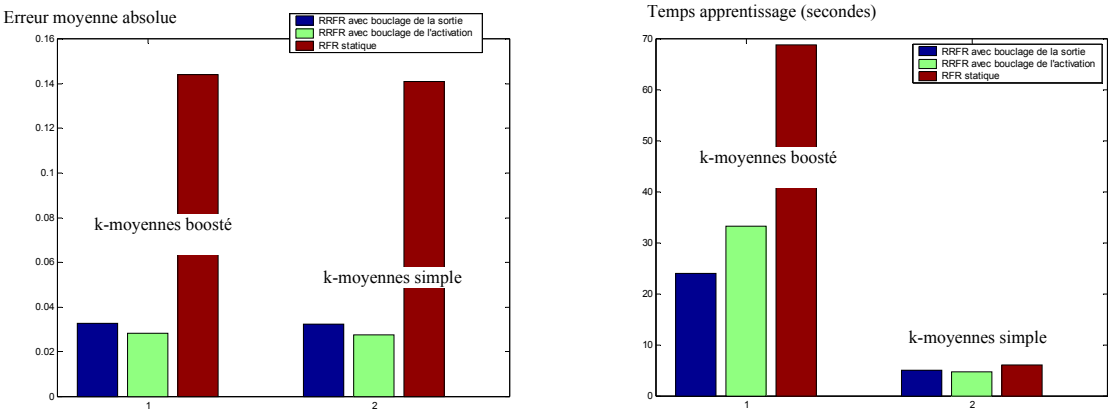


Figure 102. *Comparaison des performances entre les deux techniques d'apprentissage sur le problème de prédiction de la série MackeyGlass.*

Nous avons testé et comparé les performances du réseau *RRFR* sur la prédiction à long terme. Le *RRFR* avec ces deux types de mémoires a été comparé au *TDRBF* et au *RFR* statique sur plusieurs horizons temporels (allant de 1 jusqu'à 100). L'apprentissage des quatre réseaux de neurones a été effectué par la version que nous proposons de la technique des *k-moyennes*. Les meilleures performances de prédiction sont obtenues par le *RRFR*. La figure ci-dessous illustre les résultats obtenus. On remarque également que les quatre réseaux de neurones ont bien capturé la pseudo périodicité du signal qui est d'environ 50 unités de temps. Cette remarque est révélée par les minimums des courbes d'erreurs moyennes situés aux multiples de la demi pseudo période, et par les sommets situés au niveau des multiples des quarts de la pseudo période. Ce comportement est mis en évidence par la courbe du réseau *RFR* statique.

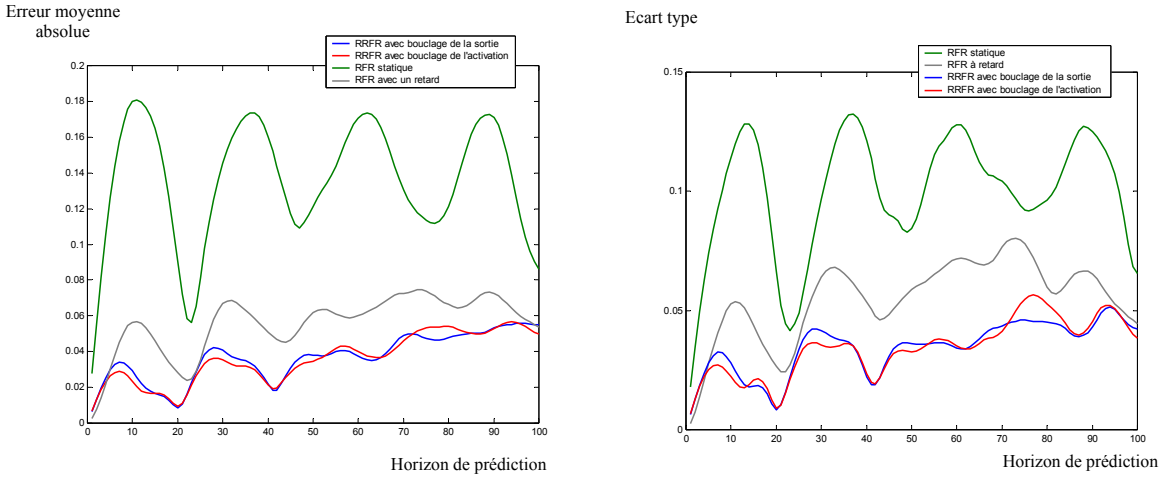


Figure 103. *Comparaison des performances de prédiction entre le RRFR, RFR et le TDRBF (avec un seul retard) sur plusieurs horizons temporels (de 1 à 100).*

V.3.2. Application et Validation sur la prédiction d'un four à gaz

Nous allons dans cette partie valider les résultats obtenus précédemment sur une application de prédiction industrielle avec des données réelles : le benchmark d'un four à gaz. Le but de cette application est de prédire la concentration de sortie en CO_2 $y(t+1)$ à partir de la sortie $y(t)$ et du débit de gaz en entrée $u(t)$. Cette application est schématisée par la Figure 104. Le réseau *RRFR* utilisé comporte un neurone linéaire et un neurone récurrent pour chacune des deux variables d'entrée.

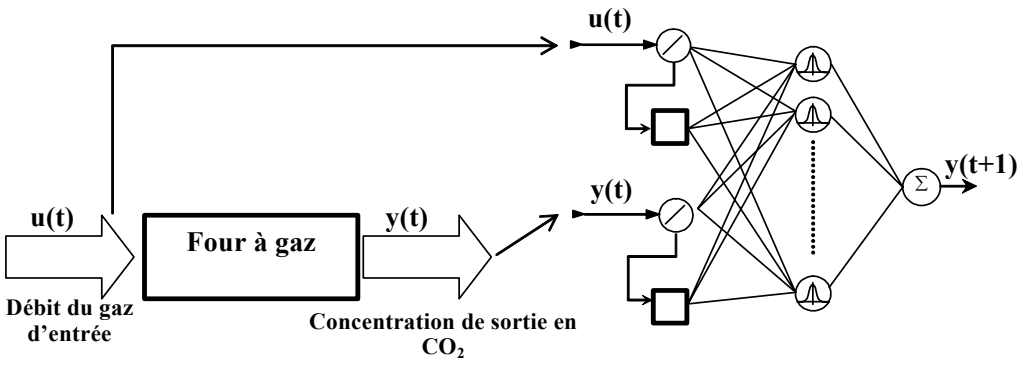
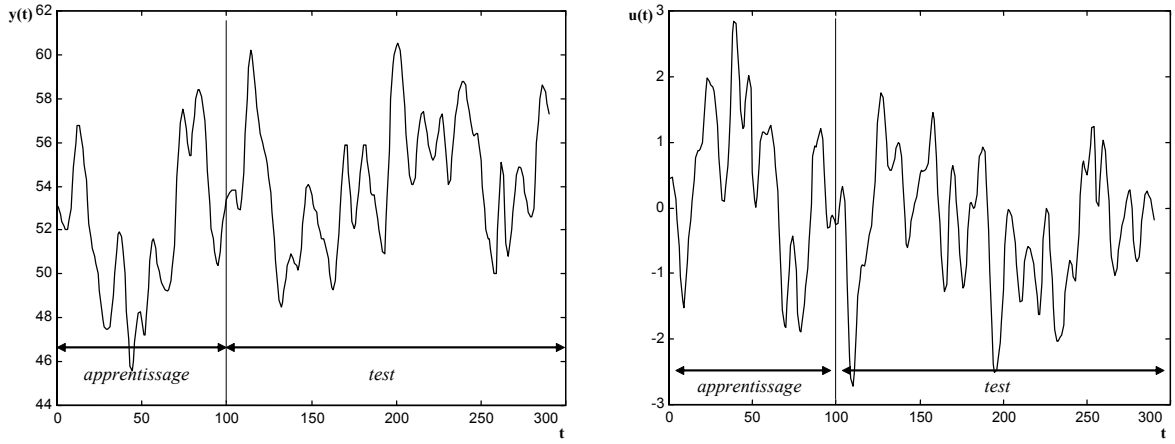


Figure 104. *Surveillance d'un four à gaz par le réseau de neurones RRFR.*

La base de données contient 300 valeurs de $y(t)$ et $u(t)$. Nous avons pris les 100 premières valeurs pour la phase d'apprentissage et les 200 autres pour la phase de test. La Figure 105 montre l'évolution des deux signaux avec les deux groupes choisis pour l'apprentissage et le test.



- a - - b –

Figure 105. *a) Concentration du CO_2 en sortie du four à gaz, b) Débit du gaz en entrée dans le four.*

En faisant des tests d'initialisation des centres avec l'algorithme *Fuzzy Min-Max*, on obtient les mêmes résultats comme dans l'application précédente, c'est-à-dire une forte concentration de nuages vides après convergence de l'algorithme des *k-moyennes* (Figure 106). L'initialisation par la technique *Fuzzy Min-Max* permet donc de garder que les centres les plus représentatifs d'un nuage de données. En appliquant alors toutes les étapes de l'algorithme des *k-moyennes* proposé (voir étapes du Tableau 12), on obtient un nombre de centres k tel que le réseau de neurones se trouve dans la zone de bonne généralisation, ceci quelles que soient les variations du paramètre d'apprentissage θ de l'équation [193] (Figure 107)**.**

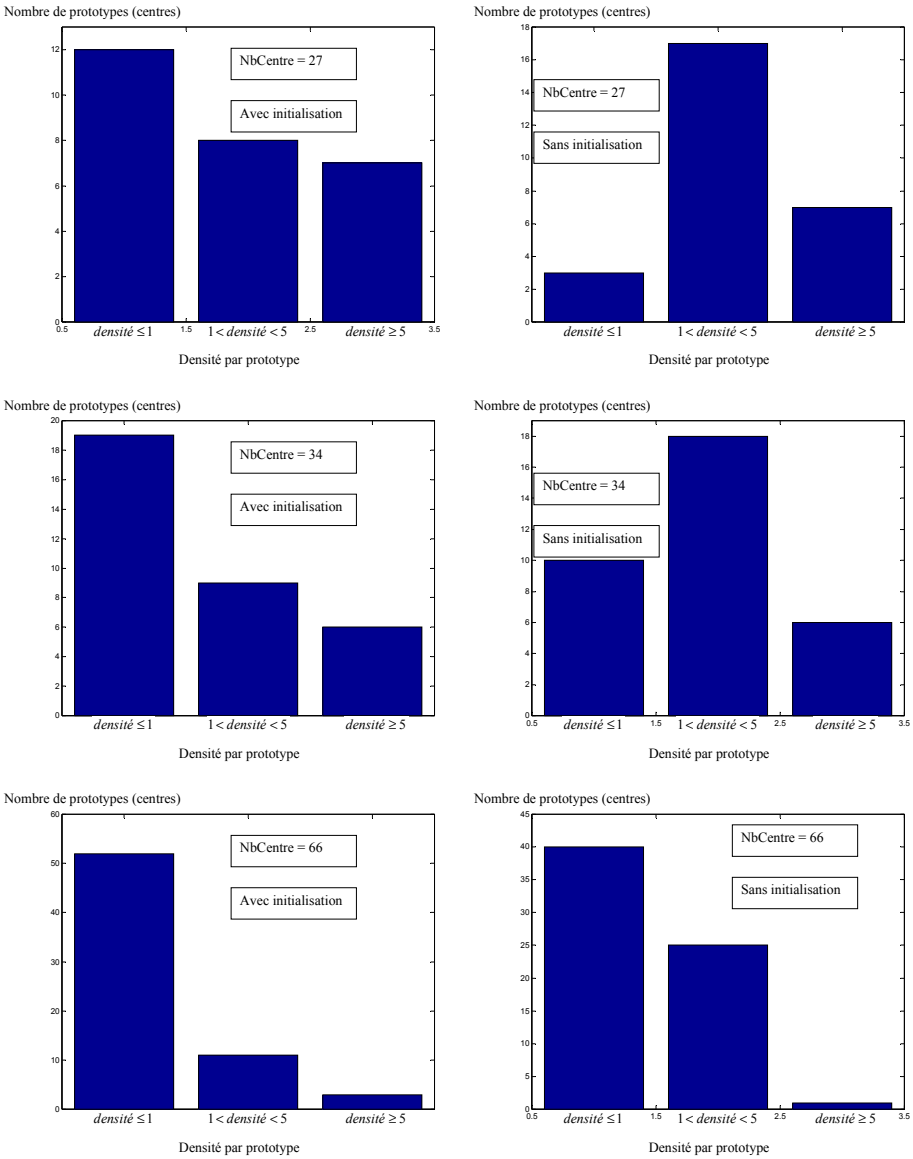


Figure 106. *Comparaison des performances de l'algorithme k-moyennes avec initialisation en utilisant l'algorithme Fuzzy Min-Max et aléatoire.*

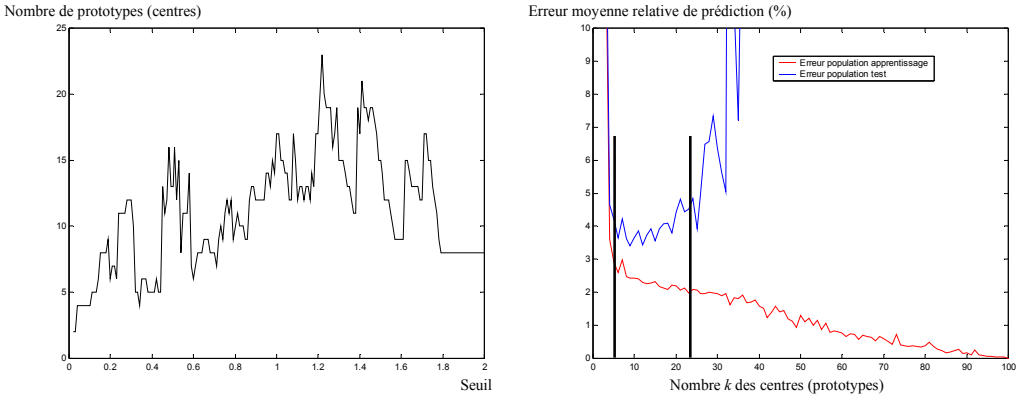


Figure 107. *Nombre de prototypes obtenus en fonction des variations du seuil θ de l'équation* [193] *de l'algorithme Fuzzy Min Max.*

Comme pour l'exemple de la série temporelle Mackey-Glass précédente, quelles que soient les variations du paramètre θ, le réseau *RRFR* converge vers la zone de bonne généralisation. On évite ainsi les deux zones de sur-apprentissage et de sous-apprentissage. Les graphes de la Figure 108 montrent les comparaisons effectuées entre la version améliorée de l'algorithme des *k-moyennes* proposée et la version classique. Les comparaisons ont été effectuées sur une centaine de tests pour chaque exemple. On remarque que, non seulement le résultat obtenu par la version améliorée est stable, mais aussi meilleur que la moyenne des 100 itérations de la version simple de l'algorithme des *k-moyennes*.

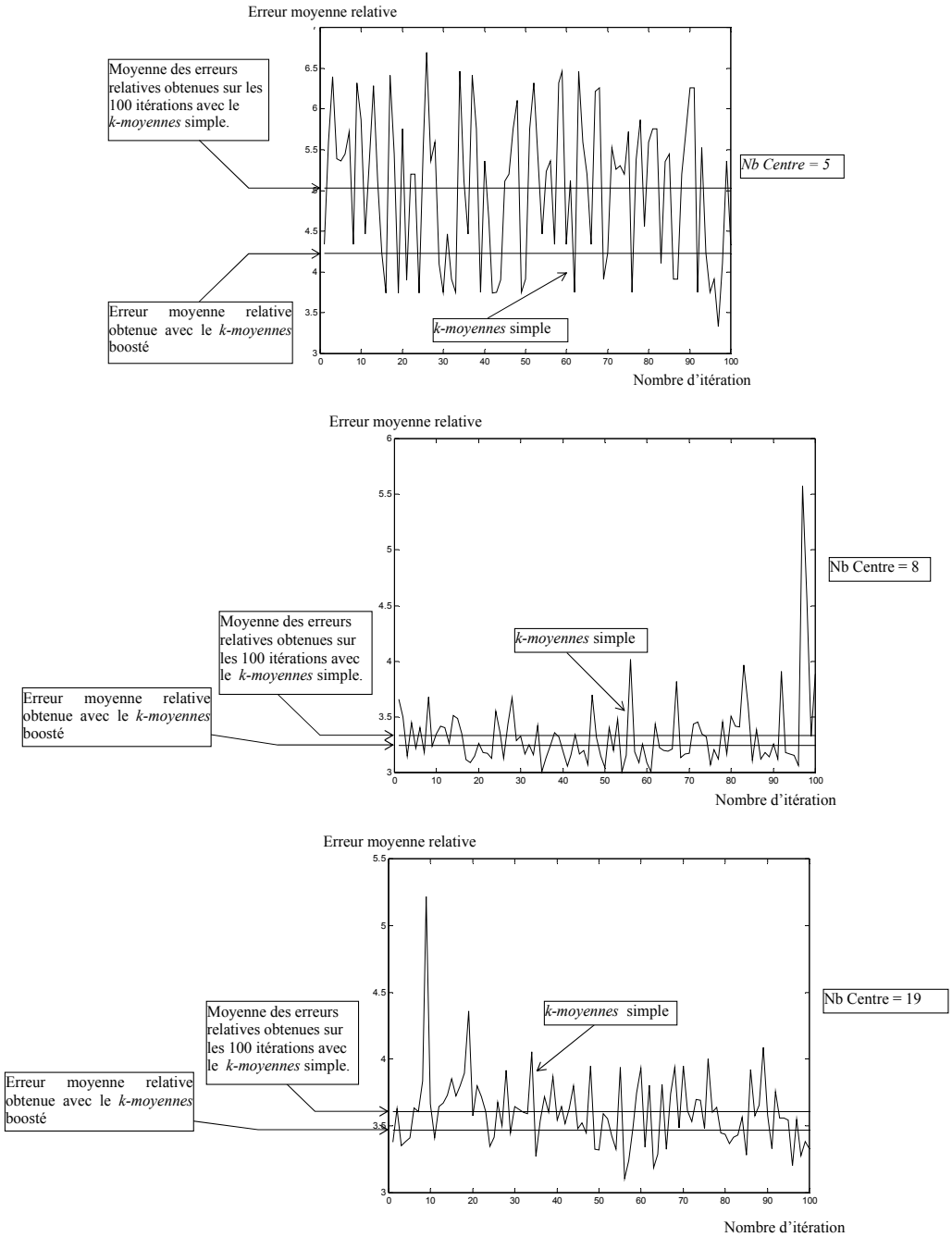


Figure 108. *Comparaison des performances de la version améliorée (boostée) et la version simple des k-moyennes.*

On conclut cette partie avec des tests de comparaison entre les performances de deux types de mémoire dynamique du *RRFR* et le réseau *RFR* statique (Figure 109). Comme dans les situations précédentes, les performances du *RRFR* sont largement meilleures que celles du *RFR* statique. Les performances obtenues par le réseau *RRFR* avec mémoire dynamique à retour local de la sortie sont légèrement meilleures que le *RRFR* avec mémoire dynamique à retour local de l'activation.

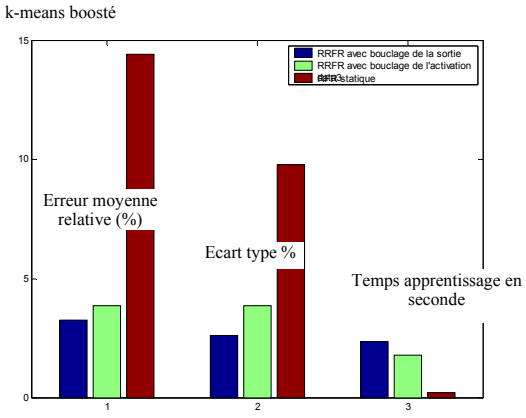
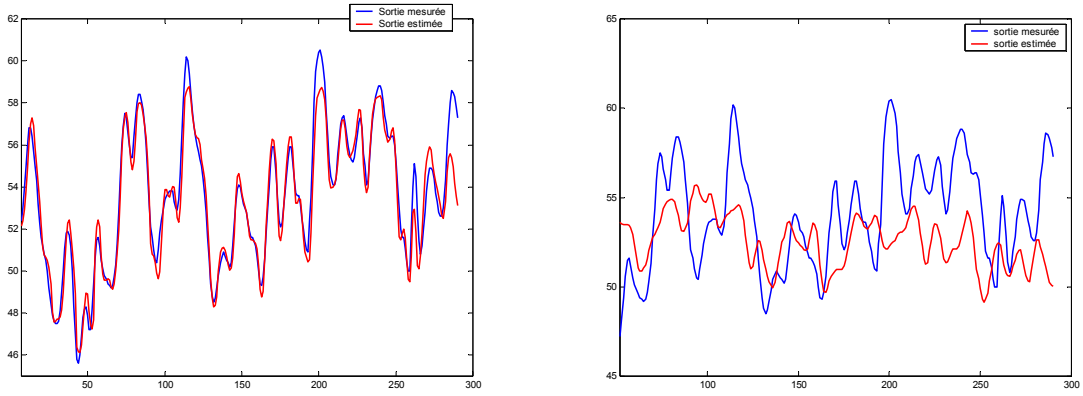


Figure 109. *Comparaison des performances entre les deux types de mémoires dynamiques et le réseau RFR statique sur une prédiction à (t + 1).*

La figure ci-dessous présente les résultats de la prédiction obtenue avec le *RRFR* avec neurone bouclé. La figure de gauche montre les prédictions à un horizon de cinq unités de temps, et celle de droite sur un horizon de cinquante unités de temps. On remarque que dans ce cas, le réseau de neurones a bien capitalisé la forme du signal.



-a- -b-

Figure 110. *Résultat de la prédiction obtenue par le réseau RRFR avec mémoire dynamique à retour local de la sortie. a) prédiction sur un horizon de cinq unités de temps b) prédiction sur un horizon de cinquante unités de temps. Les entrées du réseau sont constituées des valeurs de la commande u(t) et de la sortie y(t).*

Nous avons montré que le réseau *RRFR* est capable d'apprendre des données temporelles afin de prédire leur évolution. Cette capacité de prédiction est réalisée grâce à la mémoire dynamique qui a été greffée au réseau *RFR* statique. Cette mémoire dynamique permet au réseau *RRFR* d'hériter de la simplicité d'apprentissage et d'utilisation du réseau *RFR* tout en ayant un caractère dynamique. Les temps d'apprentissage et de convergence du réseau *RRFR* ne sont pas plus longs que ceux du *RFR* statique. Nous avons également proposé une version améliorée de l'algorithme des *k-moyennes*. A la convergence de l'algorithme proposé, le *RRFR* se trouve dans la zone de bonne généralisation. Cette version permet ainsi de garantir la convergence du réseau vers un des meilleurs résultats avec une sensibilité réduite par rapport au paramétrage de l'algorithme. La parcimonie de l'algorithme d'apprentissage est ainsi améliorée.

V.4. La reproduction de séquences

Le troisième type de test que nous allons faire subir au réseau *RRFR* concerne l'apprentissage à la reproduction de séquences temporelles. En d'autres termes, après avoir appris au réseau de neurones une trajectoire temporelle bien définie, ce dernier devra pouvoir la reproduire librement, ceci sans l'aide d'aucun mécanisme externe. Le moyen qui lui permet d'apprendre et de reproduire de telles séquences temporelles est bien l'existence de la mémoire dynamique du réseau de neurones. L'absence d'une telle mémoire des autres architectures neuronales temporelles les rend incapables de produire ce genre de comportement. Les réseaux récurrents sont donc les seuls à pouvoir apprendre à reproduire toute une séquence temporelle. Au chapitre III, nous avons présenté un large état de l'art sur les différentes architectures temporelles avec les techniques d'apprentissage les plus employées.

Les techniques utilisées pour déterminer les paramètres du réseau de neurones lui permettant de reproduire des sorties désirées à des instants désirés sont appelées « *Trajectory Learning* ». Ces algorithmes permettent de faire apprendre au réseau une certaine trajectoire spatio-temporelle. Nous avons présenté les deux techniques les plus utilisées au chapitre III : la technique de rétropropagation du gradient de l'erreur dans le temps *BPTT* et le *RTRL*. La complexité temporelle du *RTRL* peut toutefois le rendre extrêmement lourd. Quant à l'algorithme *BPTT*, le réseau récurrent est transformé en un réseau feedforward par dépliement. Cette phase de dépliement peut rendre la technique de la rétropropagation du gradient très lourde et gourmande en ressource informatique. Les deux techniques ne garantissent cependant pas le suivi du gradient total de l'erreur de toute une séquence d'apprentissage puisque la trajectoire suivie par le réseau dans l'espace d'état dépend des modifications apportées aux poids à chaque instant. Un autre aspect qui nous semble très important est directement lié à la différence structurelle qui existe entre les architectures Globalement Récurrente Globalement Feedforward et Localement Récurrente Globalement Feedforward (*GRGF* et *LRGF*). Dans une architecture *GRGF*, une erreur de sortie du réseau

de neurones peut être réinjectée à l'entrée du réseau et donc avoir des répercutions sur les nouvelles réponses du réseau (par exemple les architectures de Jordan, Elman, Moakes, Miyoshi présentées au chapitre III). Par contre, dans un réseau *LRGF*, la récurrence n'est tolérée qu'au sein du neurone lui même. Le réseau est donc vu globalement comme un réseau Feedforward. Les erreurs de sortie du réseau ont donc de moindres répercutions que pour un réseau *GRGF*. Nous allons voir dans cette partie que, grâce à la récurrence locale, la mémoire dynamique du *RRFR* permet de transformer le problème de reproduction de séquences en un problème d'interpolation linéaire.

Pour faire apprendre au réseau *RRFR* à reproduire des séquences temporelles réelles, nous allons exploiter son comportement dynamique. Grâce à la récurrence locale des neurones d'entrée, le réseau *RRFR* est capable de garder une trace d'une excitation externe, soit indéfiniment dans le temps (pour un comportement de mémorisation) soit provisoirement (pour un comportement d'oubli). Nous allons exploiter ce comportement d'oubli de manière à transformer le problème de reproduction de séquences temporelles en un simple problème d'interpolation linéaire. En effet, si l'on prend le cas du neurone bouclé[36], son évolution après une excitation externe est la suivante :

$$y(t) = f\left(w_{ii} y(t-1)\right) \qquad [194]$$

$f(.)$ est la fonction d'activation du neurone bouclé (sigmoïde) avec la condition initiale suivante :

$$y(0) = 1 \qquad [195]$$

Pendant cette phase d'évolution de la sortie de la mémoire dynamique, la mémoire statique du réseau *RRFR* transforme le problème de reproduction de séquences temporelles en une combinaison linéaire de fonctions gaussiennes. En d'autres termes, à chaque instant $t \geq 0$, la sortie du réseau *RRFR* est :

$$h\left(y(t)\right) = \sum_{n=1}^{N} w_n \phi\left(\left\| y(t) - \mu_n \right\|\right) \qquad [196]$$

où N représente le nombre de prototypes. Le vecteur de pondération $\left[w_n\right]_{n=1,\ldots,N}$ devrait être calculé de manière à avoir $h\left(y(t)\right) = \zeta(t)$, avec $\zeta(t)$ qui représente la sortie désirée de la séquence temporelle à l'instant *t*. La représentation matricielle de l'apprentissage du réseau *RRFR* sur toute l'évolution de la séquence de durée *T*, est la suivante :

$$\mathbf{\Phi.w} = \boldsymbol{\zeta} \qquad [197]$$

[36] Le raisonnement est tout à fait similaire pour le neurone à retour local de l'activation.

La valeur maximale de la longueur admissible de la séquence temporelle que le réseau est capable d'apprendre est égale à la longueur maximale de la mémoire dynamique. Celle-ci est égale[37] à environ 300 unités de temps avec $bw_{ii} = 2$. En se plaçant donc dans ces conditions, la solution de l'équation [197] est obtenue par l'inversion matricielle suivante :

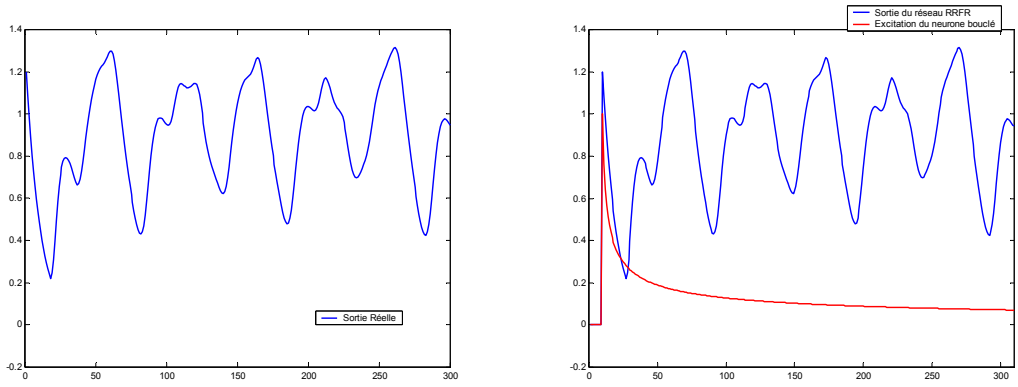
$$\mathbf{w} = \mathbf{\Phi}^{-1}.\boldsymbol{\zeta} \qquad [198]$$

Nous avons établi deux tests du réseau *RRFR* sur l'apprentissage de la séquence chaotique Mackey-Glass et la séquence de commande du problème du four à gaz précédent.

V.4.1. Reproduction de la série Chaotique Mackey-Glass

Le premier test du réseau *RRFR* est celui d'apprendre à reproduire une séquence finie de la série chaotique Mackey-Glass. Le but recherché ici est d'apprendre parfaitement à faire une association *entrée-sortie* du réseau de neurones. En d'autres termes, on force délibérément le réseau à se trouver en situation de sur-apprentissage. Tous les points d'entrée au réseau seront donc mémorisés en tant que prototypes. Le calcul des rayons d'influence est réalisé par la technique du *RCE*. La matrice $\mathbf{\Phi}$ étant obtenue et connaissant le vecteur de sortie $\boldsymbol{\zeta}$, il ne reste plus qu'à déduire le vecteur de pondération $\mathbf{w}$ par l'équation [198]. La figure ci-dessous montre les résultats de l'apprentissage du réseau *RRFR* d'une séquence d'une longueur de 300 unités de temps. La figure de gauche présente les données réelles de la série temporelle. Ces données ont servi à constituer le vecteur de sortie désirée $\boldsymbol{\zeta}$ pour la phase d'apprentissage. La figure de droite présente la sortie du réseau *RRFR* après apprentissage. Cette sortie correspond à l'évolution de la réponse du réseau *RRFR* face à l'évolution temporelle du neurone bouclé (en rouge). Ainsi, à chaque excitation externe, le réseau évoluera librement en reproduisant les 300 valeurs de la série temporelle apprise. L'erreur de reproduction de la série temporelle est quasiment nulle.

[37] *D'après l'étude faite au chapitre précédent.*

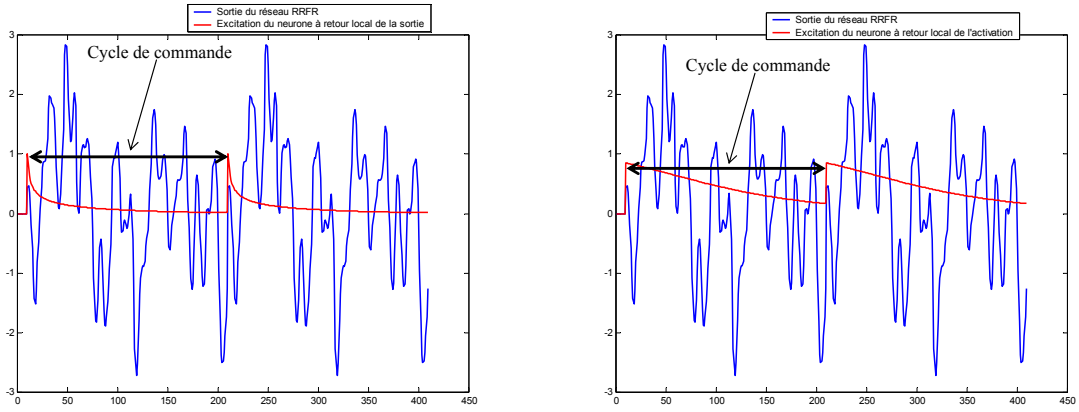


-a- *-b-*

Figure 111. *Apprentissage du réseau RRFR à reproduire les 300 premières valeurs de la série Mackey-Glass, a) données réelles de la série, b) sortie du réseau RRFR avec l'évolution de la sortie du neurone bouclé après excitation.*

V.4.2. Apprentissage d'une trajectoire de commande

Nous avons effectué le même test du réseau *RRFR* sur la reproduction d'une séquence de commande *u(t)* de l'application du four à gaz, décrite précédemment. Nous avons testé cette fois-ci les deux types de mémoires dynamiques : neurone à retour local de la sortie et neurone à retour local de l'activation. Les résultats du test sont présentés sur la Figure 112 : celle de gauche présente les résultats du *RRFR* avec neurone bouclé, et celle de droite, le réseau *RRFR* avec neurone à bouclage de l'activation. Les deux types de mémoires arrivent à reproduire remarquablement la séquence de commande à chaque excitation de la mémoire dynamique (à chaque cycle de commande par exemple).



-a- *-b-*

Figure 112. *Apprentissage du réseau RRFR de la commande u(t) du problème du four à gaz, a) RRFR avec neurone bouclé, b) RRFR avec neurone à retour local de l'activation.*

V.5. Conclusion

Dans ce chapitre nous avons testé les propriétés dynamiques du réseau *RRFR* sur les trois types d'application des réseaux de neurones en surveillance industrielle dynamique : *la reconnaissance de séquences temporelles*, *la prédiction temporelle* et *la reproduction de séquences temporelles*. Tout au long de ces tests, nous avons comparé les deux types de mémoire dynamique du réseau *RRFR* afin d'évaluer les performances de chacune d'elles. Après tous les tests effectués, nous avons constaté que leurs performances sont comparables. Nous choisissons d'adopter le neurone bouclé : neurone à retour local de la sortie. La raison essentielle de ce choix est en grande partie due à la différence structurelle entre les deux neurones récurrents. En effet, la récurrence du signal se situe avant la non-linéarité de la fonction d'activation pour le neurone à retour local de l'activation, et se situe après cette non-linéarité pour le neurone à retour local de la sortie. Cette différence de structure permet au neurone à retour local de la sortie d'intégrer la non-linéarité des données d'entrée dans sa prise en compte de la dynamique de ces données, puisque la fonction d'activation sigmoïde se trouve à l'intérieur du cycle de récurrence du neurone.

Grâce à la récurrence locale au niveau des neurones de la couche d'entrée, le *RRFR* profite de toute la simplicité du processus d'apprentissage procurée par sa partie statique (réseau *RFR* classique) et les performances et simplicités d'utilisation des architectures *LRGF*. Le premier test montre que le réseau *RRFR* est capable d'apprendre des séquences d'un Système à Evénements Discrets avec la simplicité de l'algorithme *RCE*. Sa mémoire dynamique lui permet également de caractériser une séquence réelle (palier de dégradation). Le réseau *RRFR* est capable de dissocier plusieurs paliers de dégradation et aussi d'écarter les pics de fausses alarmes. Une dégradation précoce d'un paramètre d'un équipement à surveiller peut alors être détectée avant que le signal n'ait atteint le seuil d'alarme.

Le deuxième test du *RRFR* sur la prédiction temporelle a été concluant. Deux exemples ont servi de test et validation des propriétés dynamiques du *RRFR* obtenues grâce à la récurrence locale des connexions. L'apprentissage du réseau *RRFR* se compose de deux partie : une partie qui permet de paramétrer la mémoire dynamique du réseau (couche d'entrée). Celle-ci est effectuée *a priori* afin d'avoir une mémoire dynamique la plus longue possible. La deuxième partie concerne la phase de paramétrage de la mémoire statique (paramètres des neurones gaussiens qui sont les prototypes et les rayons d'influence). Ces paramètres doivent être calculés minutieusement si l'on veut garantir une bonne généralisation au réseau de neurones. Nous avons proposé une version améliorée de la technique des *k-moyennes* qui permet de déterminer les centres les plus représentatifs d'une population d'apprentissage garantissant ainsi une bonne généralisation au réseau *RRFR*.

Le troisième test du réseau *RRFR* concerne le problème de reproduction de séquences temporelles. Contrairement aux autres architectures de réseaux récurrents, le réseau *RRFR* transforme un problème de reproduction de séquences en un problème d'interpolation linéaire. Dans ce type d'application, l'objectif recherché est d'apprendre à reproduire

correctement toute une séquence donnée. L'ensemble des points d'apprentissage de la séquence est alors mémorisé par le réseau de neurones. Le réseau *RRFR* est donc capable de reproduire remarquablement des séquences temporelles, à condition que la longueur de ces séquences temporelles soit inférieure à la longueur de la mémoire dynamique du réseau *RRFR*. Une telle précision de reproduction de séquences temporelles est pratiquement impossible à obtenir avec des architectures globalement récurrentes. L'apprentissage de ces réseaux pour ce type d'application est souvent une opération très complexe.

Pour conclure ce chapitre, nous pouvons mettre en évidence deux contributions. La première contribution est la proposition d'une architecture d'un réseau de neurones récurrent (*RRFR*) avec, d'une part, des performances dynamiques intéressantes et, d'autre part, une simplicité d'utilisation et de paramétrage. L'architecture proposée s'est montrée efficace dans les trois applications de surveillance industrielle dynamique à savoir la reconnaissance de séquences temporelles, la prédiction temporelle et la reconstitution de séquences temporelles. Nous verrons au chapitre suivant que l'architecture proposée peut être exploitée pour le développement d'un outil de surveillance en temps réel entièrement paramétrable à distance via le Web.

Notre deuxième contribution concerne la proposition d'une version améliorée de l'algorithme des *k-moyennes* pour le calcul des paramètres de la mémoire statique du réseau *RRFR*. Nous avons identifié et testé les faiblesses de la version simple de l'algorithme des *k-moyennes* qui sont : l'initialisation du nombre *k* des centres, l'initialisation aléatoire des *k* centres et le calcul des rayons des prototypes. Nous avons proposé une version qui permet de booster la version classique. Une initialisation par l'algorithme *Fuzzy Min-Max* permet de déterminer les *k* centres de telle sorte à ce que l'algorithme converge vers une zone de bonne généralisation. On garantit ainsi une certaine optimalité de l'apprentissage avec une stabilité du résultat. Nous avons appliqué cette technique uniquement sur la partie prédiction temporelle à cause du nombre important des points d'apprentissage. La technique proposée peut très bien être appliquée pour déterminer les paramètres du réseau *RRFR* en particulier et le *RFR* en général, et ceci pour n'importe quel problème d'apprentissage où l'on cherche à garantir une bonne capacité de généralisation au réseau de neurones, à condition que la base d'apprentissage soit conséquente.

Après tous ces tests et évaluations du réseau *RRFR* avec sa mémoire dynamique et son algorithme d'apprentissage, nous allons présenter au dernier chapitre une solution originale d'exploitation du réseau *RRFR* pour la surveillance en temps réel embarquée. Le réseau *RRFR* sera restructuré en une architecture capable d'être chargée dans un Automate Programmable Industriel pour profiter du traitement temps réel ainsi que de la fiabilité du matériel. L'apprentissage s'effectuera à distance via le coupleur Web de l'automate.

Chapitre VI

Développement d'un système de surveillance temps réel accessible à distance par un serveur WEB

Résumé : *L'externalisation de la maintenance commence à prendre une certaine ampleur au sein des entreprises soucieuses de réduire les coûts de la maintenance. Les avantages de cette externalisation sont entre autres : d'un côté une meilleure connaissance du budget de la maintenance donc une meilleure maîtrise des coûts, et d'un autre côté pouvoir se recentrer sur son véritable métier, en confiant cette fonction à des professionnels pouvant ainsi assurer une maintenance distante des moyens de production. La solution que nous développons dans ce dernier chapitre s'encadre tout à fait dans ce contexte de e-maintenance. Le réseau RRFR développé dans nos travaux de recherche est ainsi structuré en un programme évolutif en langage automate assurant ainsi une surveillance dynamique en temps réel. Le choix d'une architecture LRGF comme mémoire dynamique d'un réseau RFR est très avantageux pour une telle implémentation. L'apprentissage du réseau RRFR est entièrement géré à distance, par connexion TCP/IP.*

Abstract : *The maintenance externalization becomes an important management technique for the industrial companies which want to reduce their maintenance costs. The important advantages of this type of management are: first the company has better knowledge of the maintenance budget so a better cost control, and second the company is able to focus to its primary goal by entrusting the maintenance to a professional company. The solution that we develop in this last chapter turns completely in this e-maintenance context. The RRBF network developed in our research is structured in an evolutionary program interpreted in a PLC language. The choice of the LRGF architecture as a dynamic memory of the RBF network is very advantageous for such an implementation. The learning of the RRBF network is completely managed by a TCP/IP connection.*

Chapitre VI

Développement d'un système de surveillance temps réel accessible à distance par un serveur WEB

VI.1. Introduction

L'étude effectuée tout au long de ce rapport nous a mené à proposer une architecture de réseau de neurones dynamique (*RRFR*) pour des applications de surveillance industrielle. Nous avons montré quels peuvent être les avantages d'utiliser un réseau de neurones par rapport aux autres techniques (qui se basent sur l'existence d'un modèle formel de l'équipement, comme les techniques de l'automatique) afin de remplir les fonctions de détection et de diagnostic des défaillances. Notre choix du type de réseau de neurones s'est porté sur les Réseaux à Fonctions de base Radiales pour leur flexibilité et leur facilité d'utilisation. Nous nous sommes également penché sur l'étude des réseaux de neurones temporels car la prise en compte de la dynamique des signaux de surveillance (le passé d'un signal capteur) est un critère important en surveillance. Notre choix s'est porté sur les architectures *LRGF* non seulement pour leur simplicité d'apprentissage et de mise en œuvre mais également pour l'efficacité de leur aspect dynamique. Plus particulièrement, nous avons greffé une mémoire dynamique à base de neurones à retour local de la sortie au réseau *RFR* statique. Au chapitre précédent nous avons testé avec succès le réseau *RRFR* sur les trois types d'application des réseaux de neurones pour la surveillance dynamique industrielle. Dans ce dernier chapitre, nous allons montrer que l'architecture *RRFR* proposée (Figure 113) peut se montrer très intéressante de par sa simplicité pour une exploitation de surveillance d'équipements industriels en temps réel entièrement paramétrable à distance via le Web. En effet, nous avons développé une solution originale concernant l'implémentation du réseau *RRFR* dans un Automate Programmable Industriel (API). Cette solution de surveillance neuronale embarquée permet de profiter de la rapidité du traitement des données et de la sûreté de fonctionnement de l'automate industriel.

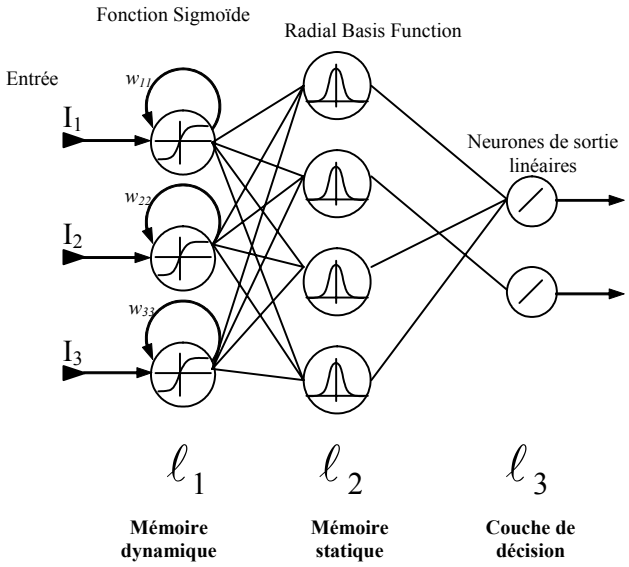


Figure 113. *Réseau Récurrent à base de Fonctions Radiales avec ses deux mémoires (dynamique et statique) embarqué dans l'automate programmable.*

Cette partie applicative s'encadre dans le domaine de la *e-maintenance*, en faisant appel aux supports modernes de communication ainsi qu'en donnant plus d'autonomie et d'intelligence aux équipements de surveillance temps réel. L'étude a été développée en collaboration avec une entreprise de la région de Besançon : *AVENSY Ingénierie*. En se spécialisant dans le domaine du développement et de la mise en oeuvre d'automates programmables serveurs Web, *AVENSY Ingénierie* exploite le marché du suivi de production et celui de la gestion de la maintenance. Son originalité réside dans l'exploitation du concept « *coupleur Ethernet-TCP/IP* » présent dans les configurations modernes d'automates programmables industriels. *AVENSY Ingénierie* offre donc des solutions de suivi de production et de supervision avec un traitement temps réel et un accès distant à l'information à travers le support de communication TCP/IP.

Le coupleur automate joue donc le rôle d'un serveur Web et permet d'établir une jonction entre les protocoles de communication Internet (TCP/IP) comme le HTTP et le programme automate. Cette dimension *« accès à l'information à distance »* permet à un expert distant de suivre l'évolution de plusieurs variables du programme automate à travers une Interface Homme Machine (IHM). Ce suivi peut être fait d'une manière passive (opération de lecture des variables du programme automate) ou active (opération d'écriture des variables du programme automate).

C'est dans ce contexte que nous avons voulu apporter notre contribution pour une nouvelle opportunité d'exploitation du marché de la surveillance industrielle en temps réel. Nous avons donc opté pour la même démarche que la société AVENSY, c'est-à-dire avoir un accès à l'information à distance avec une surveillance en ligne temps réel effectuée par un automate.

Nous avons dû pour cela « *décomposer* » le réseau *RRFR* en fonctions élémentaires pour qu'il soit interprétable en langage automate. En effet, le langage de programmation automate (step7 pour les API Siemens) représente un langage de bas niveau et nécessite une certaine restructuration du réseau *RRFR*.

La solution que nous avons développée comporte deux parties :

- *Une partie surveillance temps réel en ligne* : le réseau *RRFR* chargé dans l'unité centrale de l'automate traite les variables de surveillance et donne sa réponse en temps réel,
- *Une partie apprentissage et affichage des résultats de surveillance à distance* : le réseau de neurones chargé dans l'automate devra être entièrement mis à jour par un expert distant via le coupleur web. L'aspect ***évolutif*** du programme neuronal en langage automate est donc un critère important dans la mise en oeuvre de notre solution de surveillance en ligne. Le deuxième critère est que le programme soit utilisable par un grand nombre de clients, cela pour minimiser les frais de main d'œuvre de développement. L'aspect ***flexibilité*** du programme neuronal est donc un deuxième critère important à prendre en compte par notre solution.

Un premier prototype de cette maquette a été présenté lors d'un salon sur les microtechniques[38]. L'intérêt de la société *AVENSY Ingénierie* étant de commercialiser cette solution de surveillance neuronale temps réel, les deux premières étapes étaient bien évidemment le dépôt d'un brevet et le lancement d'une étude de marché.

Ce chapitre est structuré essentiellement en deux parties. Dans la première partie, nous décrivons les différentes étapes qui nous ont conduits au choix de la solution proposée. Après plusieurs concertations avec notre partenaire, nous avons abouti à une solution qui semble s'adapter aux besoins actuels des entreprises industrielles. La deuxième partie de ce chapitre est consacrée à la description de la solution retenue, en y associant une série de tests de performance.

VI.2. Les différentes étapes du développement de la solution de surveillance dynamique en temps réel

Avant de décrire la genèse de la solution proposée, nous allons rappeler les conditions que doit remplir la configuration finale de surveillance imposées par *AVENSY Ingénierie*. Une sorte de cahier des charges a donc été établi dont le contenu est :

- de pouvoir assurer une surveillance en ***temps réel***,

[38] *14ème SALON INTERNATIONAL DES MICROTECHNIQUES, MICRONORA, Besançon 2002.*

- d'avoir une architecture ***flexible*** et ***évolutive***,
- d'avoir une ***simplicité d'utilisation*** et de ***paramétrage***,
- d'être entièrement paramétrable à ***distance*** (apprentissage à distance)

La configuration finale de la solution de surveillance temps réel présentée dans ce chapitre est l'aboutissement de toute une succession de configurations qui se sont montrées inintéressantes et, surtout, qui ne répondaient pas aux exigences citées précédemment. La première configuration la plus naturelle était de délocaliser la surveillance dans un ordinateur distant. Les variables de surveillances sont récupérées par le coupleur Web et transférées vers l'ordinateur distant via la couche TCP/IP. Les raisons pour lesquelles nous avons écarté cette configuration étaient, d'une part, l'impossibilité d'effectuer du traitement temps réel et, d'autre part, sa grande vulnérabilité aux perturbations pouvant affecter le système (perturbations du réseau TCP/IP ou du système d'exploitation de l'ordinateur). Contourner ces obstacles était bien évidemment de localiser le traitement de surveillance au niveau de l'automate. C'est alors que se posait la question sur la façon de mener l'apprentissage. Trois solutions sont dans ce cas possibles : le programme d'apprentissage peut être localisé soit au niveau d'un ordinateur distant, soit au niveau de l'unité centrale de l'automate, soit enfin au niveau du coupleur Web de l'automate. Nous avons opté pour la dernière solution, c'est-à-dire localiser le programme d'apprentissage du réseau de neurones au niveau du coupleur Web. Toutes ces étapes qui nous ont menées à la solution finale sont présentées en détail dans cette partie avec les limites de chaque configuration.

VI.2.1. Première implantation

La première idée d'application était de concevoir un outil de surveillance neuronal implémenté dans un ordinateur personnel dédié seulement au traitement de surveillance. Ce programme peut être réalisé grâce aux outils de programmation comme MATLAB, VisualC++ ou autres. Les variables de surveillance sont récupérées par la CPU de l'automate via le réseau local industriel[39] (Figure 114). Ces variables sont ensuite acheminées vers le PC distant grâce au coupleur WEB de l'automate. Ce dernier représente un serveur Web embarqué dans l'automate. Une couche TCP/IP permet de gérer la communication entre le coupleur et le PC distant. Les données de surveillance sont donc acheminées vers le PC par un format de fichier XML[40]. Le programme de surveillance neuronal traite alors le fichier de données XML. Il renvoie à son tour la réponse de la surveillance sous le même format de fichier vers le coupleur WEB de l'automate. Une Interface Homme Machine (IHM) programmée sous JAVA chargée dans le coupleur WEB, permet à un expert distant d'avoir un

[39] Dans notre application nous avons utilisé le réseau local industriel de SIEMENS appelé ***Profibus***.
[40] Ce format de fichier représente un format de donnés pour les protocoles TCP/IP.

suivi permanant de l'évolution de l'équipement. La Figure 114 montre le schéma général de cette configuration de surveillance.

Cette configuration de surveillance présente néanmoins quelques faiblesses qui sont :

- *Lenteur du traitement* : le temps de réaction d'une telle configuration est tributaire de tout un cycle de cheminement des données de surveillance :
  - Connexion du PC au Coupleur WEB par le réseau TCP/IP.
  - Envoi des données par le serveur WEB de l'automate vers le PC,
  - Traitement des données de surveillance par le programme neuronal,
  - Envoi des résultats de surveillance vers le serveur WEB de l'automate.

  Il est évident qu'une telle configuration ne permet pas d'avoir une surveillance de l'équipement industriel en temps réel.
- *Instabilité de l'application* : ce type d'application est très sensible aux perturbations pouvant affecter le système qui sont essentiellement de deux types :
  - Perturbation du réseau TCP/IP. Une rupture de connexion pour plusieurs raisons a pour conséquence l'arrêt du processus de surveillance. Ceci peut provoquer des effets indésirables et dangereux pour l'équipement à surveiller.
  - Perturbation au niveau du système d'exploitation du PC. En effet, les systèmes d'exploitation, même les versions industrielles, ne sont pas garantis à 100% contre un dysfonctionnement impromptu.

Suite à ces constatations, nous avons décidé d'abandonner cette configuration pour nous pencher plutôt sur une deuxième idée qui est présentée au paragraphe suivant.

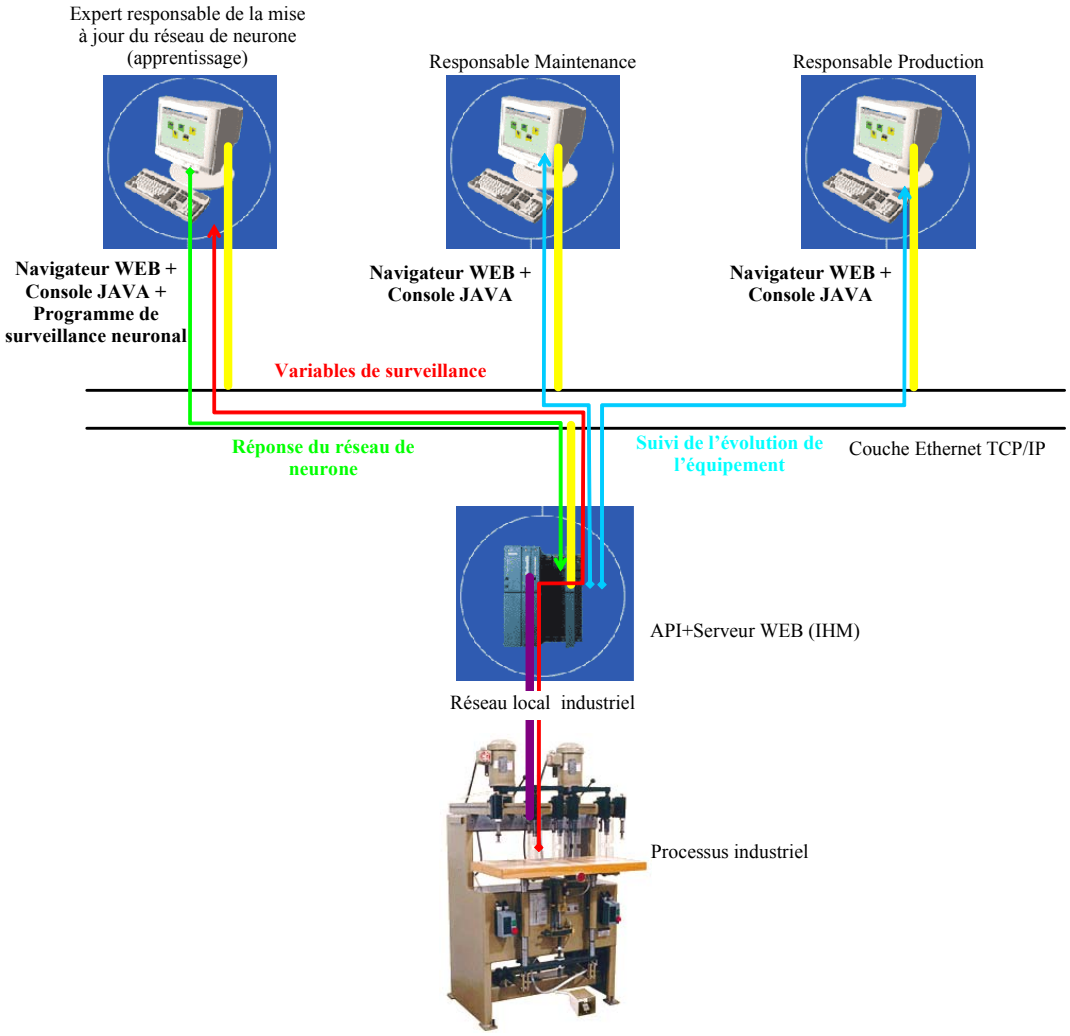


Figure 114. *Schéma général de l'implémentation initiale du réseau RRFR pour une surveillance à distance.*

VI.2.2. Deuxième implantation

Afin de contourner les deux problèmes cités précédemment, nous avons opté pour une solution de surveillance embarquée au niveau de l'automate. Le programme de surveillance neuronal ne sera plus exécuté au niveau du PC distant mais chargé dans la CPU de l'automate. Les variables de surveillance seront dans ce cas traitées localement au niveau de l'API. Une surveillance en temps réel est donc possible. Le système pourra même agir sur l'équipement en cas de panne dangereuse (défaillance critique). L'outil que nous proposons représente un outil de surveillance et d'aide à la décision. Le coupleur WEB permet dans ce cas non seulement d'avoir une Interface Homme Machine distante, mais également de mettre à jour le

réseau de neurones depuis une connexion TCP/IP : ***apprentissage à distance***. Les deux programmes (IHM + Apprentissage) sont programmés en langage *JAVA*. Le choix de ce type de langage de programmation est motivé essentiellement par les deux avantages suivants :

- *Le JAVA est un langage distribué* : le langage JAVA possède une importante bibliothèque de routines permettant de gérer les protocoles de communication TCP/IP.
- *Le JAVA est un langage multi plateforme* : le langage JAVA possède également une bibliothèque d'exécution qui se veut indépendante de la plate-forme (ou du système d'exploitation du PC).

La Figure 115. montre l'architecture générale d'une telle configuration avec les différents postes distants. Une base de données mémorisant toute l'évolution du système à surveiller est créée et mémorisée dans le coupleur WEB de l'API. Cette base de données contient les variables de surveillance horodatées ainsi que les différentes réponses du réseau de neurones. Un expert distant peut ainsi se baser sur cet historique pour mettre à jour les paramètres du réseau de neurones embarqué.

Pour aboutir à cette configuration, trois choix étaient alors possibles (comme le montre le Tableau 13) :

	Solution 1		**Solution 2**		**Solution 3**	
	Soft	*Hard*	*Soft*	*Hard*	*Soft*	*Hard*
Programme de surveillance	Step 7	API	Step 7	API	Step 7	API
Processus d'apprentissage	Matlab ou VisualC++	PC Dédié	Step 7	API	JAVA	Coupleur WEB
Interface Homme Machine	JAVA	Coupleur WEB	JAVA	Coupleur WEB	JAVA	Coupleur WEB

Tableau 13. *Les différentes possibilités de mise en œuvre d'une application de surveillance en ligne.*

- *Solution 1* : La première solution était de garder la phase d'apprentissage au niveau d'un PC distant en utilisant des supports de programmation comme Matlab ou VisualC++. Un expert humain récupère alors les données d'apprentissage par le réseau TCP/IP. Après calcul des nouveaux paramètres du réseau de neurones, ces derniers doivent être chargés dans l'automate. Pour ce faire, le réseau de neurones sous forme de langage Step7 doit être restructuré avec les nouveaux paramètres et rechargé de nouveau dans l'automate avec une console de programmation step7. Celle-ci doit avoir une liaison directe avec l'automate par réseau local industriel. L'expert doit alors se déplacer au niveau de l'équipement. Nous avons

écarté cette solution pour cause de lourdeur de mise à jour du réseau de neurones embarqué dans l'automate. A chaque mise à jour, l'expert doit reprogrammer le réseau de neurones avec le langage step7 et le recharger localement dans l'automate avec une console de programmation. Cet expert devra certainement avoir quelques connaissances en réseaux de neurones afin de pouvoir interpréter en langage step7[41] les paramètres d'apprentissage. Ceci complique considérablement la mise en œuvre d'une telle application dans un environnement industriel.

- *Solution 2* : La deuxième solution est de vouloir embarquer la procédure d'apprentissage dans l'automate. L'objectif de cette solution est de rendre le réseau de neurones capable d'apprendre en ligne (en temps réel). Cette option est quasi-impossible à obtenir. Le langage de programmation step7 (étant un langage de programmation bas niveau) n'est pas fait pour ce genre de procédure, c'est-à-dire la prise en compte d'un algorithme d'apprentissage tel que la version simple des k-moyennes, et encore moins la version améliorée proposée au chapitre précédent. D'autre part, ceci provoquerait des temps de traitements supplémentaires au niveau de l'automate. Nous avons donc écarté aussi cette deuxième solution.

- *Solution 3* : La troisième solution que nous avons exploitée est celle décrite au début de ce paragraphe (voir Figure 115). La mise à jour du réseau de neurones (apprentissage) est gérée par le programme JAVA localisé au niveau du coupleur WEB. L'expert responsable du paramétrage du réseau de neurones ne sera plus obligé de le reprogrammer en step7 ni même de se déplacer au niveau de l'automate pour le recharger dans la CPU (par réseau local industriel). Toute la phase de mise à jour s'effectue par connexion TCP/IP via le coupleur WEB. La réussite d'une telle configuration nécessite une structure du programme neuronal évolutive et flexible. Au paragraphe suivant, nous allons développer d'avantage cette solution qui semble être la plus intéressante.

[41] Qui est un langage de programmation bas niveau correspondant aux automates SIEMENS.

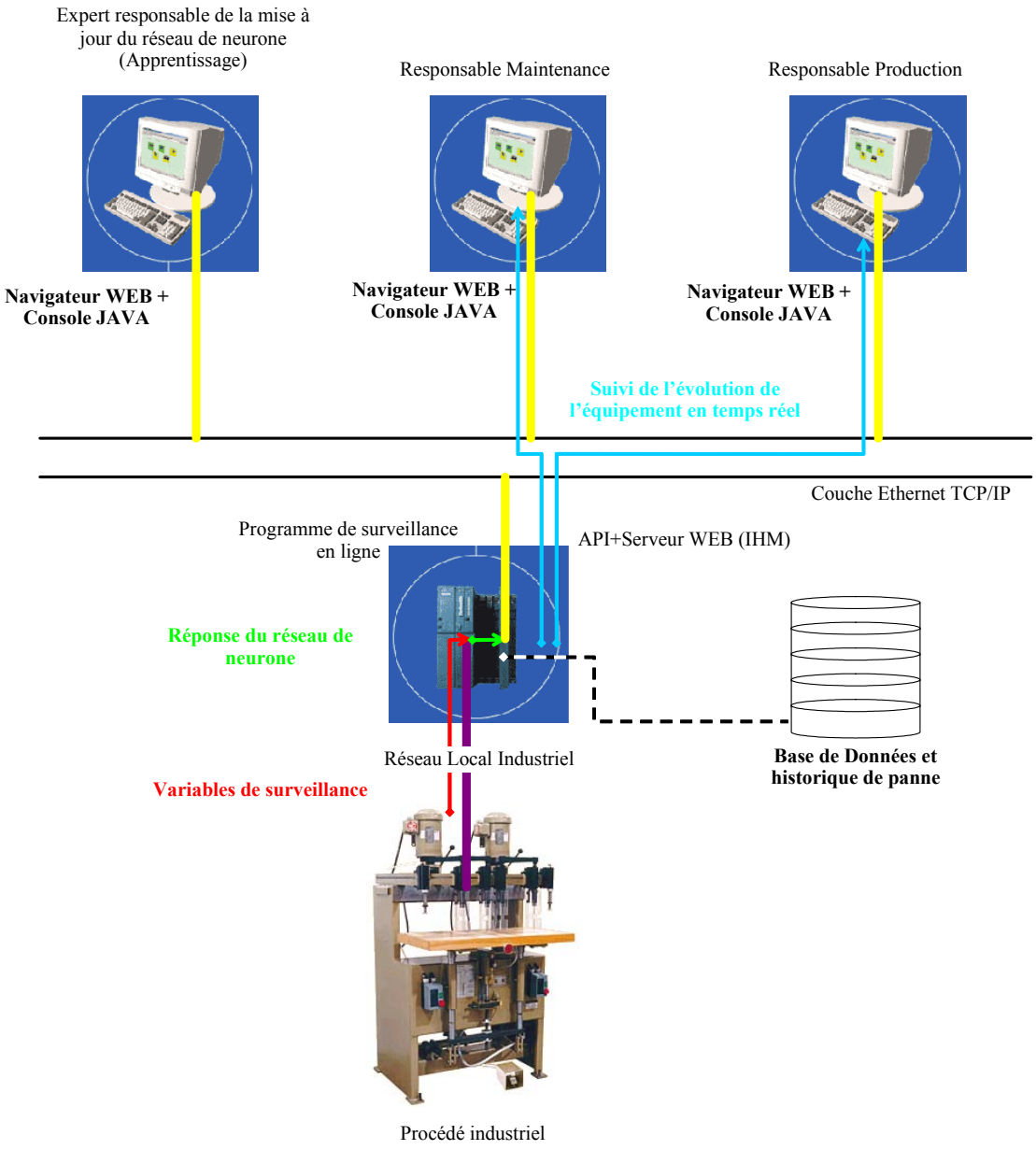


Figure 115. *La solution choisie pour une application de surveillance en ligne avec apprentissage à distance via le protocole de communication Internet TCP/IP. Le programme de surveillance est localisé au niveau de l'unité centrale de l'automate. Le coupleur Web permet d'ouvrir la liaison à un échange de données et d'information entre l'automate qui surveille la machine et un expert distant, à travers une couche TCP/IP. Cet expert peut très bien être situé à l'intérieur de l'entreprise (réseau interne) ou bien à l'extérieur de l'entreprise (réseau Internet).*

VI.3. Description de la solution proposée

VI.3.1. Introduction

L'architecture globale de la solution proposée est illustrée par la Figure 116. Une fois chargée dans la CPU de l'automate, la structure du programme neuronal est capable d'évoluer en étant supervisée par un expert distant. Toute la phase d'apprentissage est gérée via le coupleur WEB par la connexion TCP/IP. Le programme JAVA chargé dans le coupleur permet de faire la liaison entre l'automate et l'expert. Pour ce faire, deux fonctions de base sont nécessaires : une fonction de ***lecture*** et une fonction ***d'écriture***. Ces deux fonctions sont fournies par le constructeur[42] et permettent donc de communiquer avec l'automate soit par réseau interne à l'entreprise (*Intranet*) soit par réseau externe (*Internet*). Les conditions de la structure du programme neuronal en step7 qui nous ont été imposées par notre partenaire industriel sont les suivantes :

- ***Evolutif*** : La structure doit être entièrement évolutive via le coupleur WEB. L'utilisateur n'aura pas à reprogrammer et recharger le réseau de neurones dans la CPU de l'automate. Cette facilité d'utilisation est un critère très important et très apprécié par les industriels qui ne possèdent pas forcément de connaissances sur l'outil neuronal. Le paramétrage du réseau de neurones doit être entièrement transparent à l'utilisateur. Cette solution ouvre des perspectives vers une externalisation de la maintenance via le web. En effet, l'expert distant pourra suivre les différentes évolutions des variables de surveillance ainsi que les réponses du réseau de neurones chargé de la surveillance de l'équipement. Il pourra également le paramétrer par connexion TCP/IP.
- ***Flexible*** : La structure du programme neuronal doit être standard pour un équipement quelconque. C'est-à-dire que le programme doit être capable de s'adapter à un nombre quelconque de variables de surveillance (nombre de sorties capteur) et aussi de s'adapter à plusieurs types de variables de surveillance (entier, double entier, booléen, etc.). Cette contrainte a pour exigence une plus grande ouverture au maximum de clients potentiels à la société AVENSY. Le développement d'un produit générique coûterait moins cher qu'un produit fait sur mesure à un client précis.

[42] Dans notre application, les deux fonctions ont été fournies par *SIEMENS*.

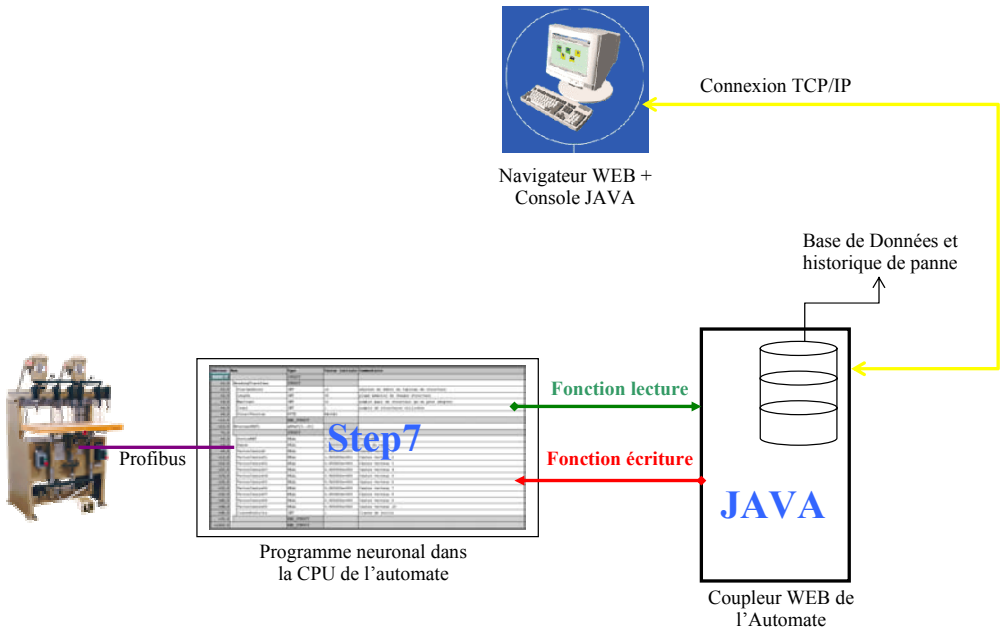


Figure 116. *Les différentes interactions entre chaque composant de l'application de surveillance.*

VI.3.2. Structure du programme neuronal en langage step7

Dans cette partie, nous présentons la structure générale de l'implémentation du réseau *RRFR* dans un Automate Programmable Industriel. Ce programme neuronal doit remplir les deux conditions du cahier des charges citées précédemment, en l'occurrence *évolutif* et *flexible*. Nous ne nous attarderons pas sur le détail du programme step7. Nous donnerons juste le principe de base général du programme pour la bonne compréhension de son fonctionnement. Nous nous sommes basés essentiellement sur deux composants[43] de programmation pour l'implémentation en step7 du réseau *RRFR* :

- Un composant permet de définir des fonctions de calcul type comme le calcul de la sortie du neurone bouclé en fonction de ses entrées ou encore celle du neurone gaussien. Le programme principal fera appel à ces fonctions pour le calcul des différentes sorties des neurones. En programme step7, ce composant est appelé (**FC**).
- Un composant permet de stocker tous les paramètres du programme qui sont les différents paramètres du réseau *RRFR*. Ces paramètres seront exploités par les composants FC pour les différents calculs. En langage step7, ce composant est appelé (**DB**).

[43] Nous ne nous attarderons pas sur le détail de ces deux composants. Pour plus d'explication, se référer à la documentation technique du constructeur SIEMENS.

La Figure 117 présente la structure globale du programme neuronal en step7. Chaque couche du réseau de neurones est représentée par une FC et une DB. La communication avec le programme step7 via le coupleur web ne peut se faire que sur des variables se trouvant dans un composant DB. On comprend alors que pour pouvoir accéder à toutes les variables de surveillance ainsi qu'à tous les paramètres du réseau de neurones, il faudrait les stocker à des adresses bien identifiées, localisées dans les différentes DB. Dans la structure step7, un numéro est associé à chaque DB, et chaque variable possède une adresse absolue fixe dans la DB. En localisant ainsi l'emplacement de chaque variable du programme, on peut soit afficher cette variable sur un poste distant, soit changer sa valeur par un expert distant grâce aux fonctions de lecture écriture.

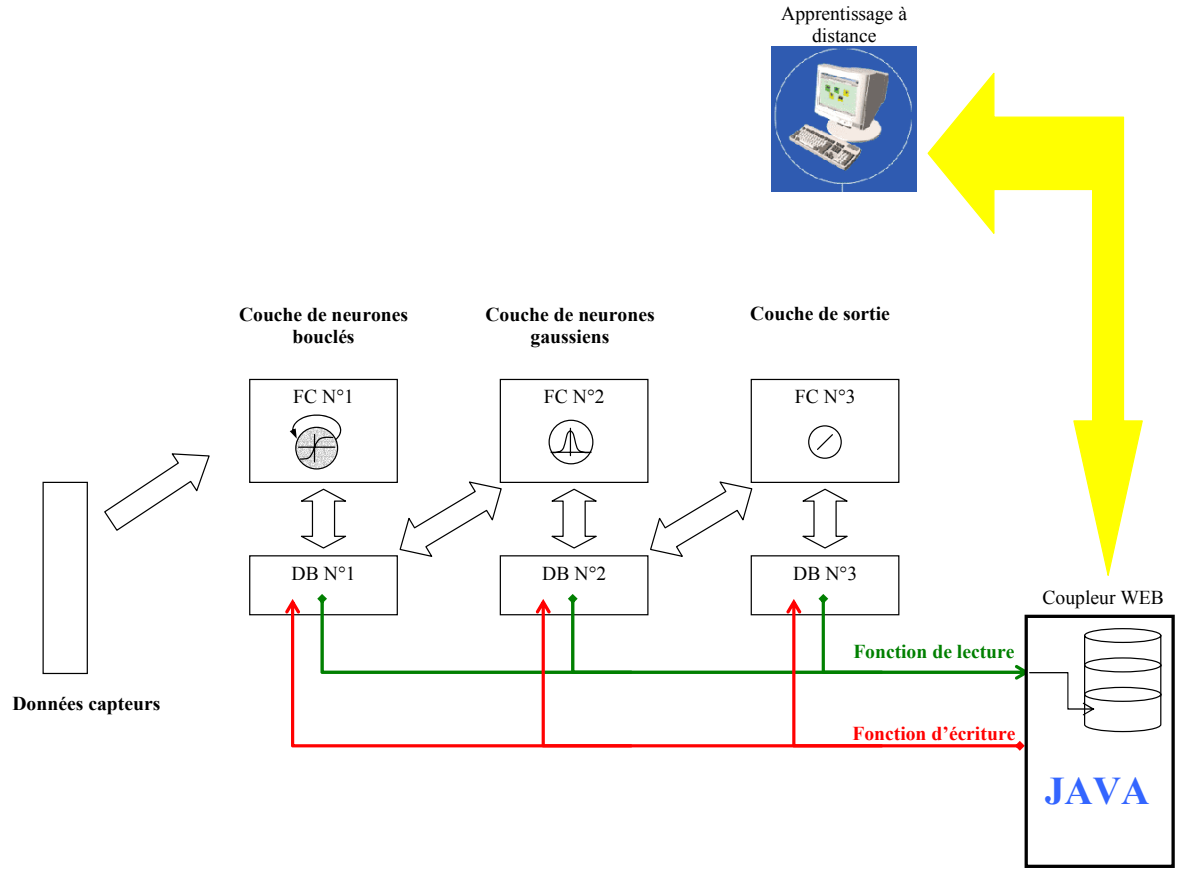


Figure 117. *Organigramme du réseau RRFR tel qu'il a été chargé dans la CPU de l'automate. La communication avec le coupleur Web se fait à travers des variables situées dans le composant appelé DB.*

Les deux critères de *flexibilité* et *d'évolutivité* sont obtenus grâce à l'utilisation de variables de type « ***structure*** » pour chaque DB des différentes couches du réseau de neurones. Chaque variable de type structure contient une entête où figurent certaines informations. Ces informations sont exploitées par des *pointeurs* et servent à identifier l'adresse de chaque

paramètre du réseau de neurones. Par exemple, la Figure 118 présente la DB N°2 de la couche de neurones gaussiens. Les paramètres de cette couche sont :

- le nombre de neurones gaussiens qui composent cette couche,
- le rayon d'influence ainsi que le vecteur centre de chaque prototype,
- la réponse ainsi que la classe de sortie de chaque neurone gaussien.

Ces paramètres sont mémorisés dans une sous-structure de la structure principale. Chaque sous-structure représente les paramètres d'un seul neurone gaussien. L'entête de la structure principale contient :

- l'adresse (en octet) de début de la première sous-structure (premier neurone gaussien) : *StartAddress*,
- la longueur (en octet) de chaque sous-structure : *Length*,
- le maximum de sous-structures que peut contenir toute la structure principale : *MaxCount*,
- le nombre de structures qui ont été créées (nombre de neurones gaussiens) : *Count*.

On peut ainsi retrouver l'adresse absolue de tous les paramètres du $n^{ème}$ neurone gaussien par l'expression suivante :

$$AddressParamNeu(n) = StartAddress + (n-1) * Length + AddressParam \qquad [199]$$

où *AddressParam* représente l'adresse relative de chaque paramètre des neurones. Pour l'exemple de la DB N°2 (voir Figure 118), on a *StartAddress* = 10 et *Length* = 50. Si l'on veut, par exemple, retrouver l'adresse absolue du rayon d'influence du $n^{ème}$ neurone, sachant que son adresse relative est de 4 octets, on obtient :

$$AddressRayNeu(n) = 10 + (n-1) * 50 + 4 \qquad [200]$$

Tous les paramètres du réseau de neurones peuvent alors être localisés avec précision. Des procédures d'incrémentation et d'adressage ont été programmées dans chaque FC avec comme base l'équation [199]. La condition de fin d'incrémentation pour chaque fonction FC est donnée par la valeur *Count* de la DB correspondante. Cette valeur représente le nombre de neurones par couche.

Adresse	Nom	Type	Valeur initiale	Commentaire
0.0		STRUCT		
+0.0	HeadingTraceZone	STRUCT		
+0.0	StartAddress	INT	10	adresse de début du tableau de structure
+2.0	Length	INT	50	plage mémoire de chaque structure
+4.0	MaxCount	INT	21	nombre maxi de structure qu'on peut adopter
+6.0	Count	INT	1	nombre de structures utilisées
+8.0	StructVersion	BYTE	B#16#1	
=10.0		END_STRUCT		
+10.0	NeuronneRBF1	ARRAY[0..20]		
*0.0		STRUCT		
+0.0	SortieRBF	REAL	0.000000e+000	SortieRBF
+4.0	Rayon	REAL	2.000000e+000	carré du rayon
+8.0	VectorCentre0	REAL	5.000000e+000	Centre vecteur 1
+12.0	VectorCentre01	REAL	1.000000e+001	Centre vecteur 2
+16.0	VectorCentre02	REAL	0.000000e+000	Centre vecteur 3
+20.0	VectorCentre03	REAL	0.000000e+000	Centre vecteur 4
+24.0	VectorCentre04	REAL	0.000000e+000	Centre vecteur 5
+28.0	VectorCentre05	REAL	0.000000e+000	Centre vecteur 6
+32.0	VectorCentre06	REAL	0.000000e+000	Centre vecteur 7
+36.0	VectorCentre07	REAL	0.000000e+000	Centre vecteur 8
+40.0	VectorCentre08	REAL	0.000000e+000	Centre vecteur 9
+44.0	VectorCentre09	REAL	0.000000e+000	Centre vecteur 10
+48.0	ClasseDeSortie	INT	1	Classe de sortie
=50.0		END_STRUCT		
=1060.0		END_STRUCT		

Entête de la structure principale

Paramètres d'un neurone gaussien (sous-structure)

Figure 118. *Présentation de la DB N°2 du programme step7. La structure globale est constituée d'une entête et d'un nombre fini de sous-structures qui représentent des neurones gaussiens.*

Si l'on revient à la Figure 117, le fonctionnement de tout le programme en step7 se déroule selon l'algorithme suivant :

1. Le nombre de sous-structures de la DB N°1 est fixé par l'utilisateur grâce au paramètre *Count* de la DB N°1. Ce paramètre représente le nombre de variables de surveillance. Ces sous-structures englobent les paramètres des neurones bouclés qui sont :
    - l'entrée du neurone bouclé (données capteur),
    - paramètres de la fonction d'activation sigmoïde,
    - la valeur de l'auto-connexion,
    - la réponse du neurone bouclé.

2. Début du fonctionnement de la FC N°1
    2.1. Procédure qui récupère les données capteur et les mémorise dans les sous-structures de la DB N°1,
    2.2. Procédure qui calcule la réponse de chaque neurone bouclé en fonction de ses paramètres et de son entrée. Sa réponse est mémorisée à l'adresse correspondante dans la DB N°1.

3. Début du fonctionnement de la FC N°2. Cette fonction calcule la réponse de chaque neurone gaussien en fonction de ses paramètres (mémorisés dans la DB N°2) et des sorties des neurones bouclés (mémorisées dans la DB N°1). Ces réponses sont mémorisées dans les sous-structures de la DB N°2.

4. Début du fonctionnement de la FC N°3. Cette fonction calcule les sorties du réseau de neurones en fonction des réponses des neurones gaussiens. Ces sorties sont mémorisées dans la DB N°3.

VI.3.3. Apprentissage et visualisation des données

La phase de mise à jour du réseau de neurones (phase d'apprentissage) consiste à re-paramétrer le réseau de neurones en fonction de l'évolution de l'équipement. Cette étape est entièrement gérée par le langage de programmation JAVA localisé dans le coupleur Web de l'automate. Nous avons justifié les raisons du choix de ce type de langage de programmation au début de ce chapitre qui sont rappelons le, sa capacité de gérer les protocoles TCP/IP et aussi qu'il soit multi plateforme. Un expert distant responsable de cette mise à jour peut donc modifier les paramètres du réseau à travers une interface de visualisation. La version améliorée de l'algorithme des k-moyennes présentée au chapitre précédent est alors facilement programmable en langage JAVA. Une fois que les paramètres du réseau *RRFR* sont extraits, ces derniers sont chargés dans le programme automate par les fonctions d'écriture avec comme base l'équation [199] qui sert à identifier l'adresse absolue de tous les paramètres.

Comme le montre la Figure 119, une base de données est créée par le programme JAVA. Cette base de données contient des informations qui seront exploitées pour la phase d'apprentissage. Ces informations sont les valeurs des différentes variables de surveillance horodatées (se trouvant dans la DB N°1) ainsi que les différentes réponses du réseau de neurones (dans la DB N°3). Un expert distant peut alors consulter cet historique, analyser les différentes données de cette base et exécuter l'algorithme d'apprentissage. Les paramètres du nouveau réseau de neurones sont alors obtenus. L'expert peut ainsi recharger ces nouveaux paramètres dans la CPU de l'automate par la fonction d'écriture. Les DB concernées par cette phase d'écriture sont les DB N°1 et N°2 où figurent respectivement les paramètres des neurones bouclés et des neurones gaussiens. L'ajout de nouveaux neurones gaussiens devient alors aisé. Il suffit de mettre le nouveau nombre de neurones gaussiens au niveau du nombre de sous-structures contenues dans la structure principale de la DB N°2 (le paramètre *Count* au niveau de l'entête de cette DB - voir Figure 118 -) ainsi que les valeurs de leurs paramètres (prototypes et rayons d'influence) aux adresses correspondantes. Le programme neuronal chargé dans l'automate est ainsi entièrement ***évolutif*** par connexion TCP/IP via le coupleur web.

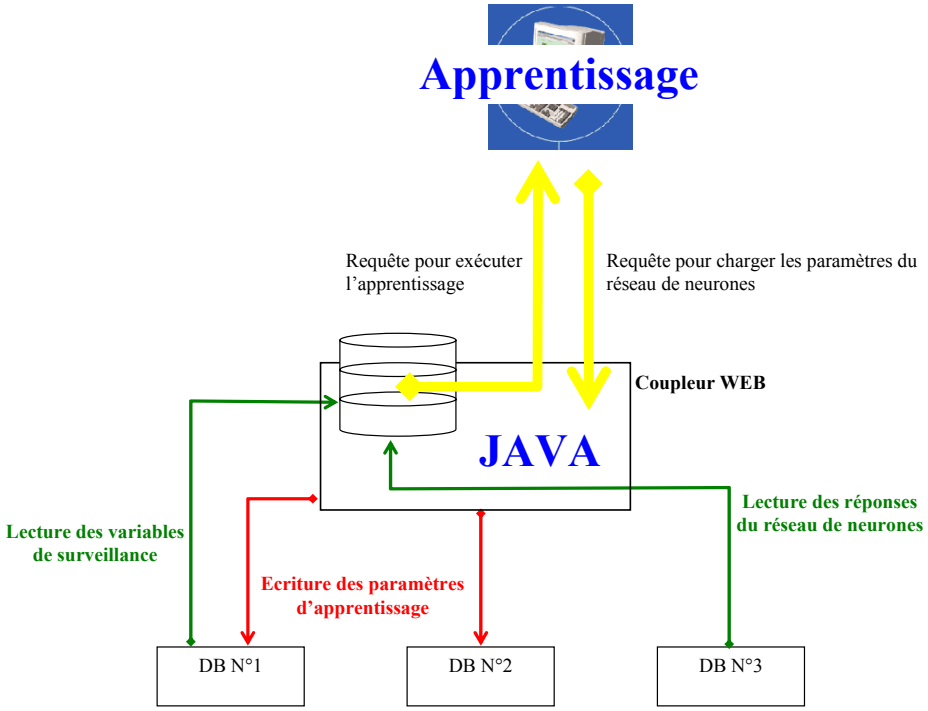


Figure 119. *L'organigramme du processus d'apprentissage du réseau de neurones chargé dans l'automate. Le programme JAVA chargé dans le coupleur peut être exécuté à distance par liaison TCP/IP. Une base de données est également disponible au niveau du coupleur et consultable à distance. Celle-ci contient des informations horodatées concernant les variables nécessaires à l'apprentissage du réseau RRFR. Après avoir déterminé les nouveaux paramètres du réseau RRFR, l'expert distant peut mettre à jour le programme neuronal dans l'automate grâce aux fonctions d'écriture. Les DB concernées par cette phase sont la DB N°1 et N°2 où figurent les informations de la première couche ainsi que la deuxième du réseau RRFR.*

La Figure 120 montre l'interface de visualisation pour un utilisateur passif c'est-à-dire sans pouvoir d'écriture dans l'automate. En d'autres termes, cette interface ne sert qu'à lire les données de surveillance. Un tel degré de sécurité est nécessaire si l'on veut ouvrir l'accès au programme de surveillance à des personnes qui ne sont pas forcément qualifiées pour la phase de paramétrage du programme neuronal (phase d'apprentissage). Cette délicate phase d'apprentissage est gérée par une interface dédiée à un expert responsable de la mise à jour du programme de surveillance (Figure 121). Cet expert est donc autorisé à écrire dans le programme neuronal de l'automate et ainsi paramétrer le réseau *RRFR* à distance.

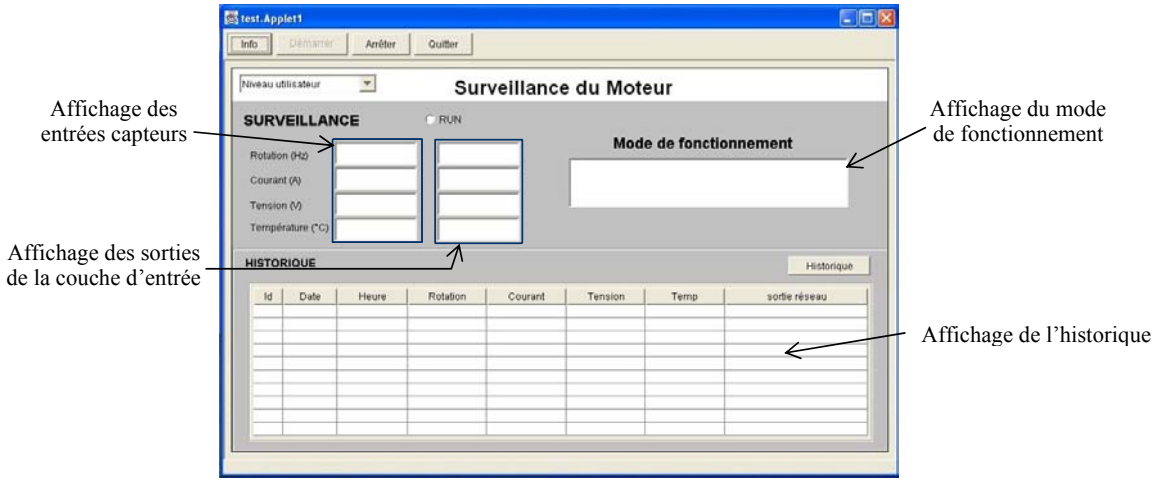


Figure 120. *Interface de visualisation (IHM) pour un utilisateur sans pouvoir (passif). Ce dernier ne peut que visualiser les valeurs des paramètres de surveillance, le mode de fonctionnement ainsi que l'historique de l'équipement.*

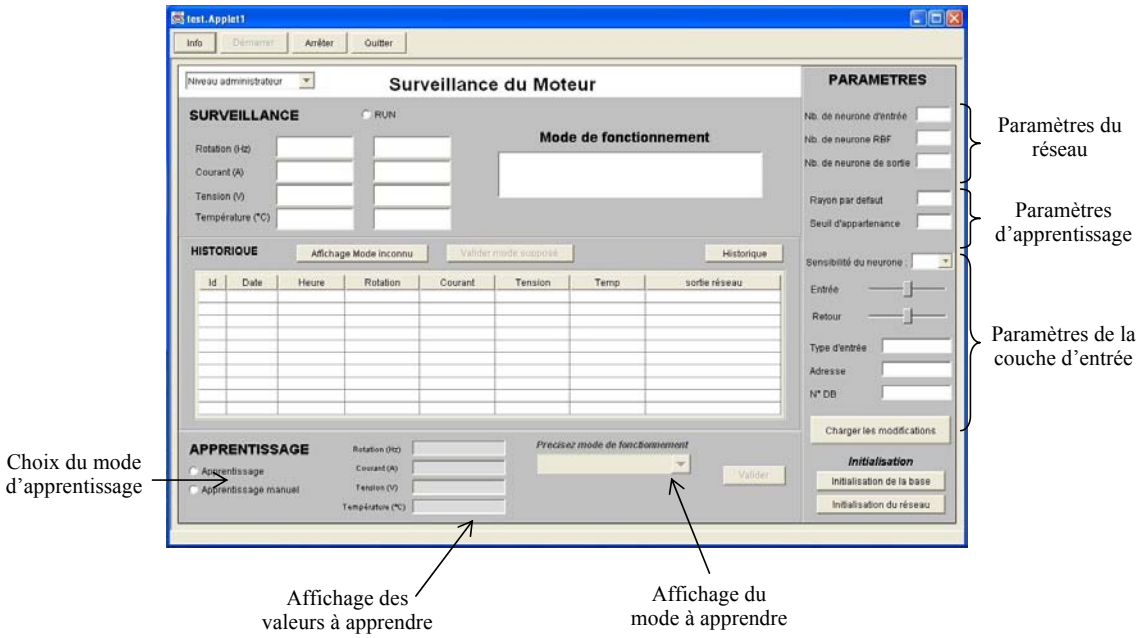


Figure 121. *Interface de visualisation (IHM) pour utilisateur avec pouvoir (expert actif). Ce dernier est la seule personne à pouvoir configurer le programme neuronal chargé dans l'unité central de l'automate. Cet expert peut se trouver au sein même de l'entreprise (réseau local) ou carrément à l'extérieur de l'entreprise.*

VI.4. Evaluation des performances du programme

VI.4.1. Description de la maquette de test

Nous avons évalué et testé les performances du programme neuronal sur une maquette comprenant un automate SIEMENS avec :

- une CPU de type 414-2,
- un coupleur Web de type CP 443-1,
- un variateur de type MICROMASTER 4,
- un moteur à courant continu.

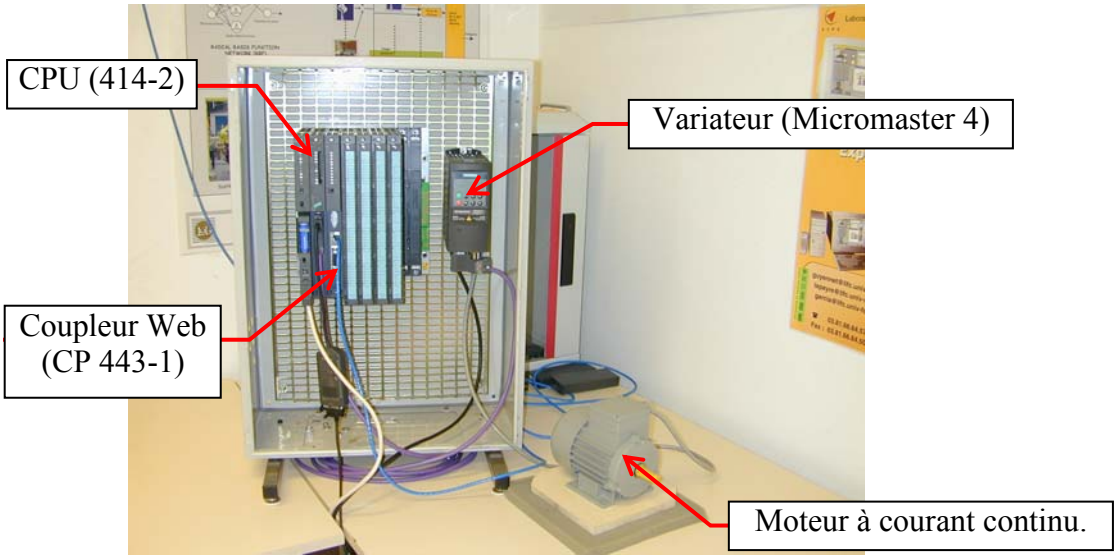


Figure 122. *Maquette de test.*

VI.4.2. Evaluation des temps de cycle

Plusieurs temps de cycles pour plusieurs dimensions de réseaux de neurones ont été évalués sur cette maquette. En fonction des différents temps d'exécution de chaque module du programme neuronal, nous avons pu formaliser ce temps de cycle (Tc) par la relation suivante avec une précision de 5 % :

$$Tc = 0.241 \times Nent + 0.0152 \times Nent \times Nrbf + 0.08 \times Nrbf \quad [201]$$

où *Nent* représente le nombre de neurones bouclés en entrée, *Nrbf* le nombre de neurones gaussiens. Le temps de cycle (Tc) est estimé en millisecondes (ms). Le Tableau 14. présente les différents temps de cycles en fonction de plusieurs topologies du réseau *RRFR*.

Nent	Nrbf	Tc *(ms)*
2	10	1,586
2	20	2,69
2	30	3,794
2	50	6,002
2	80	9,314
2	100	11,522
2	150	17,042
2	200	22,562
5	20	4,325
5	30	5,885
5	50	9,005
5	80	13,685
5	100	16,805
5	200	32,405
5	350	55,805
5	***500***	***79,205***

Nent	Nrbf	Tc *(ms)*
8	50	12,008
8	80	18,056
8	100	22,088
8	150	32,168
8	200	42,248
8	250	52,328
8	300	62,408
8	400	82,568
10	100	25,61
10	150	37,21
10	250	60,41
10	400	95,21
10	500	118,41
10	700	164,81
10	800	188,01
10	1000	234,41

Tableau 14. *Différents temps de cycle en fonction de plusieurs dimensions du réseau RRFR. Un réseau de 5 neurones bouclés et 500 neurones gaussiens possède un temps de cycle d'environ 80 ms. Cela dépend de la nature du problème, mais l'on peut déjà supposer qu'avec une telle dimension on peut obtenir de bons résultats (classification ou approximation de fonction).*

Cette relation entre la dimension du réseau *RRFR* et le temps de cycle est aussi représentée par le graphe ci-après :

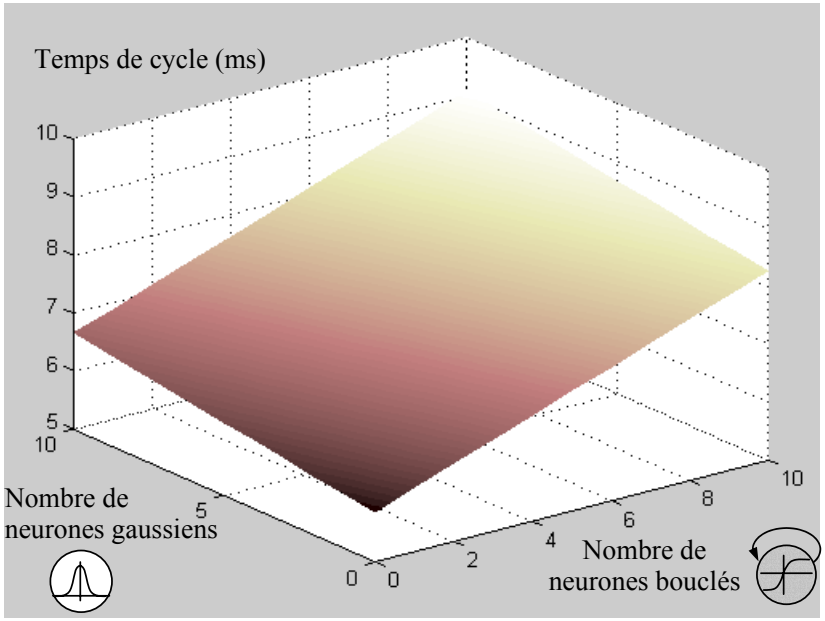


Figure 123. *Temps de cycle du programme neuronal step7 en fonction du nombre de neurones bouclés et de neurones gaussiens.*

D'après ces résultats, un réseau *RRFR* avec 10 neurones bouclés et 1000 neurones gaussiens possède un temps de cycle de 234 ms. Ceci représente une dimension d'un réseau *RRFR* assez importante rarement obtenue en pratique (que ce soit en classification ou en approximation de fonctions). On peut penser qu'une dimension moyenne d'un réseau *RRFR*, avec laquelle on peut obtenir de bons résultats[44], est constituée d'environ 5 neurones bouclés et 500 neurones gaussiens. Une telle dimension possède un temps de cycle de 79 ms. Sauf pour des processus extra rapides, un temps de cycle de cet ordre (environ 80 ms) représente une perspective intéressante pour des applications de surveillance temps réel de processus industriel. Néanmoins, en fonction de l'application, on peut avoir deux cas de figure : une situation où le temps de cycle du programme neuronal est inférieur au temps d'acquisition. Dans ce cas, une synchronisation entre les deux cycles est nécessaire. Par contre, si le temps d'acquisition est plus rapide que le temps de cycle du programme neuronal, on perd dans ce cas une partie des données de surveillance. Nous allons étudier plus en détail ces deux situations.

VI.4.2.1. Temps de cycle du réseau de neurones inférieur au cycle d'acquisition

En fonction de la valeur du temps de cycle du système à surveiller et de la dimension du réseau de neurones chargé dans l'automate, on pourrait avoir une situation où la dynamique du réseau *RRFR* est plus rapide que celle du système à surveiller. En d'autres termes, le temps de cycle du programme neuronal de surveillance est plus rapide que le temps de réponse du système d'acquisition. Cette situation provoquerait un dysfonctionnement du programme neuronal. Prenons par exemple un réseau *RRFR* avec 2 neurones bouclés et 10 neurones gaussiens. Nous avons vu au chapitre précédent qu'avec une telle dimension de réseau *RRFR* on avait obtenu de bons résultats concernant la prédiction de la concentration en CO_2 du four à gaz. Le temps de cycle d'une telle architecture de réseau *RRFR* dans l'automate est de 1,5 ms. Si le temps de réponse du cycle d'acquisition est supérieur au temps de cycle du réseau de neurones, ce dernier risque de considérer la même valeur restée stable entre deux acquisitions comme une nouvelle valeur d'acquisition. La prédiction du signal de sortie serait dans ce cas complètement erronée.

Par exemple, le temps de réponse du variateur de notre maquette de test, pour le cycle d'une acquisition d'une variable du moteur, est d'environ 20 ms. Le temps de cycle d'un réseau *RRFR* avec la dimension précédente (2 neurones bouclés et 10 neurones gaussiens) est de 1,5 ms. Avec de telles données, le programme automate exécute environ 10 cycles avant qu'une nouvelle acquisition ne soit disponible. L'aspect dynamique du réseau de neurones est ainsi fortement déstabilisé. On comprend alors qu'il serait très important de synchroniser le programme neuronal avec la période d'acquisition des variables si l'on veut garder l'aspect dynamique du réseau *RRFR*. Cette synchronisation entre le début de chaque cycle du

[44] *Par exemple la carte IBM/ZISC (Zero Instruction Set Computer), qui représente une implémentation Hard d'un réseau RFR possède 576 neurones gaussiens.* (Zemouri *et al.,* 2001)

programme automate et la période d'acquisition de la variable de surveillance est réalisée par la détection d'un front montant généré à chaque nouvelle acquisition. La mise à jour du réseau de neurones chargé dans l'automate est ainsi effectuée à chaque nouvelle acquisition.

Pour mieux expliquer les conséquences d'une non synchronisation entre le programme neuronal et la période d'échantillonnage, nous présentons sur les figures ci-après un exemple illustratif de période d'acquisition et de cycle de traitement du programme automate.

On remarque bien que dans le cas où le temps de cycle du programme neuronal de l'automate est plus court que la période d'acquisition de la variable de surveillance, le réseau *RRFR* perd un peu de son aspect dynamique. En effet, le réseau *RRFR* traite la même valeur d'acquisition pendant plusieurs cycles (Figure 124). La conséquence de cette non synchronisation est que pour le réseau de neurones, la variable s'est stabilisée pendant un certain temps. Le réseau *RRFR* serait alors incapable de reconnaître un pic de fausse alarme ou même un palier de dégradation. Une synchronisation entre le programme neuronal et le cycle d'acquisition des variables de surveillance est donc indispensable, comme le montre la Figure 125. Une procédure a été rajoutée au programme principal pour que le calcul de la sortie du réseau de neurones soit effectué à chaque nouvelle acquisition.

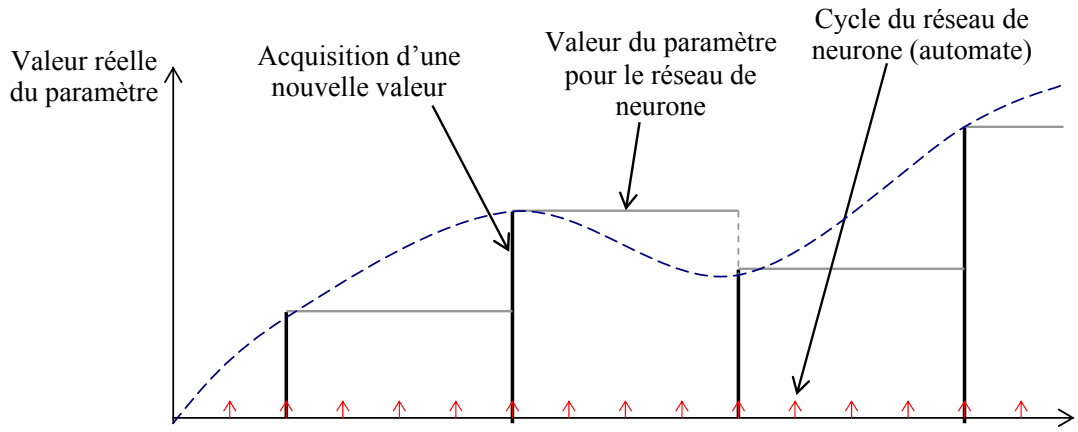


Figure 124. *Conséquences d'une non synchronisation entre le temps de cycle de l'automate et le temps de réponse de l'acquisition dans le cas où le programme automate est plus rapide que le cycle d'acquisition. Le réseau RRFR perd de sa dynamique dans le cas où son temps de cycle est inférieur au temps de réponse de l'acquisition.*

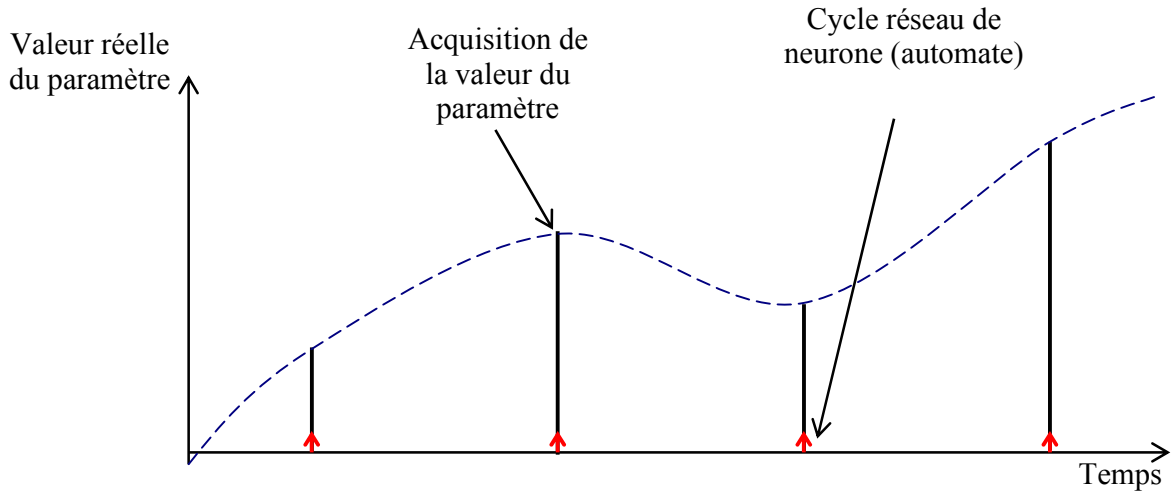


Figure 125. *Synchronisation entre temps de cycle de l'automate et temps d'acquisition du paramètre à surveiller.*

Nous présentons sur la Figure 126 le comportement de la mémoire dynamique du réseau *RRFR* dans le cas où le cycle du programme automate est plus rapide que le cycle d'acquisition. Par exemple, pour une trame d'acquisition d'une variable binaire donnée égale à $[1,0,1]$, le réseau *RRFR* est exécuté 10 fois par l'automate pour la même valeur de cette variable (égale à 0). Pour le réseau de neurones, cette donnée est restée stable sur la valeur de 0, ce qui correspond à une trame d'acquisition complètement différente de la vraie trame. En synchronisant le programme neuronal avec la période d'acquisition, le réseau de neurones traite les données à chaque acquisition. La trame traitée est alors la même que celle réellement acquise (voir Figure 127).

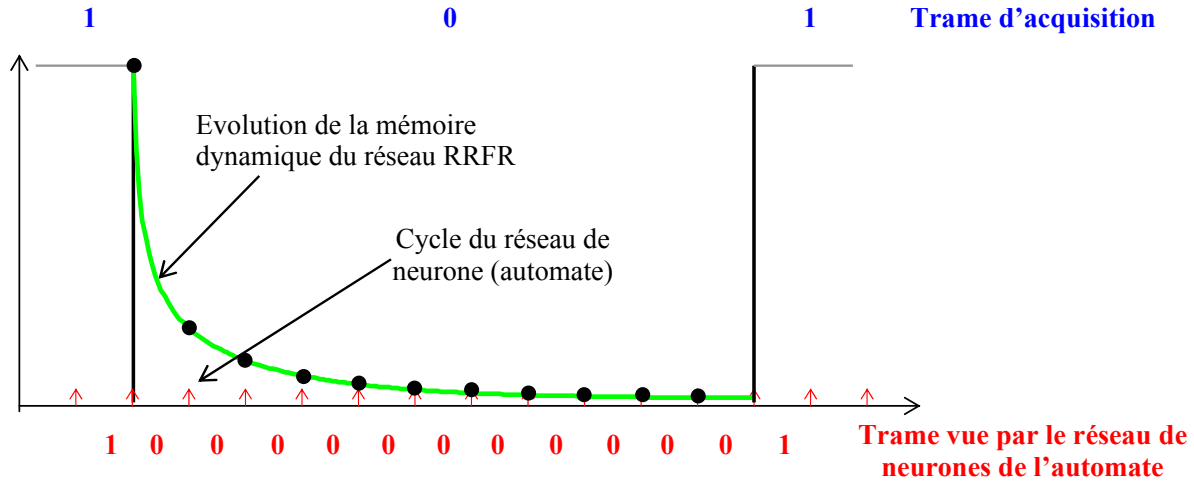


Figure 126. *Comparaison entre le cycle d'acquisition et l'évolution de la mémoire dynamique du réseau RRFR dans le cas où les deux cycles ne sont pas synchronisés. Le cycle du réseau de neurones dans l'automate est très petit par rapport au cycle d'acquisition. Cette situation provoque une perte de mémoire du réseau RRFR (la sortie de la mémoire dynamique du réseau RRFR tend vers zéro avant la nouvelle acquisition).*

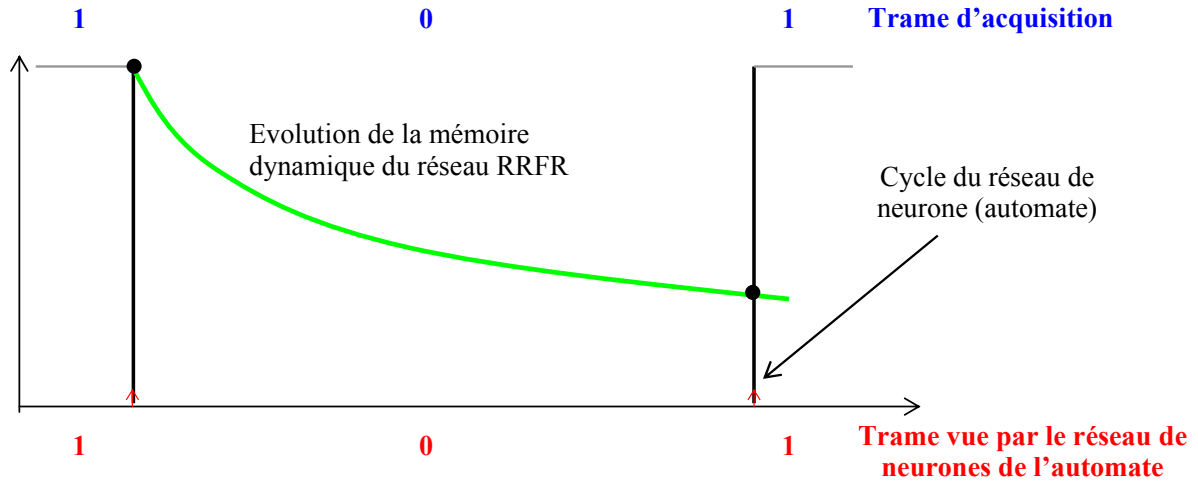


Figure 127. *Synchronisation entre le temps de cycle du réseau de neurones dans l'automate avec le cycle d'acquisition. La mise à jour de la mémoire dynamique se fait à chaque cycle d'acquisition.*

VI.4.2.2. Temps de cycle du réseau de neurones supérieur au cycle d'acquisition

La deuxième situation est celle où le temps de cycle du réseau de neurones est plus grand que la période d'acquisition. Dans ce cas, le problème de synchronisation entre les deux périodes ne se pose plus. Par contre, cette situation provoque un autre problème qui est celui de la perte d'informations. En effet, on peut voir sur la Figure 128, qu'entre deux cycles automate, trois valeurs du paramètre à surveiller ne sont pas prises en compte par le réseau de neurones. Pour ce dernier, la valeur du paramètre évolue uniquement entre les points a,b,c,d. Le point (b) est plutôt vu par le réseau comme un brusque changement plutôt que faisant parti d'une évolution (dégradation par exemple). Le réseau *RRFR* est ainsi incapable de caractériser la dynamique du signal qui est plus rapide que celle du programme neuronal.

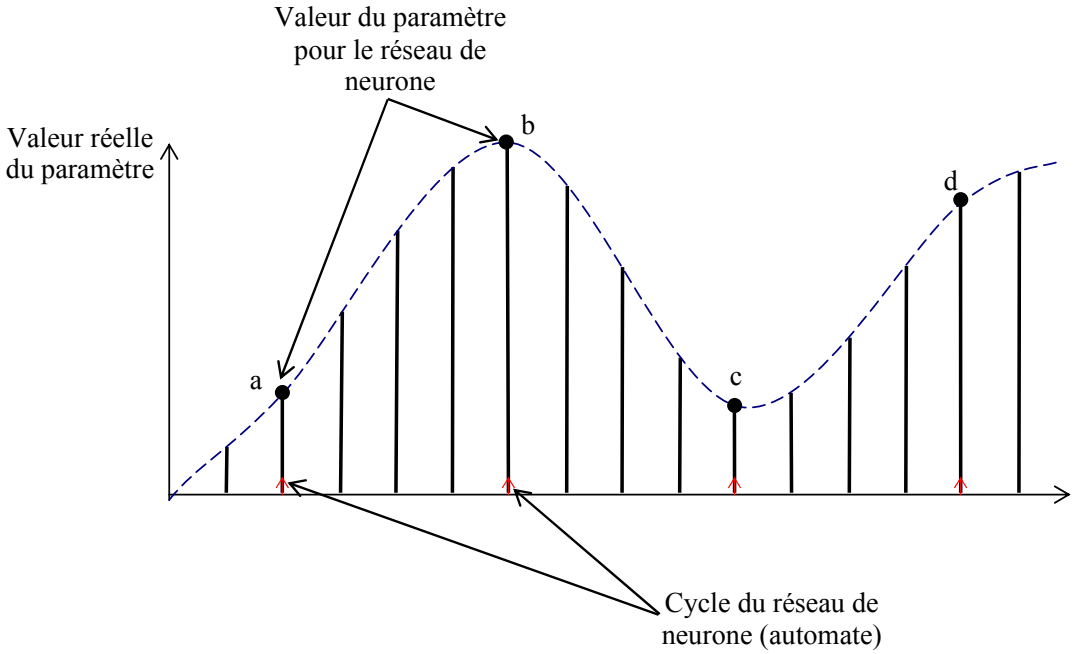


Figure 128. *Cas où le temps de cycle du programme automate est plus lent que le cycle d'acquisition. Cette situation provoque une perte d'information. Un certain nombre de valeurs du paramètre entre deux cycles automate ne sont pas prises en compte par le réseau de neurones.*

Pour éviter cette situation où le réseau *RRFR* est plus lent que le cycle d'acquisition, il faudrait diminuer le nombre de neurones gaussiens pour diminuer le temps de cycle (voir Tableau 14). En d'autres termes, connaissant le nombre de capteurs (nombre de neurones d'entrée) que l'on veut surveiller (N_{ent}) ainsi que le temps d'acquisition ($T_{acquisition}$), on peut estimer grâce à l'équation [201], le nombre maxi de neurones gaussiens ($N_{RBF_{\max}}$) pouvant être mémorisé par le réseau RRFR afin d'éviter cette situation, par l'équation ci-dessous :

$$N_{RBF_{\max}} = \frac{T_{acquisition} - 0.241 * N_{ent}}{0.08 + 0.0152 * N_{ent}} \qquad [202]$$

avec la condition suivante sur le nombre de neurones d'entrée :

$$N_{ent} < \frac{T_{acquisition}}{0.241} \qquad [203]$$

qui représente le nombre maximum de neurones d'entrée pouvant être utilisés pour ne pas avoir un temps de cycle du programme automate supérieur à celui du cycle d'acquisition.

VI.5. Conclusion

Dans ce chapitre nous avons présenté une solution d'exploitation du réseau *RRFR* pour des applications de surveillance industrielle en temps réel. L'idée de base est de mettre au point un traitement en temps réel des variables de surveillance directement à proximité de l'équipement industriel, c'est-à-dire d'exploiter une solution de surveillance par Automate Programmable Industriel (API). Le principe de cette solution est donc d'avoir un programme neuronal en langage automate chargé dans l'unité centrale de l'automate pour traiter les données capteurs. La deuxième partie de cette solution, qui est tout aussi importante que la partie traitement, concerne la phase d'apprentissage et de mise à jour du réseau *RRFR*. Cette deuxième phase est entièrement gérée à distance par un expert responsable de la mise à jour du réseau *RRFR* embarqué dans l'automate. Cette communication avec un expert distant est obtenue grâce au coupleur Web de l'automate. Ce coupleur joue le rôle d'une jonction entre la couche de communication TCP/IP et l'unité centrale de l'automate (CPU). Nous avons donc structuré le programme automate pour qu'il puisse être entièrement évolutif à distance, c'est-à-dire permettre à un expert distant d'effectuer un apprentissage par connexion TCP/IP.

Le réseau *RRFR* s'est montré performant pour ce type d'exploitation. Ceci est dû en grande partie à la simplicité de son architecture. Nous pouvons affirmer que les deux choix que nous avons effectués, en l'occurrence le choix du réseau *RFR* (pour la partie statique du traitement) ainsi que celui des architectures localement récurrentes (LRGF pour la partie dynamique), ont été très avantageux. Une autre architecture récurrente, comme un *PMC* globalement récurrent aurait été plus compliquée à embarquer dans un automate et aurait certainement pris beaucoup plus de mémoire dans la *CPU*. La gestion des connexions d'un tel réseau récurrent est beaucoup plus compliquée qu'une architecture *LRGF*.

Nous avons également effectué des tests de performance par rapport à plusieurs dimensions du réseau *RRFR* embarqué. A partir des quantifications des différents temps de cycles du programme neuronal, nous considérons que cette solution ouvre des perspectives très intéressantes en traitement temps réel des variables de surveillance à la seule condition de bien veiller à ce que le nombre de neurones gaussiens soit bien choisi pour ne pas avoir un cycle d'acquisition plus rapide que le cycle du programme neuronal dans l'automate. Ce nombre maximum a été quantifié en fonction du nombre des variables à surveiller et en fonction du cycle d'acquisition.

Le deuxième avantage de cette solution est la robustesse industrielle des architectures API. En effet, contrairement aux PC qui ne sont pas à l'abri d'un bug impromptu (même les PC industriels qui sont censé être plus robustes que les PC personnels), un automate est généralement conçu pour fonctionner dans un environnement industriel hostile avec plus de fiabilité et de robustesse.

L'aspect accès aux données du programme de surveillance et apprentissage à distance du réseau *RRFR* permet d'ouvrir une perspective très intéressante ***d'externalisation de la***

maintenance par la e-maintenance. Cette tendance d'actualité commence à prendre une certaine ampleur au sein des entreprises soucieuses de réduire les coûts de la maintenance. En effet, une solution d'externalisation de la maintenance possède certains avantages qui sont essentiellement :

- mieux connaître le budget attribué à la maintenance et donc avoir la possibilité d'optimiser les coûts,
- se recentrer sur son véritable métier de production et confier la fonction maintenance à des professionnels.

La solution présentée dans ce chapitre convient très bien à une telle application. Le programme de surveillance neuronal embarqué dans l'automate est entièrement configurable depuis n'importe quel point d'accès au réseau Internet. Un expert distant peut alors mener à bien l'apprentissage, d'autant plus que la base de données nécessaire pour cet apprentissage (historique de la machine) est également accessible à distance. Dans une grande mesure, nous avons ainsi accru l'autonomie et l'intelligence de l'équipement de surveillance, en occurrence, l'automate programmable.

En ce qui concerne la phase de commercialisation du système neuronal de surveillance avec notre partenaire *AVENSY Ingénierie*, il faudra continuer le prototypage par un travail de développement et de test de détection, de diagnostic de défaillances ainsi que des tests de prédiction en environnement industriel. Ces tests sont absolument nécessaires pour convaincre une clientèle industrielle (constructeur de machines outils, professionnels de la surveillance et de la maintenance, intégrateurs, etc.) devenue de plus en plus exigeante. Des perturbations peuvent alors être rajoutées à la maquette pour des simulations de défaillances, car provoquer de vraies défaillances avec comme risque la destruction du matériel serait relativement dangereux et potentiellement coûteux. Ces perturbations peuvent être obtenues en agissant sur les valeurs de surveillance après l'acquisition, c'est-à-dire en générant des perturbations au niveau du programme automate (soft) et non pas sur le matériel (hard). Plusieurs combinaisons de défaillances simulées peuvent alors être obtenues en agissant sur l'ensemble des données acquises.

Conclusion Générale & Perspectives

Conclusion Générale & Perspectives

Les travaux de recherche présentés dans cette thèse portent sur l'étude des réseaux de neurones artificiels pour la surveillance dynamique des systèmes de production industriels. Dans le cadre de la surveillance, notre étude concerne la détection de défaillances et le pronostic industriel. La surveillance classique a surtout tendance à traiter les variables d'une façon statique. Dans ce cas, la dynamique du système à surveiller n'est pas prise en compte, contrairement à la surveillance dynamique qui, elle, est capable par exemple de reconnaître une fausse alarme. Le pronostic quant à lui, correspond à la détection d'une dégradation avant que le système n'atteigne le seuil de défaillance. Dans ce sens, nous avons proposé une nouvelle architecture de Réseau Récurrent à Fonctions de base Radiales (*RRFR*). Le *RRFR* profite des performances ainsi que de la simplicité d'apprentissage des réseaux *RFR*, avec l'efficacité dynamique et la facilité de paramétrage des architectures localement récurrentes. Nous avons ainsi « *greffé* » une mémoire dynamique au réseau *RFR*. Pour le calcul des paramètres des fonctions gaussiennes (centre et écart type), nous avons proposé une version améliorée de la technique d'apprentissage des *k-moyennes* qui est moins sensible à la phase de paramétrage de l'algorithme. Une interprétation du réseau *RRFR* en langage automate a été présentée afin de proposer une solution d'exploitation de surveillance en temps réel entièrement paramétrable à distance via une connexion *TCP/IP*.

Les principales contributions de cette thèse

Les principales contributions de cette thèse sont regroupées en trois parties. Une première partie regroupe l'état de l'art autour de trois points d'intérêts en étroite corrélation :

- Différentes méthodologies de surveillance des systèmes de production (Chapitre I).
- Application des réseaux de neurones à la surveillance industrielle (Chapitre II).
- Différentes architectures de réseaux de neurones temporels (Chapitre III).

La deuxième partie de notre travail, articulée autour des deux chapitres suivants, synthétise l'essentiel de notre contribution scientifique. Nous avons ainsi proposé (Chapitre IV) une

nouvelle architecture d'un réseau de neurones dynamique pour la surveillance industrielle (le *RRFR*). Un ensemble de tests comparatifs ont été appliqués sur le nouvel outil (Chapitre V), en utilisant quelques benchmarks significatifs reliés aux domaines de la surveillance dynamique. Afin de renforcer la phase délicate de paramétrage du réseau de neurones (phase d'apprentissage), l'étape de test nous a permis de mettre en évidence la nécessité de développer une version améliorée de l'algorithme des *k-moyennes* qui possède des performances meilleures que celles de la version simple :

- Proposition d'un réseau de neurones dynamique (*RRFR*) avec une étude et une simulation des performances de sa mémoire dynamique (Chapitre IV).
- Evaluation des performances du réseau (*RRFR*) proposé sur des problématiques de surveillance dynamique ainsi que la présentation de l'algorithme d'apprentissage proposé (Chapitre V).

La troisième partie concerne l'exploitation industrielle de l'idée de la thèse. Cette étude a été menée en collaboration avec la société AVENSY Ingénierie, co-auteur du brevet d'invention déposé et directement intéressée par l'exploitation commerciale de cette solution. Il s'agit de la démarche d'adaptation du *RRFR* dans le cadre industriel de la *e-maintenance*, de l'étude du système de surveillance dans sa configuration la plus appropriée, ainsi que de la phase de prototypage du système :

- Développement d'une solution de surveillance intelligente distante accessible à distance par un serveur Web (Chapitre VI).

1. Première partie – état de l'art

La première démarche entreprise dans cette étude correspond à l'élaboration d'un lexique des définitions des mots clés liés à la surveillance industrielle. En effet, pendant la phase de recherche bibliographique, nous avons constaté certaines divergences entre les définitions du domaine. Nous avons donc essayé de regrouper les définitions les plus représentatives qui correspondent avec notre point de vue.

Nous avons donné un état de l'art aussi large que possible sur les travaux de surveillance par réseaux de neurones artificiels. Nous les avons classés en deux catégories : la première est celle où les réseaux de neurones artificiels sont utilisés comme outil d'approximation de fonctions (pour le pronostic). Dans la deuxième catégorie, les réseaux de neurones sont utilisés pour résoudre le problème de la surveillance par reconnaissance des formes (détection des modes). Les deux architectures neuronales les plus utilisées en surveillance sont le Perceptron Multi Couches (*PMC*) avec sa représentation globale de son espace de données et les Réseaux à Fonctions de base Radiales (*RFR*) avec une représentation plutôt locale de cet espace. Cette différence structurelle a été pour nous un facteur décisif concernant le choix de l'architecture à adopter pour le développement d'une solution de surveillance en temps réel.

Nous avons présenté l'ensemble des techniques d'apprentissage des deux réseaux de neurones (*PMC* et *RFR*). La technique d'apprentissage du *PMC* (l'algorithme de rétropropagation du gradient de l'erreur) est beaucoup plus lourde que les techniques existantes pour l'apprentissage des réseaux *RFR* (*RCE*, *DDA*, *k-moyennes*).

La prise en compte du facteur temps pour la surveillance dynamique des systèmes de production nous a orienté vers l'étude des différentes architectures de réseaux de neurones temporels. Le temps est pris en considération selon deux grandes familles : les réseaux de neurones à représentation externe du temps et ceux à représentation interne. Pour la représentation externe, un mécanisme externe au réseau de neurones est chargé de retarder les données d'entrée pendant un certain temps. L'information temporelle est alors transformée en une information spatiale. Les architectures de réseaux de neurones statiques peuvent alors être utilisées. Par contre, dans la représentation interne du temps, le réseau de neurones est capable de traiter le temps sans aucun mécanisme externe. Ces réseaux sont appelés réseaux de neurones dynamiques. Parmi ces réseaux dynamiques, seuls les réseaux récurrents possèdent une mémoire dynamique interne grâce à la récurrence des connexions. Nous avons alors focalisé notre étude sur ces réseaux, en donnant quelques architectures connues de réseaux récurrents, avec la façon dont est mené l'apprentissage. Malgré leur bonne performance, ces algorithmes d'apprentissage souffrent d'une extrême lourdeur tant en terme de ressources informatiques qu'en terme de temps de convergence.

2. Deuxième Partie – contributions scientifiques

A la fin de l'étude des réseaux de neurones temporels, l'architecture qui nous a le plus séduit est celle des réseaux de neurones récurrents, la seule à posséder une mémoire dynamique interne. Les réseaux récurrents sont capables de garder une trace interne d'un événement passé. Par contre, leur apprentissage est extrêmement lourd et coûteux en temps. Nous nous sommes alors posé la question suivante : ***pourquoi compliquer l'architecture d'un réseau de neurones avec des récurrences globales quand on peut simplifier le réseau en utilisant des récurrences locales ?*** Nous avons donc établi une étude des architectures Localement Récurrentes Globalement Feedforward (*LRGF*). Nous avons, d'une part, restructuré cette étude pour qu'elle soit en adéquation avec l'étude de l'ensemble des architectures de réseaux temporels et, d'autre part, nous avons approfondi cette étude avec des développements mathématiques et des tests de performances par simulation informatique.

La simplicité de ces réseaux réside dans le fait que la récurrence n'est autorisée qu'au sein du neurone même. Deux types d'architectures localement récurrentes existent : architecture *LRGF* avec retour de l'activation et *LRGF* avec retour de la sortie. A travers toute une étude théorique approfondie avec des tests de simulation, nous avons constaté que les deux types d'architectures *LRGF* possèdent des performances quasiment identiques. Nous avons toutefois opté pour le neurone à retour local de la sortie (le neurone bouclé de Frasconi-Gori-Soda). Les motivations de ce choix sont essentiellement dues au point du retour de la

connexion : pour le neurone à retour local de l'activation, ce point de retour se situe avant la non-linéarité du neurone alors que pour le neurone à retour local de la sortie, ce retour se trouve après la non-linéarité. Cette différence structurelle peut engendrer des conséquences dans la prise en compte de la non-linéarité d'un système. Une première perspective qui se dégage serait d'approfondir l'étude (mathématique) de l'impact du point de retour du neurone bouclé sur la prise en compte de la non-linéarité du signal d'entrée.

Le fruit de l'ensemble de l'étude est la proposition d'une architecture localement récurrente basée sur les Réseaux à Fonctions de base Radiales (*RFR*). Le terme anglophone de ces réseaux est Radial Basis Function Network (*RBF*). Le *R*éseau de neurones *R*écurrent à *F*onctions de base *R*adiales (*RRFR*) que nous avons proposé profite des avantages d'une structure dynamique et de la simplicité de paramétrage des architectures *LRGF* ainsi que de la facilité et de la flexibilité d'apprentissage des réseaux *RFR*. Le réseau *RRFR* se démarque donc des autres architectures de réseaux récurrents essentiellement par sa simplicité de paramétrage. Nous l'avons testé sur trois problématiques distinctes :

- *reconnaissance de séquences temporelles (détection)*,
- *prédiction temporelle (pronostic)*
- *reproduction de séquences temporelles (détection)*.

A travers un exemple simple d'un système à événements discrets (*SED*), nous avons montré la simplicité avec laquelle le réseau *RRFR* est capable d'apprendre plusieurs séquences booléennes simples. Sa capacité de généralisation locale lui permet de reconnaître des séquences proches de celles apprises et de détecter des séquences inconnues. Ce type d'application est très utile pour la surveillance d'un système à événements discrets. Le réseau apprend à reconnaître des séquences de bon fonctionnement et à détecter des séquences de dysfonctionnement connues. Deux séquences différentes possèdent deux prototypes différents. Donc chaque séquence de dysfonctionnement possède sa propre cause. L'expert pourra ainsi diagnostiquer la cause de chaque séquence de dysfonctionnement et faire apprendre le prototype correspondant au réseau *RRFR*. La limite du réseau *RRFR* dans ce type d'application réside dans son incapacité d'apprendre des séquences complexes, c'est-à-dire des séquences où un événement se produit plus d'une fois.

En surveillance de paramètres de type réel, l'apprentissage de séquences temporelles peut servir à la détection précoce d'un palier de dégradation et à éliminer les pics de fausse alarme. Nous avons également testé le réseau *RRFR* sur la reconnaissance des types de collision d'un bras de robot. Dans cet exemple, chaque type de collision est caractérisé par une évolution temporelle de trois mesures de force.

Le deuxième test du réseau *RRFR* concerne la prédiction temporelle. Nous avons choisi deux exemples de prédictions temporelles : la série temporelle chaotique Mackey-Glass et l'exemple de prédiction de la sortie en concentration de CO_2 d'un four à gaz. Nous avons amélioré les performances de prédiction du réseau *RRFR* grâce à une version évoluée de l'algorithme d'apprentissage des *k-moyennes*. En effet, la version classique de cet algorithme

possède certaines faiblesses liées à la phase d'initialisation : l'expert doit choisir arbitrairement le nombre *k* des centres qui seront par la suite initialisés aléatoirement parmi l'ensemble des points d'apprentissage. Nous avons exploité la technique *Fuzzy Min-Max* qui permet de déterminer les points les plus représentatifs d'une population de points d'apprentissage. Cette façon d'initialiser l'algorithme des *k-moyennes* permet de garantir une stabilité du résultat avec une erreur de prédiction proche du minimum global. Cette initialisation permet également à l'algorithme de converger vers un nombre des centres *k* garantissant une bonne généralisation en évitant les deux zones de sous-apprentissage et de sur-apprentissage. L'apprentissage du *RRFR* est ainsi moins dépendant de la phase délicate de paramétrage de l'algorithme d'apprentissage.

Le troisième test concerne l'apprentissage du réseau *RRFR* à reproduire des séquences temporelles. Grâce à sa mémoire dynamique, l'apprentissage d'une séquence temporelle n'est autre qu'un problème d'interpolation linéaire. Une fois que les paramètres des gaussiennes sont déterminés, les poids de sortie du réseau *RRFR* sont calculés de manière à résoudre le problème d'interpolation dont le vecteur d'entrée représente l'évolution de la mémoire dynamique et la sortie représente la séquence temporelle à reproduire. La seule condition est que la séquence temporelle ne soit pas plus longue que la longueur de la mémoire dynamique du réseau *RRFR*, c'est-à-dire environ 300 unités de temps. L'apprentissage d'une séquence temporelle en utilisant les algorithmes dits *trajectory learning* (*BPTT* ou *RTRL*) pour les réseaux globalement récurrents est extrêmement lourd en temps de convergence et en ressources informatique alors qu'avec le *RRFR* le calcul des poids de sortie s'effectue par simple inversion matricielle.

3. Troisième partie – exploitation industrielle

Un des objectifs de notre travail étant de développer un outil neuronal dynamique, facilement paramétrable via la couche de communication TCP/IP pour des applications de surveillance industrielle temps réel, nous avons alors décomposé le réseau *RRFR* en fonctions élémentaires. Le réseau *RRFR* est chargé dans un Automate Programmable Industriel (*API*) et assure une surveillance temps réel. La partie apprentissage et visualisation des variables de surveillance est entièrement gérée à distance grâce au coupleur serveur Web de l'automate. Un expert distant (à l'intérieur ou à l'extérieur de l'entreprise) peut ainsi suivre l'évolution de l'équipement à surveiller et également paramétrer le réseau *RRFR* à distance. Cette solution profite du traitement temps réel avec la fiabilité du fonctionnement des architectures à automates industriels. Cette application ouvre des perspectives qui peuvent être très intéressantes pour l'externalisation de la maintenance. En effet, beaucoup d'entreprises optent pour ce genre de solution qui leur permet à la fois de mieux maîtriser leur budget maintenance mais surtout de se recentrer sur leur véritable métier de production.

Perspectives

1. Perspectives scientifiques

La remarque fondamentale que nous pouvons faire est que, malgré les résultats grandement surprenants et prometteurs obtenus par les réseaux de neurones artificiels, ces derniers restent tout de même assez loin d'égaler les capacités sensorielles et, surtout, de raisonnement d'un expert humain. Nous avons vu que les réseaux de neurones en général et le *RRFR* en particulier sont très efficaces dans la détection d'une défaillance, détection d'une dégradation (palier de dégradation), modélisation et prédiction d'une évolution temporelle d'un signal non linéaire ; par contre, la fonction de diagnostic est à notre avis une tâche très complexe et ne peut être qu'en partie résolue par la technique de reconnaissance des formes. La raison principale est que l'expert humain dans sa mission de tenter de diagnostiquer la cause d'une défaillance de toute une machine ou d'un sous ensemble de cette machine, fait souvent appel à d'autres informations que les valeurs quantitatives (les données capteurs). Il utilise par exemple : son ouïe pour reconnaître les bruits anormaux d'une machine, son odorat pour détecter une odeur de brûlé et son origine, sa vision pour contrôler la qualité des pièces produites par la machine et identifier les différents défauts des pièces, son toucher pour vérifier la tension d'une courroie et également pour voir s'il n'y a pas de fuite d'huile, sa mémoire pour se rappeler de ses connaissances préalablement acquises avec d'autres machines.

La question qu'on peut se poser et qui peut ouvrir deux perspectives complètement antagonistes est de *savoir si l'on veut à tout prix remplacer l'expert humain afin d'automatiser à 100% cette tâche de diagnostic* ?

Dans l'affirmative, les recherches s'orienteront dans ce cas plutôt vers les neurosciences et le développement propre des réseaux de neurones artificiels afin de développer des architectures neuronales qui tendent à se rapprocher davantage des réseaux de neurones biologiques.

Par contre, la deuxième option qui nous semble la plus intéressante est de savoir comment faire pour extraire le bon vecteur caractéristique associé à une cause bien précise. Comment fait l'expert humain pour rassembler toutes les informations lui permettant de prendre sa décision ? Nous pensons qu'une couche ***Neuro-Flou-Temporelle*** (association des techniques de la logique floue avec les techniques neuronales temporelles) en amont du réseau de neurones pourrait offrir une piste intéressante permettant d'extraire une certaine connaissance floue de l'expert humain. Une fois ce vecteur identifié, les réseaux de neurones artificiels peuvent très bien être exploités pour apprendre cette forme. Par contre, la limite que l'on peut rencontrer en utilisant un réseau de neurones artificiel est que généralement, pour une application donnée, la dimension du vecteur d'entrée d'un réseau de neurones est a priori fixée. C'est donc là où peut résider toute la difficulté d'utiliser brutalement un réseau de neurones pour la tâche de diagnostic car, en pratique, les informations utilisées pour le

diagnostic sont souvent différentes (la dimension et la nature du vecteur d'entrée ne sont pas les mêmes pour différents diagnostics). On peut imaginer alors une solution distribuée, c'est-à-dire un réseau de neurones dynamique avec sa couche *Neuro-Flou-Temporelle* pour chaque type de cause. Chaque réseau de neurones possède bien évidemment son propre vecteur d'entrée qui caractérise le mieux l'identification de la cause de la défaillance (diagnostic). La décision globale du diagnostic final sera prise par un ***superviseur principal***. Ce superviseur peut être établi soit à partir d'une base neuronale (un type de réseau de neurones – PMC ou RFR –), soit à partir d'une architecture des systèmes ***multi-agents***. Les systèmes multi-agents représentent des outils de l'intelligence artificielle avec certaines capacités très intéressantes concernant la prise de décision en fonction de certains critères imposés par l'expert. Cette architecture peut offrir une solution intéressante pour la prise de décision globale concernant le diagnostic en fonction des réponses locales de chaque architecture neuronale associée aux différentes causes.

2. Perspectives d'exploitation industrielle

Plusieurs perspectives restent à entreprendre concernant l'exploitation industrielle. La première est de tester les capacités dynamiques du réseau *RRFR* chargé dans l'automate. En effet, les capacités du réseau *RRFR* à détecter un palier de dégradation, un pic de fausse alarme ainsi que ses capacités de prédiction n'ont pas été testées sur le prototype de l'automate programmable bien qu'elles soient validées par la simulation. Le problème récurrent à tout test d'un système de détection et de diagnostic de défaillances réside justement dans l'existence de ces défaillances. L'idée de provoquer une défaillance réelle ne serait évidemment pas envisageable de peur de détruire l'équipement. Le plus judicieux serait donc de concevoir un simulateur de pannes. On peut, par exemple, déclencher un système électronique qui perturbe une variable acquise par l'automate. La sortie de ce système serait exploitée par les cartes d'entrée (analogiques ou binaires) de l'automate. On peut ainsi perturber les variables de surveillance, en générant une situation similaire à une défaillance réelle. Par ailleurs, toute une combinaison de défaillances et de dégradations peut être générée pour tester le réseau *RRFR*.

La seconde démarche est bien évidemment d'adapter l'interface homme machine en concordance avec le type d'équipement industriel à surveiller. En effet, l'interface actuelle présente un caractère générique qui nous a servi pour effectuer la validation des fonctions de communication de base et de la structure neuronale. Cette adaptation fait partie de la suite du projet de collaboration entre le *LAB* et la société *AVENSY*.

Bibliographie

Adamson M. J. et Damper R. I., «*A recurrent network that learns to pronounce English text* », In *Proceedings International Conference on Spoken Language Processing (ICSLP'96)* 4, pp. 1704-1707, 1996.

Agteberg F.P., Geomathematics, Elsevier, Amsterdam, 1974.

Almeida L.B., « *Backpropagation in Perceptrons with feedback* », in NATO ASI Series, Vol. F41, Neural Computers, Edited by R. Eckmiller and Ch. V. d. Malsburg, 1988.

Amit D.J., « *Neural Network Counting Chimes* », Proceedings of the National Academy of Sciences USA, 85, pp. 2141-2145. 1988.

Arahal M.R., Cepeda A., Camacho F.E., « Input variable Selection for Forcasting Models », 15éme IFAC World Congress on Automatic Control, Barcelone, Espagne juillet 2002.

Atiya A.F., El-Shoura S.M., Shaheen S.I. et El-Sherif M.S., « A Comparison Between Neural Network Forecasting Techniques – Case Study : River Flow Forecasting », IEEE Transactions on Neural Networks, Vol. 10, N°2, pp. 402-409, Mars 1999.

Atiya A.F. et A.G. Parlos, « *New Results on Recurrent Network Training : Unifying the Algorithms and Accelerating Convergence* », IEEE Transactions on Neural Networks, Vol. 11, N°3, pp. 697-709, May 2000.

Aussem A., « *Théorie et Application des Réseaux de Neurones Récurrents et Dynamiques à la Prédiction, à la Modélisation et au Contrôle Adaptatif des Processus Dynamiques* », Thèse de Doctorat, Université René Descartes – Paris V, juin 1995.

Aussem A., « *Le Calcul du Gradient d'Erreur dans les Réseaux de Neurones Discrets Bouclés à Délais : Application aux Télécom et aux Sciences Environnementales* », Habilitation à Diriger des Recherches, Université Blaise Pascal Clermont-Ferrand II/ France 19 Décembre 2002.

Back A.D. et A.C. Tsoi, « A Time Series Moddeling Methodology Using FIR and IIR Synapses », Proceeding Workshop on Neural Networks for Statistical and Economic Data, Dublin, DOSES, Statistical Office of European Comminities, F. Murtagh Ed., pp. 187-194, 1990.

Basseville M., « *Detecting Changes in Signals and Systems – A Survey* », Automatica, Vol. 24, N°3, p. 309-326, 1988.

Basseville M., Nikiforov I., « *Detection of Abrupt Changes – Theory and Applications* », Information and System Sciences Serie, Prentice Hall, Englewood Cliffs, N.J., 1993.

Basseville M., Cordier M.O., « *Surveillance et diagnostic de systèmes dynamiques: approche complémentaire du traitement de signal et de l'intelligence artificielle* », Rapport INRIA N°2861, 1996.

Basseville M., *« Information Criteria for Residual Generation and Fault Detection and Isolation* », Automatica Vol. 33, N°5, p. 783-803, 1997.

Bernauer E., Demmou H., « Temporal sequence learning with neural networks for process fault détection », *IEEE International Conference on Systems, Man, and Cybernetics, IEEE-SMC 93*, vol. 2, Le Touquet France 1993, p. 375-380.

Bernauer E., Les réseaux de neurones et l'aide au diagnostic : un modèle de neurones bouclés pour l'apprentissage de séquences temporelles, thèse de doctorat, LAAS/Toulouse 1996.

Béroule D., « Un Modèle de Mémoire Adaptative, Dynamique et Associative pour le Traitement de la Parole », Thèse de Doctorat, Paris XI Orsay, 1985.

Berthold M. R., (a) « A Time Delay Radial Basis Function Network for Phoneme Recognition », *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 7, pp. 4470-4473, 1994.

Berthold M. R., (b) « The TDRBF: A Shift invariant radial basis function network », *Proceedings of the fourth Irish Neural Network Conference*, Dublin, pp. 7-12, 1994.

Berthold M. R., Diamond J., « Boosting the Performance of RBF Networks with Dynamic Decay Adjustment » *Advances in Neural Information Processing Systems,* Gerald Tesauro, David S. Touretzky, and Todd K. Leen editors, vol. 7, p. 521-528, MIT Press, Cambridge, MA, 1995.

Bezdek J.C., « *Numerical Taxonomy with Fuzzy Sets* », Journal of Mathematical Biology, Vol.1, p.57-71, 1974.

Billings S.A., and C.F. Fung, «Recurrent Radial Basis Function Networks for Adaptive Noise Cancellation». Neural Networks, Elsevier Science Publishers, Vol. 8, N°2, pp. 273-290, 1995.

Böhme T., Cox C.S, Valentin N., Denoeux T., « Comparaison of Autoassociative Neural Networks and Kohonen Maps for Signal Failure Detection and Reconstruction » In C.H. Dagli et al., editors, Intelligent Engineering Systems through Artificial Neural Networks 9, 637-644, New-York : ASME Press, 1999.

Broomhead D.S., Lowe D., « Multivariable functionnal interpolation and adaptive networks », Complexe Systems, Vol. 2, p. 321-355, 1988.

Burg T., Tschichold N., « Dynamic neurons with negative local feedback for time-series prediction », Proc. Int. Workshop on Advanced Black-box Techniques for Nonlinear Modelling, Katholieke Universiteit Leuven, Belgium, pp. 129-133, 1998.

Camarinha-Matos, L.M., L. Seabra Lopes, and J. Barata «Integration and Learning in Supervision of Flexible Assembly Systems», *IEEE Transactions on Robotics and Automation*, vol. 12, n°2, 1996, p. 202-219.

Campolucci P., Uncini A., Piazza F. et Rao B.D., « *On-Line Learning Algorithms for Locally Recurrent Neural Networks* », IEEE Transactions on Neural Networks, Vol. 10, N°2, pp. 253-271, Mars 1999.

Chang F.J., Liang J.M., Chen Y.C., « Flood Forecasting Using Radial Basis Function Neural Network », IEEE Transactions on Systems, Man and Cybernetics – Part C : Applications and Reviews, Vol. 31, N° 4, November 2001.

Chappelier J.C., RST : une architecture connexionniste pour la prise en compte de relations spatiales et temporelles. Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications de Paris, janvier 1996.

Chappelier J.C., Grumbach A., « A Kohonen Map for Temporal Sequences », *Proceeding of neural Networks and Their Application, NEURAP'96, IUSPIM*, Marseille, mars 1996, p. 104-110.

Chen J., Patton R.J., « *Robust Model-Based Fault Diagnosis for Dynamic Systems* », Kluwer Academic Publisher, Boston, Dordrecht, London 1999.

Combacau M., *Commande et surveillance des systèmes à événements discrets complexes : application aux ateliers flexibles*, thèse de Doctorat, Université P.Sabatier de TOULOUSE 1991.

Combastel C., Méthodes d'aide à la décision pour la détection et la localisation de défauts dans les entraînements électriques Thèse de Doctorat INPG, 2000.

Cussenot C., Surveillance et diagnostic de la chaîne de dépollution d'une automobile, Thèse de doctorat, Université de Rennes 1, 1996.

Daniel O., Les réseaux de Pétri stochastiques pour l'évaluation des attributs de la sûreté de fonctionnement des systèmes manufacturiers, Thèse de doctorat, Laboratoire d'Automatique de Grenoble, Institut National Polytechnique de Grenoble, janvier 1995.

Dash S., Venkatasubramanian V., « Challenges in the industrial applications of fault diagnostic systems », *Proceedings of the conference on Process Systems Engineering Comput. & Chem.* Engng24 (2-7), Keystone, Colorado, July 2000, p. 785-791.

Day S.P., et Davenport M.R., « *Continuous Time Temporal Back Propagation with Adaptable Time Delays* », IEEE Transaction on Neural Networks, Vol. 4, N°2, pp. 348-354, mars 1993.

Demmou H., Bernauer E., « Using Self-Recurrent Neurons for Fault Detection and Diagnosis », 3ème IFAC en WIMS, Roumanie, 1995.

Denoeux T., Govaert G., « Combined supervised and unsupervised learning for system diagnosis using Dempster-Shafer theory », In P. Borne, M. Staroswiecki, J. P. Cassar and S. El Khattabi (Eds) *CESA'96 IMACS Multiconference, Computational Engineering in Systems Applications. Symposium on Control, Optimization and Supervision*, volume 1, pages 104-109, Lille, July 9-12, 1996.

Denoeux T., Masson M., Dubuisson B., « Advanced pattern recognition techniques for system monitoring and diagnosis: a survey », *Journal Européen des Systèmes Automatisés (RAIRO-APII-JESA)*, 31(9-10):1509-1539, 1998.

Desforges X., « Méthodologie de surveillance en fabrication mécanique : application de capteur intelligent à la surveillance d'axe de machine outil », Thèse de Doctorat, Université de Bordeaux I, 1999.

Devauchelle G., Diagnostic mécanique des fatigues sur les structures soumises à des vibrations en ambiance de travail, Thèse de Doctorat, Université de Paris IX Dauphine, 1991.

DeVries B., et J.C. Principe, « A Theory for Neural Networks with Time Delays », Advances in Neural Information Processing Systems, 3, R.P. Lippmann Ed., pp. 162-168, 1991.

Dreyfus G., Martinez J-M., Samuelides M., Gordon M.B., Badran F., Thiria S., Hérault L., *Réseaux de neurone, Méthodologies et Application*, Paris, Edition Eyrolles, 2002

Dubois D., Gentil S., « *Intelligence Artificielle et Automatique* », Revue d'Intelligence Artificielle, Vol. 8, N°1, p. 7-27, 1994.

Dubuisson B., *Diagnostic et reconnaissance des formes*, Editions Hermès, Paris 1990.

Dubuisson B., Boutleux E., Dague P., Denoeux T., Didelet E., Gandvalet Y., Masson M., *Diagnostic, Intelligence Artificielle et reconnaissance des formes*, Editions Hermès, Paris 2001.

Duda R. O., E.H.P., *Pattern classification and scene analysis*, John Wiley & Sons, 1973.

Elman J.L., « Finding Structure in Time », *Cognitive Science*, vol. 14, juin 1990, p. 179-211.

Evsukoff A., Raisonnement approché pour la surveillance des procédés, Thèse de Doctorat, INPGrenoble, 1998

Fache A., Dubois O., Billat A., « On the invertibility of the RBF model in a predictive control strategy », European Symposium on Artificial Intelligence Networks, Bruges-Belgique, 21-23 avril 1999, p. 381-386.

Ferariu L. et Marcu T., « *Evolutionary Design of Dynamic Neural Networks Applied to System Identification* », 15th IFAC World Congress, Barcelona, Spain 2002.

Frank P. M., « Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge Based Redundancy – Asurvey and New Results, Automatica, Vol. 26, p. 459-474, 1990.

Frasconi P., Gori M. et Soda G., « *Local Feedback Multilayered Networks* », Neural Computation, Vol. 4, pp. 120-130, 1992.

Frasconi P., Gori M., Maggini M et Soda G., « *Unified Integration of Explicit Knowledge and Learning by Exemple in Recurrent Networks* », IEEE Transaction on Knowledge and Data Engineering, Vol. 7, N° 2, April 1995.

Frasconi P., Gori M., M. Maggini, G. Soda, « *Representation of Finite State Automata in Recurrent Radial Basis Function Networks*», *Machine Learning* , Vol. 23, pp. 5-32, 1996.

Freitas N., I.M. Macleod and J.S. Maltz,, *« Neural networks for pneumatic actuator fault détection »,* Transactions of the SAIEE*, vol. 90, n° 1, 1999, p. 28-34.*

Frélicot C., *Un système adaptatif de diagnostic prédictif par reconnaissance des formes floues*, thèse de doctorat, Université de technologie de Compiègne, Compiègne 1992.

Fukunaga K., *Statistical Pattern Recognition*, Academic Pess, 2e édition, 1990.

Dunn J.C., « *A Fuzzy Relative of the ISODATA Process an its Use in Detecting Compact Well-Separated Clusters* », Journal of *Cybernetics*, Vol.3, p. 32-57, 1974.

Garcia-Salicetti S., « *Une Approche Neuronale Prédictive pour la Reconnaissance en-ligne de l'Ecriture Cursive* », Thèse de doctorat, Université de Paris VI, Décembre 1996.

Gertler J., J., « Survey of model-based failure detection and isolation in complexe systems », IEEE Control Systems Magazine, Vol. 8, N° 6, p. 3-11, 1988.

Gertler J.J., « *Fault Detection and Diagnosis in Engineering Systems* », Marcel Dekker, Inc., New York, Basel, Hong Kong 1998.

Ghosh J., Beck S., Deuser L., « A Neural Network Based Hybrid System for Detection, Characterization and Classification of Short-Duration Oceanic Signals », *IEEE Jl. of Ocean Engineering*, vol. 17, n° 4, October 1992, p. 351-363.

Ghosh J., Nag A., *Radial Basis Function Network,* in Radial Basis Function Neural Network Theory and Applications, R. J. Howlett and L. C. Jain (Eds), Physica-Verlag., 2000.

Giles C. L., Miller C.B., Chen D., Sun G.Z., Chen H.H. et Lee Y.C., « *Extracting and learning an unknown grammar with recurrent neural networks* », In J.E. Moody, S.J. Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems* 4, pages 317–324, San Mateo, CA, Morgan Kaufmann Publishers, 1992.

Gori M., « *An Extension of BPS* », in Proceeding of the 2nd Intern Workshop on Neural Networks and their Applications, Nimes, France 1989.

Gori M., Bengio Y. et Mori R.D., « *BPS : A Learning Algorithm for Capturing the Dynamic Nature of Speech* », International Joint Conference on Neural Networks, Vol. 2, pp. 417-423, 1989.

Gori M. et Soda G., *« Temporal Pattern Recognition Using EBPS* », EURASIP, 1990.

GRP-SPSF, Rapport du Groupement pour la Recherche en Productique, journée d'étude de l'atelier de Systèmes de Production Sûr de Fonctionnement, 19 et 20 novembre 1998, Besançon/ France.

Hardy R.L., "Multiquadratic equations of topography and other irregular surfaces", J. Geophys. Res, 76:1905-1915,1971.

Hashem S., Keller P.E., Kouzes R.T., Kangas L.J., «Neural Network Based Data Analysis for Chemical Sensor Analysis», *Proceedings of SPIE's AeroSense '95 Conference* , Orlando, Florida, 17-21 April 1995, Forthcoming.

Hassibi B., D.G. Stork et G.J. Wolff, « Optimal Brain Surgeon and General Network Pruning », in Proceedings of the IEEE International Conference on Neural Networks, San Francisco, pp. 293-299, 1993.

Hassoun M. H., *Fundamentals of Artificial Neural Networks*. Cambridge, MA : MIT Press, 1995.

Haykin S., *Neural Networks : a comprehensive foundation*, Prentice Hall International Editions, 2nd edition, ISBN 0-13-908385-5, 1999.

Hebb D.O., « The organization of the behavior », New York :Wiley, 1949.

Hernandez N.G., Système de diagnostic par Réseaux de Neurones et Statistiques : Application à la détection d'hypovigilance d'un conducteur automobile, thèse de doctorat, LAAS/Toulouse, 1999.

Hernandez N.G., Système de diagnostic par Réseaux de Neurones et Statistiques : Application à la détection d'hypovigilance d'un conducteur automobile, thèse de doctorat, LAAS 1999.

Hines J. W., Miller D W., Hajek B. K., « Fault Detection and Isolation: A Hybrid Approach » published in the proceedings of the *1995 American Nuclear Society Annual Meeting and Embedded Topical Meeting on Computer-Based Human Support Systems: Technology, Methods and Future*, Philadelphia, PA, Oct 29-Nov 2, 1995.

Honda K., T. Miyoshi and H. Ichihashi, « *Structural Learning of Recurrent RBF Networks with M-Apoptosis* », Proc. of IEEE International Joint Conference on Neural Networks, pp. 2390-2395, Anchorage/ USA, May, 1998.

Hopfield J.J., « Neural networks and physical Sytems with emergent collective computational abilities », *Proceeding Nat. Acad. Sci. USA, Biophysics*, vol. 79, 1982, p. 2554-2558.

Howell A.J., et Buxton H., « *Recognising Simple Behaviours using Time-Delay RBF Networks* », *Neural Processing Letters* Vol. 5, pp.97-105, 1997.

Howell A.J., et Buxton H., (a) « *Learning Gestures for Visually Mediated Interaction* », *Proc. British Machine Vision Conference (BMVC'98)*, pp.508-517, Southampton, UK, 1998.

Howell A.J., et Buxton H., (b) « *Towards Visually Mediated Interaction using Appearance-Based Models* », *ECCV'98 Workshop on the Perception of Human Action*, Freiburg, Germany, June 1998.

Hubel, D., Wiesel, T. « Ferrier lecture: Functional architecture of macaque monkey visual cortex. » *Proc. Roy. Soc. Lond. B. Biol. Sci.*, 198:1-59. 1977

Hudak M.J., « RCE Classifiers: Theory and Practice » *in Cybernetics and systems*, vol. 23 , 1992, p.483-515.

Hutchinson J.M., A Radial Basis Function Approach to Financial Time Series Analysis, Thèse de doctorat, Massachusetts Institute of Technology (MIT), 1994.

Hwang Y.S., Bang S.Y., « An efficient method to construct a Radial Basis Function Neural Network classifier », *Neural Networks*, vol. 10, n° 08, 1997, p. 1495-1503.

Isermann R., *« Process Fault Detection Based on Modeling and Estimation Methods – A Survey* », Automatica, Vol. 20, N°4, p. 387-404, 1984.

Jacquemin C., « A Temporal Connectionnist Approach to Natural Language », Sigart Bulletin, Vol. 5, N°03, 1994.

Jordan M.I., (a) « Serial order: a parallel distributed processing approach », University of California, Institute for cognitive science, 1986.

Jordan M.I., (b) « Attractor Dynamics and Parallelism in a Connectionist Sequential Machine », Proceedings of the 8thAnnual Conference of the Cognitive Science Society, pp. 531-546, 1986

Katsillis G., Chantler M.J., « Can dependency-based diagnosis cope with simultaneous equations ? », Working notes of the 8th International Workshop on Principles of Diagnosis DX'97, Mont-Saint-Michel, France 1997.

Keller P., Kouzes R.T., Kangas L.J., « Three Neural Network Based Sensor System for Environemental Monitoring », *Proceedings IEEE Electro94 Conference*, Boston, MA, USA, May 1994

Knudsen, E., Konishi, M., « Mechanisms of sound localization in the barn owl (tyto alba) » *Journal of Compartive Physiology*, 133:13-21. 1979.

Koch, C., « *Biophysics of Computation: Information Processing in Single Neurons* », Oxford University Press, 1999.

Kohonen, T. « *Self-Organization and associative memory*. » Heidelberg: Springer-Verlag, Berlin, 3rd edition. 1989

Koivo H.N, « artificial neural networks in fault diagnosis and control », *control in engineering practice*, vol.2, n°1, 1994, p. 89-101.

Lang K, M. Witbrock « Learning to Tell Two Spirals Apart », in proceeding of connectionist models summer school, 1988.

Le Cun Y., « Une procédure d'apprentissage pour réseau à seuil asymétrique »,Cognitiva, Vol. 85, p. 599-604, 1985.

Le Cun Y., J.S. Denker et S.A. Solla « Optimal Brain Damage », in Advances in Neural Information Processing Systems 2, Ed. D.S. Touretzsky, Morgan Kaufmann, pp. 598-605, 1990.

Lefebvre D., *Contribution à la modélisation des systèmes dynamiques à événements discrets pour la commande et la surveillance*, Habilitation à Diriger des Recherches, Université de Franche Comté/ IUT Belfort - Montbéliard 2000.

Legrand J.F., et Garda P., « *Prédiction de Trafic GSM par Méthodes Connexionnistes* », XIémes Journées Neurosciences et Science pour l'Ingénieur NSI'2002.

Liu, Y.H. et Wang, X.J., « *Spike-frequency adaptation of a generalized leaky integrate and fire model neuron* », *J. Comp. Neurosci.*, N°10, pp. 25-45, 2001.

Loiez E., Contribution au diagnostic de systèmes analogique, Thèse de doctorat de l'Université des sciences et technologies de Lille, France 1997.

Lopes N., Ribeiro B., « Part Quality Prediction in an Injection Moulding Process Using Neural Networks », in proceedings of WMC, ISM,1999.

Mackey M. et Glass L., « *Oscillations and Chaos in Physiological Control System* », Science, pp. 197-287, 1977.

MacQueen J., « *Some methods for classification and analysis of multivariate observations* », Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability, volume 1, pages 281-297, University of California Press, Berkeley, 1967.

Mak M. W., « *A Learning Algorithm for Recurrent Radial Basis Function Networks* », Neural Processing Letters, Vol. 2, No. 1, pp. 27-31, January 1995.

Mak M.W., Kung S.Y., « Estimation of Elliptical Basis Function Parameters by the EM Algorithms with Application to Speaker Verification », *IEEE Trans. on Neural Networks*, vol. 11, n° 4, July 2000, p. 961-969.

Mangeas M., « *Propriétés Statistiques des Modèles Paramétriques non Linéaires de Prévisions de Séries Temporelles, Application aux Réseaux de Neurones à Propagation Directe* » Thèse de Doctorat, SAMOS / Université Paris I, 1996.

Masson M.H., Dubuisson B., Frélicot C., « *Conception d'un module de reconnaissance des formes floues pour le diagnostic* », Journal Européen des Systèmes Automatisés (*RAIRO-APII-JESA*), p. 319-341, 1996.

McCulloch W.S., Pitts W., « A logical calculus of the ideas immanent in nervous activity », Bulletin of Mathematical Biophysics, Vol. 5, p. 115-133, 1949.

Meador J., Wu A., Tseng H.,. Lin T.S, « Fast Diagnosis of Integrated Circuit Faults Using Feedforward Neural Network », IEEE International Joint Conference on Neural Networks, Seattle, July, 1991.

Mellouk A., « *Développement d'un système hybride neuro-prédictif : application à la reconnaissance de la parole continue* », Thèse de Doctorat, LIP6/ Université de Paris 6 Octobre 1994.

Micchelli C.A., « *Interpolation of scattered data: distance matrices and conditionally positive definite functions* », Contsructive Approximation, N°2, pp. 11-22, 1986.

Mirea L. et Marcu T., « *System Identification Using Functional Link Neural Networks with Dynamic Structure* », 15th IFAC World Congress, Barcelona, Spain 2002.

Miyoshi T., H.Ichihashi, S.Okamoto and T.Hayakawa, « *Learning Chaotic Dynamics in Recurrent RBF Network* », Proc. of IEEE ICNN'95, pp. 588-593, Perth/ Western Australia1995.

Moakes P. A., and S. W. Beet., « Non-linear speech analysis using recurrent radial basis function networks», In Neural Networks for Signal Processing IV, eds.: J. Vlontzos, J-N. Hwang and E. Wilson, pp 319-328. IEEE Press, 1994.

Monostori L., « AI and Machine Learning Techniques for Managing Complexity, Changes and uncertainties in Manufacturing », 15^{éme} IFAC World Congress on Automatic Control, Barcelone, Espagne juillet 2002.

Moody J., Darken J., « Fast Learning in networks of locally tuned processing units », *Neural Computation*, 1989, vol. 1, p. 281-194.

Mozer M. C., « *Neural network architectures for temporal pattern processing* », In A. S. Weigend & N. A. Gershenfeld (Eds.), *Time series prediction: Forecasting the future and understanding the past* (pp. 243-264). Redwood City, CA: Sante Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XVII, Addison-Wesley Publishing, 1993.

Mustawi M.T., Ahmed W., Chan K.H., Faris K.B., Hummels D.M., « on the training of Radial Basis Function Classifiers », *Neural Networks*, vol. 5, 1992, p. 595-603.

Pal S.K., Majumder D., « *Fuzzy Set and Decision Making Approaches in Voweland Speaker Recognition* », IEEE Transactions on Systems, Man, and Cybernetics, Vol. 7, p. 625-629,1977.

Patton R., Frank P., *Clark R., Fault Diagnosis in Dynamic Systems : Theory and Application*, International Series in Systems and Control Engineering, Prentice Hall International, London, UK 1989.

Pearlmutter B.A., « *Dynamic Recurrent Neural Networks* », CMU-CS-90,196, Carnegie Mellon University, School of Computer Science, décembre 1990.

Petsche T.A., Marcontonio A., Darken C., Hanson S.J., M.kuh G., Santoso I., *A Neural Network autoassociator for induction motor failure prediction*, Cambridge: MIT Press, Edition D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, Advances in Neural Information Prodessing Systems 8, 1996, p. 924-930.

Pineda F.J., « *Generalization of Backpropagation to recurrent Neural Networks* », Physical Review Letters, N° 59, pp. 2229-2232, 1987.

Pineda F.J., « Dynamics and Architecture for Neural Computation », Journal of Complexity, Vol. 4, pp. 216-245, 1988.

Ploix JL., et G. Dreyfus, « *Early fault detection in a distillation column: an industrial application of knowledge-based neural modelling* »**,** *Neural Networks: Best Practice in Europe,* B. Kappen, S. Gielen, eds, pp. 21-31 (World Scientific, 1997).

Poddar P. et Unnikrishnan K.P., (a) « *Nonlinear Prediction of Speech Signals Using Memory Neuron Networks* », Neural Networks for Signal Processing I, B. H. Juang, S. Y. Kung and C. A. Kamm, Eds. IEEE Press, 1991.

Poddar P. et Unnikrishnan K.P., (b) « *Memory Neuron Networks : A Prolegomenon* », General Motors Research Laboratories Report GMR-7493, October 21, 1991.

Poggio T., Girosi F., A Theory of Networks for Approximation and Learning, AI Memo N°1140, Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center for Biological Information Processing Whitaker College, July 1989

Poulard H., statistiques et réseaux de neurones pour un système de diagnostic. Application au diagnostic de pannes automobiles, Thèse de Doctorat, LAAS/France, 1996.

Powell M.J.D. « Radial Basis Functions for multivariable interpolation : a review » J.C. Masson and M.G. Gox editors, algorithms for approximation. Clarendon Press, Oxford, 1987.

Racoceanu D., Zerhouni N., Addouche N., « Modular Modeling and Analysis of a Distributed Production System with Distant Specialised Maintenance », *IEEE ICRA 2002, Washington, USA, 11-15 mai 2002*.

Renals S., and Rohwer R., « Phoneme classification experiments using radial basis functions » Proceedings of the international Joint Conference on Neural Networks, p. I-461-I-467, Washington D.C., June 1989. IEEE TAB Neural Network Committee.

Rengaswamy R., Venkatasubramanian V., « A Syntactic Pattern Recognition Approach for Process Monitoring and Fault Diagnosis », *Engineering Applications of Artificial Intelligence Journal*, 8(1), 1995, p. 35-51.

Robinson A., et F. Fallside, « The Utility Driven Dynamic Error Propagation Network », Combridge Univ. Eng. Dept., Tech . Rep. CUED/F-IN-FENG/TR.1, 1987.

Rosenblatt F., « The Perceptron :a probabilistic model for information storage in the brain », Psychological Review, Vol. 65, p. 386-408, 1958.

Rohwer R., Forrest B., « Training Time-Dependence in Neural Network », *in IEEE First International Conference on Neural Networks*, M. Caudill et C.Butler, vol. 2, San Diego, California, juin 1987, p. 701-708.

Rojas R., *Neural Networks : A Systematic Introduction,* Springer-Verlag Berlin Heidelberg, 1996.

Rummelhart D.E, Hinton G.E., Williams R.J., « Learning Internal Representation by Error Propagation », *in Parallel Distributed Processing Explorations in the Microstructure of Cognition*, vol. 1, The MIT Press/Bradford Books, D.E. Rumelhart and J.L.McClelland, 1986, p. 318-362.

Rynkiewi J., « *Modèles Hybrides Intégrant des Réseaux de Neurones Artificiels à des Modèles de Chaînes de Markov Cachées : Application à la Prédiction de Séries Temporelles* », Thèse de Doctorat, SAMOS / Université de Paris I, 2000.

Sakai M., Homma N. et Abe K., « *A Statistical Approximation Learning Method for Simultaneous Recurrent Networks* », 15th IFAC World Congress, Barcelona, Spain 2002.

Schagen I.P., « *Interpolation in two dimensions – a new technique* ». J. Inst Math. Appl., Vol. 23, pp. 53-59, 1979.

Schmidhuber J., « A Fixed Storage $O(N^3)$ Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks », Neural Computation, Vol. 4, N°2, pp. 243-248, 1992.

Sejnowski T.J., Rosenberg C.R., NetTalk: a parallel network that learns to read aloud, electrical engineering and computer science technical report, the johns hopkins university, 1986.

Shioya S., Huang J.H., Shimizu H., « Online Fault Detection in Virginiamycin Production », 15éme IFAC World Congress on Automatic Control, Barcelone, Espagne juillet 2002.

Simpson P.K., « Fuzzy min-max neural networks – Part I : Classification » *IEEE Transaction on Neural Networks*, Vol.3, N°5, pp. 776-786, September 1992.

Simpson P.K., « Fuzzy min-max neural networks – Part II : Clustering » *IEEE Transaction on Fuzzy Systems*, Vol.1, N°1, pp. 32-45, February 1993.

Singh S., « Neural Network Separation of Temporal Data », Proceeding of IEEE International Joint Conference on Neural Networks (IJCNN'99), Washington D.C., 10-16 July, 1999.

Sinha M., Gupta M.M. et Nikiforuk P.N., « *Hybrid Neural Models for Time Series Forecasting* », 15th IFAC World Congress, Barcelona, Spain 2002.

Smyth P., « detecting novel fault conditions with hidden Markov models and neural netwoks », *Pattern Recognition in Practice IV*, 1994, p. 525-536.

Sorel M., et Sima J., « *Robust implementation of finite automata by recurrent RBF networks*», Proceedings of the SOFSEM Seminar on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic, 431-439, Berlin: Springer-Verlag, LNCS 1963, 2000.

Sun G.Z., H.H. Chen et Y.C.Lee, « Green's Function Method for Fast on-line Learning Algorithm of Recurrent Neural Networks », Advances Neural Information Processing System, Vol. 4, pp. 333-340, 1990.

Svarer C., L.K. Hansen et J. Larsen, « On Design and Evaluation of Tapped Delay Line Networks », in Proceedings of the IEEE International Conference on Neural Networks, San Francisco, pp. 46-51, 1993.

Terstyanszky G., Kovacs L., « Improving Fault Diagnosis Using Proximity and Homogeneity Measure », 15éme IFAC World Congress on Automatic Control, Barcelone, Espagne juillet 2002.

Toguyeni A.K.A., *Surveillance et diagnostic en ligne dans les ateliers flexibles de l'industrie manufacturière*, Thèse de doctorat, Université de Lille 1992.

Toomarian N., et J., Barhen, « Adjoint-Functions and Temporal Learning Algorithms in Neural Networks », Advances in Neural Information Processing Systems, Vol. 3, pp. 113-120, 1991.

Tromp L., *Surveillance et Diagnostic de systèmes industriels complexes : une approche hybride Numérique/Symbolique*, Thèse de Doctorat, Université de Rennes1/IRISA, 2000.

Tsoi C.T., Back A.D., « *Locally Recurrent Globally Feedforward Networks : A Critical Review of Architectures* », IEEE Transaction on Neural Networks Vol.05, pp. 229-239, 1994.

Tyan C. Y., Wang P. P., Bahler D., « Neural Fault Diagnosis and Fuzzy Fault Control for a Complex Linear Dynamic System, » Published in the book of series of *Advances in Fuzzy Theory and Technology, Volume II*, ISBN: 0-9643456-1-7, pp. 357-375, 1994.

Urbani D., « *Méthodes statistiques pour la sélection d'architectures neuronales : application à la modélisation de processus dynamiques* », Thèse de Doctorat, ESPCI/ Université Pierre et Marie Curie - Paris VI, Novembre 1995.

Vaucher G., « Un Modèle de Neurone Artificiel Conçu pour l'Apprentissage non Supervisé de Séquences d'Evénements Asynchrones », Revue VALGO, ISSN 1243-4825, Vol. 1, pp. 66-107, ACTH 1993.

Vemuri A., Polycarpou M., « Neural Network Based Robust Fault Diagnosis in Robotic Systems », *IEEE Transactions on Neural Networks*, vol. 8, n°. 6, novembre 1997, p. 1410-1420,.

Vemuri A., Polycarpou M., Diakourtis S., « Neural Network Based Fault Detection and Accommodation in Robotic Manipulators », *IEEE Transactions on Robotics and Automation*, vol. 14, n° 2, avril 1998, p. 342-348.

Villemeur A., *Sûreté de fonctionnement des systèmes industriels,* Edition EYROLLES, Collection DER-EDF, Volume 67, 1988.

Waibel A., Hanazawa T., Hinton G., Shikano K., Lang K., « Phoneme recognition using time delay neural network » *IEEE Trans. in Acoustics, Speech and Signal Processing,* vol. 37, n° 3, p. 328-339, 1989.

Warwick K., Irwin G.W., Hunt K.J., « *Neural Networks for Control and Systems* », IEE Control Engineering Series 46, Peter Peregrinus Ltd., London, United Kingdom, 1992.

Washio T., Hotoda H., « *Discovering admissible simultaneous equations of large scale systemes* », 15th National Conference on Artificial Intelligence AAAI-98, Madison, WI, Etats-Unis, p. 189-196, 1998.

Weber P., diagnostic de procédés par l'analyse des estimations paramétriques de modèles de représentation à temps discret, Thèse de Doctorat, INPG, 1999.

Werbos P.J., Beyond regression: New tools for prediction and analysis in the behavioral science, Thèse de doctorat, Harvard University, 1974.

Werbos P., « Backpropagation Trough time : What it does and how to do it », Proceedings IEEE, Vol. 78, 1990.

Widrow B., Hoff M. E., « Adaptive switching circuits », dans 1960 IRE WESCON Convention Record, New York : IRE, p. 96-104, 1960.

Williams R.J., Zipser D., « A Learning Algorithm for Continually Running Fully Recurrent Neural Networks », *Neural Computation*, vol.1, juin 1989, p. 270-280.

Willsky A.S., « *A Survey of Design Methods for Failure Detection in Dynamic Systems* », Automatica, Vol. 12, p. 601-611, 1976.

Wu A., Meador J., « A Measurement Selection for Parametric IC Fault Diagnosis, » *Journal of Electronic Testing: Theory and Applications,* Kluwer Academic Publishers, Vol. 5, No. 1, pp. 9- 18, Feb. 1994.

Xu L., « RBF nets, mixture experts, and Bayesian Ying-Yang learning », *Neurocomputing*, 1998, vol. 19, N° 1-3, p. 223-257.

Yu W.S., et Wang G.C., « *Adaptive Control Design Using Delayed Dynamical Neural Networks for a Class of Nonlinear Systems* », Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'01), Seoul, Korea, Mai 2001.

Zemouri R., Racoceanu D., Zerhouni N., « The RRBF : Dynamic representation of time in radial basis function network » Proc. of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'2001, vol. 2, pp.737-740, 15-18 octobre 2001, Antibes, Juan les Pins.

Zemouri R., Racoceanu D., Zerhouni N. – a – « From the spherical to an elliptic form of the dynamic RBF neural network influence field » *World Congress on Computational Intelligence, International Joint Conference on Neural Networks (IJCNN)*, Honolulu, Hawaii, USA, May 12-17, 2002.

Zemouri R., Racoceanu D., Zerhouni N. – b – « *Réseaux de neurones Récurrents à Fonction de base Radiales :RRFR/ Application au pronostic* », Revue d'Intelligence Artificielle, RSTI série RIA, Vol. 16, N°03, 2002.

Zemouri R., Racoceanu D., Zerhouni N. – c – « *Application of the dynamique RBF Network in a monitoring problem of the production systems* », 15^{éme} IFAC World Congress on Automatic Control, Barcelone, Espagne, juillet 2002.

Zemouri R, Racoceanu D., Zerhouni N. – a – « *Réseaux de neurones récurrents à fonctions de base radiales RRFR : Application à la surveillance dynamique* », Revue Systèmes /JESA, Vol. 37, N°1, pp. 49-81, 2003.

Zemouri R., Racoceanu D., Zerhouni N. – b – « *Recurrent Radial Basis Function network for Time-Series Prediction* », Engineering Applications of Artificial Intelligence (Elsevier Science), Volume 16, Issue 5-6, pp. 453-463, Novembre 2003.

Zhang Q., Identification et Surveillance de systèmes Dynamiques, Habilitation à diriger les recherches, Université de Rennes1, Institut de Formation Supérieure en Informatique et en Communication, 1999.

Zwingelstein G., *Diagnostic des défaillances : Théorie et pratique pour les systèmes industriels*, Edition HERMES 1995.

Thèse de Monsieur Mohamed-Ryad ZEMOURI

Contribution à la surveillance des systèmes de production à l'aide des réseaux de neurones dynamiques : Application à la e-maintenance.

Résumé : Les méthodes de surveillance industrielle sont divisées en deux catégories : méthodes de surveillance avec modèle formel de l'équipement, et méthodes de surveillance sans modèle de l'équipement. Les modèles mathématiques formels des équipements industriels sont souvent entachés d'incertitudes et surtout difficiles à obtenir. Cette thèse présente l'application des réseaux de neurones artificiels pour la surveillance d'équipements industriels. Nous proposons une architecture de Réseaux à Fonctions de base Radiales qui exploite les propriétés dynamiques des architectures localement récurrentes pour la prise en compte de l'aspect temporel des données d'entrée. En effet, la prise en compte de l'aspect dynamique nécessite des architectures de réseaux de neurones particulières avec des algorithmes d'apprentissage souvent compliqués. Dans cette optique, nous proposons une version améliorée de l'algorithme des k-moyennes qui permet de déterminer aisément les paramètres du réseau de neurones. Des tests de validation montrent qu'à la convergence de l'algorithme d'apprentissage, le réseau de neurones se situe dans la zone appelée « zone de bonne généralisation ». Le réseau de neurones a été ensuite décomposé en fonctions élémentaires facilement interprétables en langage automate. La partie applicative de cette thèse montre qu'un traitement de surveillance en temps réel est possible grâce aux architectures à automates programmables industriels. Le réseau de neurones chargé dans l'automate est entièrement configurable à distance par le protocole de communication TCP/IP. Une connexion Internet permet alors à un expert distant de suivre l'évolution de son équipement et également de valider l'apprentissage du réseau de neurones artificiel.

Mots-Clés : surveillance, détection de dégradation, diagnostic, e-maintenance, réseaux de neurones dynamiques, réseaux de neurones localement récurrents, apprentissage.

Contribution to the production system monitoring using dynamic neural networks : Application to the e-maintenance.

Abstract : The industrial monitoring methods are divided into two categories: monitoring methods based on the existence of the equipment formal model, and those which not use any equipment formal model. Generally, there are many uncertainties in the formal model and for complex industrial equipment, it is very difficult to obtain a correct mathematical model. This thesis presents an application of the artificial neural networks to the industrial monitoring. We propose a new architecture of Radial Basis Function Networks which exploits the dynamic properties of the locally recurrent architectures for taking into account the input data temporal aspect. Indeed, the consideration of the dynamic aspect requires rather particular neural networks architectures with special training algorithms which are often very complicated. In this sense, we propose an improved version of the k-means algorithm which allows to determine easily the neural network parameters. The validation tests show that at the convergence of the learning algorithm, the neural network is situated in the zone called « good generalization zone ». The neural network was then decomposed into elementary functions easily interpretable in industrial automation languages. The applicative part of this thesis shows that a real-time monitoring treatment is possible thanks to the automation architectures. The neural network loaded in a PLC is completely configurable at distance by the TCP/IP communication protocol. An Internet connection allows then a distant expert to follow the evolution of its equipment, and also to validate the artificial neural network learning.

Key Words : Monitoring, degradation detection, diagnosis, e-maintenance, dynamic neural networks, locally recurrent neural networks, learning.